T.C.
İSTANBUL ÜNİVERSİTESİ-CERRAHPAŞA
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

ÇEŞİTLİ MATEMATİKSEL PROBLEMLERİN ÇÖZÜMÜNDE PROGRAMLAMA DİLLERİNİN VE PLATFORMLARININ KARŞILAŞTIRILMASI

Uğur Osman YÜCEL

DANIŞMAN
Doç. Dr. Rüya ŞAMLI

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

İSTANBUL-2020

Bu çalışma, 26.06.2020 tarihinde aşağıdaki jüri tarafından Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Mühendisliği Programında Yüksek Lisans tezi olarak kabul edilmiştir.

Tez Jürisi

Doç. Dr. Rüya ŞAMLI(Danışman)
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Doç. Dr. Selçuk SEVGEN
İstanbul Üniversitesi - Cerrahpaşa
Mühendislik Fakültesi

Dr. Öğr. Üyesi Zeynep TURGUT
Haliç Üniversitesi
Mühendislik Fakültesi

Unvan
Bir öğe seçin.
Bir öğe seçin.

Unvan Bir öğe seçin.
Bir öğe seçin.
Bir öğe seçin.

20.04.2016 tarihli Resmi Gazete'de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince; Bu Lisansüstü teze, İstanbul Üniversitesi-Cerrahpaşa'nın abonesi olduğu intihal yazılım programı kullanılarak Lisansüstü Eğitim Enstitüsü'nün belirlemiş olduğu ölçütlere uygun rapor alınmıştır.

ÖNSÖZ

Bu tez çalışmasının gerçekleştirilmesinde, tecrübelerini ve bilgilerini benimle paylaşan, benden hiçbir zaman ümidini kesmeyen değerli hocam Doç. Dr. Rüya ŞAMLI'ya sonsuz teşekkürlerimi sunarım. Tez dönemim boyunca yaşadığım her zorlukta benimle birlikte olan ve beni destekleyen annem Fatma YÜCEL başta olmak üzere babam Mehmet Fatih YÜCEL ve kardeşim Hatice Zeynep YÜCEL'e teşekkürü bir borç bilirim. Bu süreçte bana olan yardımlarından dolayı kız arkadaşıma, arkadaşlarıma ve iş arkadaşlarıma da teşekkür ederim.

Haziran 2020 Uğur Osman YÜCEL

İÇİNDEKİLER

Sayfa No

ÖNSÖZ ..... **iv**
İÇİNDEKİLER ..... **v**
ŞEKİL LİSTESİ ..... **viii**
TABLO LİSTESİ ..... **ix**
SİMGE VE KISALTMA LİSTESİ ..... **xiv**
ÖZET ..... **xv**
SUMMARY ..... **xvi**
1. GİRİŞ **1**
2. GENEL KISIMLAR **2**
2.1. GENEL MATEMATİKSEL ALGORİTMALAR 2
2.1.1. Faktöriyel Hesaplama Algoritması 2
2.1.2. Üs Alma Algoritması 3
2.1.3. Asal Sayı Bulma Algoritması 4
2.2. ARAMA ALGORİTMALARI 6
2.2.1. Doğrusal Arama Algoritması 7
2.2.2. İkili Arama Algoritması 8
2.3. SIRALAMA ALGORİTMALARI 9
2.3.1. Eklemeli Sıralama Algoritması 9
2.3.2. Seçmeli Sıralama Algoritması 10
2.3.3. Kabarcık Sıralama Algoritması 12
2.3.4. Çabuk Sıralama Algoritması 13
2.3.5. Birleştirme Sıralama Algoritması 14
2.3.6. Gnome Sıralama Algoritması 15
2.3.7. Yığın Sıralama Algoritması 16
2.4. POLİNOMSAL OLMAYAN ALGORİTMALAR 18
2.4.1. Gezgin Satıcı Problemi Çözüm Algoritması 19
2.4.2. Sırt Çantası Problemi Çözüm Algoritması 19
2.4.3. Hanoi Kuleleri Problemi Çözüm Algoritması 20
2.5. LİTERATÜR TARAMASI 20
3. MALZEME VE YÖNTEM **24**

3.1. PROGRAMLAMA DİLLERİ....24
3.1.1. C Programlama Dili....24
3.1.2. C# Programlama Dili....24
3.1.3. Java Programlama Dili....25
3.1.4. Python Programlama Dili....25
3.2. PROGRAMLAMA PLATFORMLARI....25
3.2.1. Visual Studio....26
3.2.2. Code::Blocks....27
3.2.3. NetBeans....28
3.2.4. Jet Brains PyCharm....29
3.3. ALGORİTMA TASARLAMA YAKLAŞIMLARI....30
3.3.1. İteratif Algoritma Tasarımı....30
3.3.2. Rekürsif Algoritma Tasarımı....30
3.3.3. Geri İzlemeli Algoritma Tasarımı (Backtracking)....30
4. BULGULAR....31
4.1. GENEL MATEMATİKSEL ALGORİTMALARIN UYGULAMALARININ PERFORMANS ANALİZİ....33
4.1.1. Faktöriyel Hesaplama Algoritması Uygulamasının Performans Analizi....33
4.1.2. Üs Alma Algoritması Uygulamasının Performans Analizi....36
4.1.3. Asal Sayı Bulma Algoritması Uygulamasının Performans Analizi....39
4.2. ARAMA ALGORİTMALARININ UYGULAMALARININ PERFORMANS ANALİZİ....41
4.2.1. Doğrusal Arama Algoritması Uygulamasının Performans Analizi....41
4.2.2. İkili Arama Algoritması Uygulamasının Performans Analizi....43
4.3. SIRALAMA ALGORİTMALARININ UYGULAMALARININ PERFORMANS ANALİZİ....44
4.3.1. Eklemeli Sıralama Algoritması Uygulamasının Performans Analizi....45
4.3.2. Seçmeli Sıralama Algoritması Uygulamasının Performans Analizi....47
4.3.3. Kabarcık Sıralama Algoritması Uygulamasının Performans Analizi....48
4.3.4. Çabuk Sıralama Algoritması Uygulamasının Performans Analizi....50
4.3.5. Birleştirmeli Sıralama Algoritması Uygulamasının Performans Analizi....52
4.3.6. Gnome Sıralama Algoritması Uygulamasının Performans Analizi....53
4.3.7. Yığın Sıralama Algoritması Uygulamasının Performans Analizi....55
4.4. POLİNOMSAL OLMAYAN ALGORİTMALARIN PERFORMANS ANALİZİ....56
4.4.1. Gezgin Satıcı Problemi Çözüm Algoritması Uygulamasının Performans Analizi....57

4.4.2. Sırt Çantası Problemi Çözüm Algoritması Uygulamasının Performans Analizi....58
4.4.3. Hanoi Kuleleri Problemi Çözüm Algoritması Uygulamasının Performans Analizi....60
5. TARTIŞMA VE SONUÇ 62
KAYNAKLAR....65
ÖZGEÇMİŞ 68

ŞEKİL LİSTESİ

Sayfa No

Şekil 2.1: 7 sayısı için Doğrusal Arama Algoritması çalışması 7

Şekil 2.2: 7 Sayısı için İkili Arama Algoritması Çalışması.....8

Şekil 2.3: Örnek bir tamsayı dizisi için Eklemeli Sıralama Algoritması Çalışması.....10

Şekil 2.4: Örnek bir tamsayı dizisi için Seçmeli Sıralama Algoritması Çalışması 11

Şekil 2.5: Örnek bir tamsayı dizisi için Kabarcık Sıralama Algoritması Çalışması.....12

Şekil 2.6: Örnek bir tamsayı dizisi için Çabuk Sıralama Algoritması Çalışması.....13

Şekil 2.7: Örnek bir tamsayı dizisi için Birleştirme Sıralama Algoritaması Çalışması 15

Şekil 2.8: Örnek bir tamsayı dizisi için Gnome Sıralama Algoritması Çalışması 16

Şekil 2.9: Örnek bir tamsayı dizisi için Yığın Sıralama Algoritması Çalışması 17

Şekil 3.1: Visual Studio Platformundan Alınan Ekran Görüntüsü.....26

Şekil 3.2: Code::Blocks Platformundan Alınan Ekran Görüntüsü.....27

Şekil 3.3: NetBeans Platformundan Alınan Ekran Görüntüsü 28

Şekil 3.4: JetBrains PyCharm Platformundan Alınan Ekran Görüntüsü.....29

TABLO LİSTESİ

Sayfa No

Tablo 2.1: 4 Sayısı için İteratif Faktöriyel Hesaplama Algoritması ile Çözüm Örneği 3

Tablo 2.2: 4 Sayısı için İteratif Faktöriyel Hesaplama Algoritması ile Çözüm Örneği 3

Tablo 2.3: 2^3 Sayısı için İteratif Üs Alma Algoritması ile Çözüm Örneği.....4

Tablo 2.4: 2^3 Sayısı için Rekürsif Üs Alma Algoritması ile Çözüm Örneği.....4

Tablo 2.5: 6 Sayısının Asal Olup Olmaması Durumunun Adım Adım Kontrolü 5

Tablo 2.6: 7 Sayısının Asal Olup Olmaması Durumunun Adım Adım Kontrolü 5

Tablo 2.7: 2 ile 10 Arasındaki Sayılar için Asal Sayı Bulma Algoritması ile Çözüm Örneği 6

Tablo 4.1: İteratif Faktöriyel Hesaplama Algoritması için C Programlama Dili Performans Analizi. 34

Tablo 4.2: Rekürsif Faktöriyel Hesaplama Algoritması için C Programlama Dili Performans Analizi. 34

Tablo 4.3: İteratif Faktöriyel Hesaplama Algoritması için C# Programlama Dili Performans Analizi. 34

Tablo 4.4: Rekürsif Faktöriyel Hesaplama Algoritması için C# Programlama Dili Performans Analizi. 35

Tablo 4.5: İteratif Faktöriyel Hesaplama Algoritması için Java Programlama Dili Performans Analizi. 35

Tablo 4.6: Rekürsif Faktöriyel Hesaplama Algoritması için Java Programlama Dili Performans Analizi. 35

Tablo 4.7: İteratif Faktöriyel Hesaplama Algoritması için Python Programlama Dili Performans Analizi. 36

Tablo 4.8: Rekürsif Faktöriyel Hesaplama Algoritması için Python Programlama Dili Performans Analizi. 36

Tablo 4.9: İteratif Üs Alma Algoritması için C Programlama Dili Performans Analizi. 37

Tablo 4.10: Rekürsif Üs Alma Algoritması için C Programlama Dili Performans Analizi. 37

Tablo 4.11: İteratif Üs Alma Algoritması için C# Programlama Dili Performans Analizi. 37

Tablo 4.12: Rekürsif Üs Alma Algoritması için C# Programlama Dili Performans Analizi. 38

Tablo 4.13: İteratif Üs Alma Algoritması için Java Programlama Dili Performans Analizi. 38

Tablo 4.14: Rekürsif Üs Alma Algoritması için Java Programlama Dili Performans Analizi. 38

Tablo 4.15: İteratif Üs Alma Algoritması için Python Programlama Dili Performans Analizi. 39

Tablo 4.16: Rekürsif Üs Alma Algoritması için Python Programlama Dili Performans Analizi. 39

Tablo 4.17: Asal Sayı Bulma Algoritması için C Programlama Dili Performans Analizi. 40

Tablo 4.18: Asal Sayı Bulma Algoritması için C# Programlama Dili Performans Analizi. 40

Tablo 4.19: Asal Sayı Bulma Algoritması için Java Programlama Dili Performans Analizi. 40

Tablo 4.20: Asal Sayı Bulma Algoritması için Python Programlama Dili Performans Analizi. 41

Tablo 4.21: Doğrusal Arama Algoritması için C Programlama Dili Performans Analizi. 41

Tablo 4.22: Doğrusal Arama Algoritması için C# Programlama Dili Performans Analizi. 42

Tablo 4.23: Doğrusal Arama Algoritması için Java Programlama Dili Performans Analizi. 42

Tablo 4.24: Doğrusal Arama Algoritması için Python Programlama Dili Performans Analizi. 42

Tablo 4.25: İkili Arama Algoritması için C Programlama Dili Performans Analizi. 43

Tablo 4.26: İkili Arama Algoritması için C# Programlama Dili Performans Analizi. 43

Tablo 4.27: İkili Arama Algoritması için Java Programlama Dili Performans Analizi. 44

Tablo 4.28: İkili Arama Algoritması için Python Programlama Dili Performans Analizi. 44

Tablo 4.29: Eklemeli Sıralama Algoritması için C Programlama Dili Performans Analizi. 45

Tablo 4.30: Eklemeli Sıralama Algoritması için C# Programlama Dili Performans Analizi. 46

Tablo 4.31: Eklemeli Sıralama Algoritması için Java Programlama Dili Performans Analizi. 46

Tablo 4.32: Eklemeli Sıralama Algoritması için Python Programlama Dili Performans Analizi. 46

Tablo 4.33: Seçmeli Sıralama Algoritması için C Programlama Dili Performans Analizi. 47

Tablo 4.34: Seçmeli Sıralama Algoritması için C# Programlama Dili Performans Analizi. 47

Tablo 4.35: Seçmeli Sıralama Algoritması için Java Programlama Dili Performans Analizi. 48

Tablo 4.36: Seçmeli Sıralama Algoritması için Python Programlama Dili Performans Analizi. 48

Tablo 4.37: Kabarcık Sıralama Algoritması için C Programlama Dili Performans Analizi. 49

Tablo 4.38: Kabarcık Sıralama Algoritması için C# Programlama Dili Performans Analizi. 49

Tablo 4.39: Kabarcık Sıralama Algoritması için Java Programlama Dili Performans Analizi. 49

Tablo 4.40: Kabarcık Sıralama Algoritması için Python Programlama Dili Performans Analizi. 50

Tablo 4.41: Çabuk Sıralama Algoritması için C Programlama Dili Performans Analizi. 50

Tablo 4.42: Çabuk Sıralama Algoritması için C# Programlama Dili Performans Analizi. 51

Tablo 4.43: Çabuk Sıralama Algoritması için Java Programlama Dili Performans Analizi. 51

Tablo 4.44: Çabuk Sıralama Algoritması için Python Programlama Dili Performans Analizi. 51

Tablo 4.45: Birleştirmeli Sıralama Algoritması için C Programlama Dili Performans Analizi. 52

Tablo 4.46: Birleştirmeli Sıralama Algoritması için C# Programlama Dili Performans Analizi. 52

Tablo 4.47: Birleştirmeli Sıralama Algoritması için Java Programlama Dili Performans Analizi. 53

Tablo 4.48: Birleştirmeli Sıralama Algoritması için Python Programlama Dili Performans Analizi. 53

Tablo 4.49: Gnome Sıralama Algoritması için C Programlama Dili Performans Analizi. 54

Tablo 4.50: Gnome Sıralama Algoritması için C# Programlama Dili Performans Analizi. 54

Tablo 4.51: Gnome Sıralama Algoritması için Java Programlama Dili Performans Analizi. 54

Tablo 4.52: Gnome Sıralama Algoritması için Python Programlama Dili Performans Analizi. 55

Tablo 4.53: Yığın Sıralama Algoritması için C Programlama Dili Performans Analizi. 55

Tablo 4.54: Yığın Sıralama Algoritması için C# Programlama Dili Performans Analizi. 55

Tablo 4.55: Yığın Sıralama Algoritması için Java Programlama Dili Performans Analizi. 56

Tablo 4.56: Yığın Sıralama Algoritması için Python Programlama Dili Performans Analizi. 56

Tablo 4.57: Gezgin Satıcı Problemi Çözüm Algoritması için C Programlama Dili Performans Analizi. 57

Tablo 4.58: Gezgin Satıcı Problemi Çözüm Algoritması için C# Programlama Dili Performans Analizi. 57

Tablo 4.59: Gezgin Satıcı Problemi Çözüm Algoritması için Java Programlama Dili Performans Analizi. 58

Tablo 4.60: Gezgin Satıcı Problemi Çözüm Algoritması için Python Programlama Dili Performans Analizi. 58

Tablo 4.61: Sırt Çantası Problemi Çözüm Algoritması için C Programlama Dili Performans Analizi. 59

Tablo 4.62: Sırt Çantası Problemi Çözüm Algoritması için C# Programlama Dili Performans Analizi. 59

Tablo 4.63: Sırt Çantası Problemi Çözüm Algoritması için Java Programlama Dili Performans Analizi. 59

Tablo 4.64: Sırt Çantası Problemi Çözüm Algoritması için Python Programlama Dili Performans Analizi. 60

Tablo 4.65: Hanoi Kuleleri Problemi Çözüm Algoritması için C Programlama Dili Performans Analizi. 60

Tablo 4.66: Hanoi Kuleleri Problemi Çözüm Algoritması için C# Programlama Dili Performans Analizi. 61

Tablo 4.67: Hanoi Kuleleri Problemi Çözüm Algoritması için Java Programlama Dili Performans Analizi. 61

Tablo 4.68: Hanoi Kuleleri Problemi Çözüm Algoritması için Python Programlama Dili Performans Analizi. 61

SİMGE VE KISALTMA LİSTESİ

Kısaltmalar	Açıklama
NP	Nondeterministic-Polynomial – Polinomsal Olmayan
P	Polynomial – Polinomsal
ADL	Architecture Description Language – Mimari Tanımlama Dili
API	Application Programming Interface – Uygulama Programlama Arayüzü
JVM	Java Virtual Machine – Java Sanal Makinesi
CPU	Central Processing Unit – Merkezi İşlem Birimi

ÖZET

YÜKSEK LİSANS TEZİ

ÇEŞİTLİ MATEMATİKSEL PROBLEMLERİN ÇÖZÜMÜNDE PROGRAMLAMA DİLLERİNİN VE PLATFORMLARININ KARŞILAŞTIRILMASI

Uğur Osman YÜCEL

İstanbul Üniversitesi-Cerrahpaşa

Lisansüstü Eğitim Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

Danışman : Doç. Dr. Rüya ŞAMLI

Bir yazılım geliştiricisi için, doğru programlama dili ve platformu seçimi büyük bir önem arz etmektedir. Doğru programlama dili seçimi, programın performansına ve yazılım geliştiricisinin eforuna ve performansına doğrudan etki eder. Bir proje için programlama dili seçilirken, projenin kısıtları, yazılım geliştiricisinin öncelikleri gibi bir çok kritere dikkat edilmelidir. Bu tez kapsamında, farklı programlama dilleri ile literatürde en çok kullanılan genel matematiksel algoritmalar, arama ve sıralama algoritmaları ve polinomsal olmayan problemlerin çözüm algoritmaları geliştirilmiş, performans analizleri gerçekleştirilmiştir.

Haziran 2020, 93 sayfa.

Anahtar kelimeler: Programlama Dilleri, Performans Analizi, Yazılım Geliştirme, Yazılım Ölçümü

SUMMARY

M.Sc. THESIS

COMPARISION OF PROGRAMMING LANGUAGES AND PLATFORMS IN SOLUTION OF VARIOUS MATHEMATICAL PROBLEMS

Uğur Osman YÜCEL

Istanbul University-Cerrahpasa

Institute of Graduate Studies

Department of Computer Engineering

Supervisor : Assoc. Prof. Dr. Rüya ŞAMLI

It is one of the most important points to select the best programming language for a project for the software developer. Programming language selection directly affects the program's performance and also developers's effort and performance.It must be considered that project constraints, software developer's priorities while selecting the programming language for a project. Within the scope of this thesis, general mathematical algorithms, searching and sorting algorithms and solutions algorithms for NP problems have been developed and performance analyzes have been performed wtih different programmin languages.

June 2020, 93. pages.

Keywords: Programming Languages, Performance Analysis, Software Development, Software Measurement

1. GİRİŞ

Bilgisayar biliminin en önemli noktalarından biri, yazılımların başarılı bir şekilde tasarlanmasıdır. Bir yazılımın başarılı bir şekilde tasarlanabilmesi için kullanılacak programlama dilinin ve platformun doğru şekilde seçilmesi büyük önem arz etmektedir. Programlama dünyasında her programlama dili ve platformun birbirine üstün özellikleri bulunabilir. Burada önemli olan yazılım geliştiricinin hangi kriterlere göre seçim yaptığıdır. Bu tez çalışmasında; bu soruya bir cevap olabilmesi açısından C, C#, Java ve Python Programlama dilleri ve Visual Studio, Code::Blocks, Net Beans ve Jet Brains PyCharm platformları kullanılarak, Windows işletim sistemi üzerinde, genel matematiksel algoritmalar, arama ve sıralama algoritmaları ve polinomsal olmayan bazı problemlerin çözümleri gerçeklenmiş ve bu işlemler esnasında, programlama dilleri ve platformlar; işlemci (Merkezî İşlem Birimi – Central Processing Unit – CPU) kullanım yüzdeleri, bellek kullanımları, çöp toplama sayıları, çalışma süreleri açısından karşılaştırılmıştır.Bu programlama dillerinin seçilmesinin amacı yapısal, nesneye yönelik programlama, ve betik dillerinin örneklendirilmesidir.

Bu tez çalışmasının kalanı şu şekilde organize edilmiştir: Bölüm 2'de Genel Kısımlar başlığı altında, tezde ele alınan bahsedilen bütün problemler ve çözüm algoritmalarının genel anlatımı verilmiş ve konu ile ilgili bir literatür taraması sunulmuştur; Bölüm 3'te Malzeme ve Yöntem başlığı altında, programlama dilleri, programlama platformları ve bu tezde kullanılacak olan algoritma tasarlama yaklaşımları, Bölüm 4'te ise gerçekleştirilen ölçümlerden elde edilen bulgular ve bu bulguların yorumlandığı grafikler açıklanmıştır. Tez çalışmasında gerçekleştirilen analizlerin değerlendirilmesi ve gelecekte gerçekleştirilebilecek çalışmalar Bölüm 5'te sunulmuştur.

2. GENEL KISIMLAR

Bu bölümde, tez çalışması esnasında gerçekleştirilen analizlerin kolay bir biçimde anlaşılabilmesi için gerekli olan kavramlar genel özellikleriyle açıklanmıştır. Bu kavramlar; genel matematiksel algoritmalar, arama algoritmaları, sıralama algoritmaları ve polinomsal olmayan algoritmalardır. Bunlar dışında, bu bölümde tez çalışmasında ele alınmış olan çalışmalara benzerlik gösteren ve literatürde daha önce yayınlanmış çalışmaların da bir özeti verilmiştir.

2.1. GENEL MATEMATİKSEL ALGORİTMALAR

Bu kısımda, pek çok problemin çözümünde kullanılmakta olan genel matematiksel algoritmalar ele alınmıştır. Bu tez kapsamında, genel matematiksel algoritma olarak; Faktöriyel Hesaplama Algoritması, Üs Hesaplama Algoritması ve Asal Sayı Bulma Algoritması incelenmiş ve bu algoritmaların farklı programlama dillerindeki performans analizleri yapılmıştır. Faktöriyel hesaplama ve üs hesaplama işlemleri için geliştirilen algoritmalar iteratif ve rekürsif (özyinelemeli) yaklaşımların her ikisine uygun olarak da tasarlanmış ve bu tasarımlar kendi içerisinde de karşılaştırılmıştır. Bu sayede algoritmaların programlama dilleri ve platform kriterleri dışında tasarımları ile ilgili de karşılaştırma ve analizler gerçekleştirilmiştir.

2.1.1. Faktöriyel Hesaplama Algoritması

Faktöriyel, 0 ve 1 için 1; 1'den büyük tam sayılar için ise n söz konusu sayı olmak üzere $1x2x3x \dots xn$ şeklinde hesaplanan bir sayısal değerdir. Birçok matematiksel işlemde kendisine yer bulan bu faktöriyel sayısının hesaplanmasını ifade eden Faktöriyel Hesaplama Algoritması bu tez çalışmasında, hem iteratif hem de rekürsif olarak tasarlanarak kodlanmıştır.

İteratif olarak tasarlanan Faktöriyel Hesaplama Algoritması adım adım aşağıdaki gibi ifade edilebilir.

1. Fonksiyon sonunda geri döndürülmek üzere bir sayı tanımlanır ve 1'e eşitlenir
2. Sayı her defasında faktöriyeli alınacak sayıyla çarpılır.
3. Faktöriyeli alınacak sayı 1 eksiltilir.
4. Faktöriyeli alınacak sayı 1'e eşit olana dek adım 2'ye geri dönülür.

Bu algoritmada geri döndürmek üzere tanımlanan sayı değişkeninin 1'e eşitlenme sebebi çarpma işleminin etkisiz elemanı olmasıdır. Bu algoritmanın zaman karmaşıklığı $O(n)$'dir. Rekürsif olarak tasarlanan Faktöriyel Hesaplama Algoritması ise adım adım aşağıdaki gibi ifade edilebilir:

1. Fonksiyon, faktöriyeli alınacak sayı ile çarpılarak ve daha sonra sayı 1 eksiltilerek kendini geri döndürür.
2. Sayı 1'e eşit olana kadar bu adım tekrar edilir.

Bu algoritmanın zaman karmaşıklığı da iteratif algoritma gibi $O(n)$'dir. Algoritmaların karmaşıklığının aynı olması, diğer karşılaştırma kriterlerini etkilememesi açısından avantajlı bir durumdur. Tablo 2.1 ve Tablo 2.2'de iki farklı algoritma tasarımı için 4 sayısının faktöriyelinin hesaplanması adım adım gösterilmiştir.

Tablo 2.1: 4 Sayısı için İteratif Faktöriyel Hesaplama Algoritması ile Çözüm Örneği

Sayaç	Sayı	Sonuç
0	4	4
1	3	12
2	2	24
3 (Döngü Sonu)	1	24

Tablo 2.2: 4 Sayısı için İteratif Faktöriyel Hesaplama Algoritması ile Çözüm Örneği

Sayı	Fonksiyon Değeri	Sonuç
4	1	4
3	4	12
2	12	24
1 (Durma Koşulu)	24	24

2.1.2. Üs Alma Algoritması

Üs alma işlemi, matematikte, a^b şeklinde ifade edilen işlemin sonucunun hesaplanmasıdır. Bu tez çalışmasında üs alma işlemi gerçekleştirilirken a'nın 0'a eşit olması, b'nin 0'a eşit olması, b'nin negatif bir değer olması, b'nin rasyonel bir sayı olması gibi özel durumlar elimine edilmiştir. Üs Alma Algoritması da Faktöriyel Hesaplama Algoritması'na benzer şekilde hem iteratif hem de rekürsif olarak tasarlanarak kodlanmıştır. İteratif olarak tasarlanan Üs Alma Algoritması adım adım aşağıdaki gibi ifade edilebilir.

1. Fonksiyonun sonunda geri döndürmek üzere bir sayı tanımlanır ve 1'e eşitlenir.
2. Tanımlanan sayı, üs sayısı kadar taban sayısı ile çarpılır.

Geri döndürmek üzere tanımlanan sayı değişkeninin 1'e eşitlenme sebebi çarpma işleminin etkisiz elemanı olmasıdır. Bu algoritma $O(n)$ zaman karmaşıklığına sahiptir. Rekürsif olarak tasarlanan Üs Alma Algoritması adım adım aşağıdaki gibi ifade edilebilir.

1. Fonksiyon, taban sayısı ile çarpılarak, kendisinin üs sayısı 1 eksiltilmiş hali ile geri döndürülür
2. Bu işlem üs sayısı 0'a eşit olana dek devam eder.

Bu algoritmanın zaman karmaşıklığı da iteratif algoritma gibi $O(n)$'dir. Algoritmaların karmaşıklığının aynı olması, diğer karşılaştırma kriterlerini etkilememesi açısından avantajlı bir durumdur.

Tablo 2.3 ve Tablo 2.4'te 2^3 Sayısı için üs alma algoritması adım adım gösterilecektir.

Tablo 2.3: 2^3 Sayısı için İteratif Üs Alma Algoritması ile Çözüm Örneği

Üs	Sonuç
3	1
2	2
1	4
0 (Döngü Sonu)	8

Tablo 2.4: 2^3 Sayısı için Rekürsif Üs Alma Algoritması ile Çözüm Örneği

Üs	Fonksiyon Değeri	Sonuç
3	1	2
2	2	4
1	4	8
0 (Durma Koşulu)	8	8

2.1.3. Asal Sayı Bulma Algoritması

Asal sayılar, matematikte sadece 1'e ve kendisine bölünebilen sayılar olarak tanımlanmıştır ve şifreleme başta olmak üzere bilgisayar biliminde pek çok alanda kullanılmaktadır. Dolayısıyla bir sayının asal olup olmadığının tespit edilmesi, bilgisayar bilimi açısından büyük önem arz

etmektedir. Bu tezde Asal Sayı Bulma Algoritması uygulanırken fonksiyona parametre olarak üst sınır değeri verilmiştir. 2'den başlayarak verilen üst sınıra kadar olan tüm asal sayılar ekrana yazdırılmıştır. Bu algoritma adım adım aşağıdaki gibi ifade edilebilir.

1. 2'den başlayarak sayıya kadar giden bir döngü oluşturulur.
2. Döngünün içinde if bloğuyla sayının asal olup olmadığı kontrol edilir.
3. Eğer sayı asal ise ekrana yazdırılır.
4. Döngünün sonunda sayı 1 arttırılır.
5. Üst sınıra gelinene kadar adım 2'ye geri dönülür.

Bu algoritmanın zaman karmaşıklığı $O(n)$'dir. Bunun sebebi ise bir sayının asal olup olmadığı kontrol edilirken, yarısından küçük tüm sayılara bölünüp bölünmediğinin kontrol edilmesidir. Diğer bir deyişle döngü $n/2$ kez tekrar etmektedir. Sabit sayılar zaman karmaşıklığına etki etmediği için bu algoritmanın zaman karmaşıklığı $O(n)$'dir.

Tablo 2.5 ve Tablo 2.6'da 6 sayısının ve 7 sayısının asal sayı olup olmadıkları durumlarının adım adım kontrolü ve Tablo 2.7'de 2 ile 10 arasındaki sayılar için asal sayı bulma algoritması ile çözümü adım adım gösterilmiştir.

Tablo 2.5: 6 Sayısının Asal Olup Olmaması Durumunun Adım Adım Kontrolü

Adım	İşlem
1	Sayı = 6
2	6 % 2 = 0
3	6 % 3 = 0
4	Sayının 2 ve 3 ile bölümünden kalanlarının 0'a eşit olması nedeniyle 6 sayısı ASAL DEĞİLDİR

Tablo 2.6: 7 Sayısının Asal Olup Olmaması Durumunun Adım Adım Kontrolü

Adım	İşlem
1	Sayı = 7
2	7 % 2 = 1
3	7 % 3 = 1
4	Sayının 2 ve 3 ile bölümünden kalanlarının 0'a eşit olmaması nedeniyle 7 sayısı ASAL SAYIDIR

Tablo 2.5. incelendiğinde, 6 sayısı için 2 ve 3 sayılarına bölünme durumu kontrol edilmiştir. Bir sayının asal olup olmadığını anlayabilmek için sayının 2'den başlayarak yarısına kadar tam olarak bölünme durumları kontrol edilir. Bu durum 2 ve 3 sayıları hariç tüm asal sayılar için geçerlidir. 6 sayısı 2 ve 3 sayılarına tam bölündüğü için 6 sayısının asal sayı olmadığı sonucuna ulaşılmıştır. Tablo 2.6.' de ise 7 sayısı 2 ve 3 sayılarının her ikisine de tam bölünmediği için 7 sayısının asal sayı olduğu sonucuna ulaşılmaktadır. Tablo 2.7'de ise bu tezde farklı programlama dilleri kullanılarak kodlanan Asal Sayı Bulma Algoritması gösterilmiştir.

Tablo 2.7: 2 ile 10 Arasındaki Sayılar için Asal Sayı Bulma Algoritması ile Çözüm Örneği

Sayı	Asal Olma Durumu
2	Sayı asaldır.
3	Sayı asaldır.
4	Sayı asal değildir.
5	Sayı asaldır.
6	Sayı asal değildir.
7	Sayı asaldır.
8	Sayı asal değildir.
9	Sayı asal değildir.

2.2. ARAMA ALGORİTMALARI

Herhangi bir veri kümesi içerisinde belli anahtar verilerin arandığı algoritmalar olarak ifade edilebilecek olan arama algoritmaları, bilgisayar biliminin en temel problemlerinden biri olarak kabul edilmektedir. Özellikle büyük miktarda verinin bulunduğu alanlarda doğru bilgiye kısa sürede ulaşılmasını sağlar. Günümüzde büyük veri (big data) konusunun her geçen gün önem kazandığı düşünülecek olursa, arama algoritmalarının da paralel olarak öneminin arttığı söylenebilir. Bu algoritmalar ile metinler, diziler, listeler, şekiller ve farklı tip veriler üzerinde arama yapılabilmektedir. Her arama algoritması için en iyi durum, en kötü durum ve ortalama durum şu şekilde ifade edilebilir:

- En iyi durum: Aranan elemanın veri kümesinde kontrol edilen ilk eleman olması durumudur.
- En kötü durum: Aranan elemanın veri kümesinde kontrol edilen son eleman olması veya veri kümesinde bulunmaması durumudur.
- Ortalama durum: Bu iki durum haricinde kalan tüm durumlardır.

Bu tez kapsamında, literatürde en çok kullanılan arama algoritmaları arasında yer alan Doğrusal Arama Algoritması ve İkili Arama Algoritması ele alınmıştır. Bu algoritmalar aşağıda kısaca açıklanmıştır.

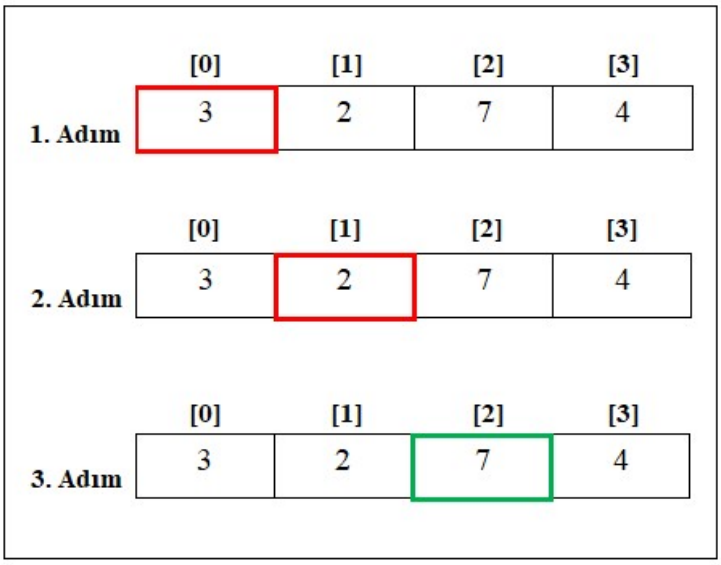
2.2.1. Doğrusal Arama Algoritması

Doğrusal arama algoritması adım adım aşağıdaki gibi ifade edilebilir:

1. Aranmak istenen elemanı, listenin ilk elemanı ile karşılaştırılır.
2. Eğer eşitlerse, bulunan indeks ekrana yazılır.
3. Eşit değillerse, sıradaki elemana geçilir.
4. Listenin sonuna gelinene kadar adım 2'ye geri dönülür.

Bu işlem $O(n)$ zaman karmaşıklığına sahiptir. Bunun sebebi ise fonksiyonun içerisindeki döngünün en fazla n kere tekrarlanabilmesidir.

Şekil 2.1'de örnek bir tamsayı dizisi ile Doğrusal Arama Algoritması adım adım gösterilmiştir. Şekilde 7 elemanı için Doğrusal Arama Algoritması adım adım gösterilmiştir. İlk adımda dizinin ilk indeksindeki sayının ile aranılan 7 sayısı ile eşit olup olmadığı kontrol edilmiştir. Sayılar eşit olmadığı için sonraki indekse geçilmiştir. 3. adımda aranılan sayı ile kontrol edilen sayı eşit olduğu için arama durdurulmuş ve fonksiyon 7 sayısının bulunduğu indeksi kullanıcıya döndürmüştür.



Şekil 2.1: 7 sayısı için Doğrusal Arama Algoritması çalışması

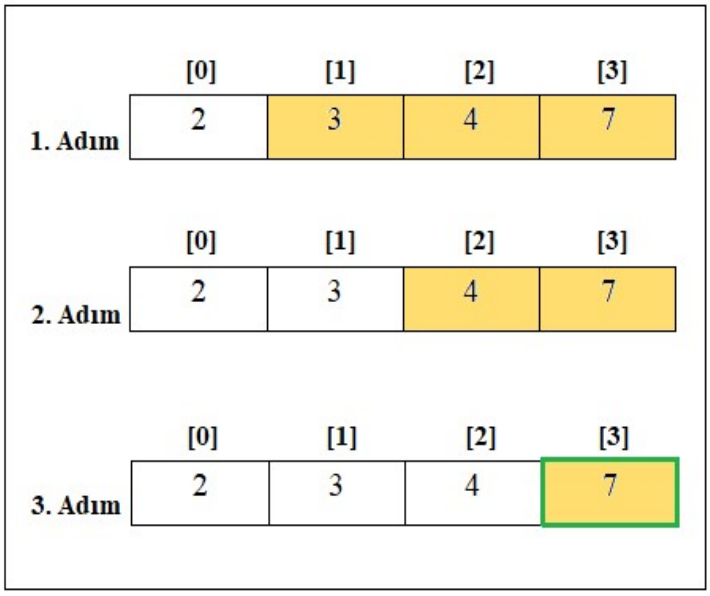
2.2.2. İkili Arama Algoritması

İkili Arama Algoritması Böl ve Fethet stratejisini izleyen bir arama algoritmasıdır. Sıralı diziler üzerinde uygulanan bu algoritma, sıralı diziyi, sıralı alt dizilere bölerek elemanı aramaktadır. Bu algoritma adım adım aşağıdaki gibi ifade edilebilir.

1. Dizinin ortanca elemanı bulunur.
2. Eğer aranan eleman ortanca elemandan büyükse, dizinin ortanca indeksi ile son indeksi arasına bakılır.
3. Eğer aranan eleman ortanca elemandan küçükse, dizinin ilk indeksi ile ortanca indeksi arasına bakılır.
4. Eğer aranan eleman ortanca elemana eşitse, ekrana bulunduğu indeks yazdırılır.
5. Dizideki her eleman kontrol edilene dek, oluşan her yeni alt dizi için 1. adıma dönülür.

Bu algoritmanın zaman karmaşıklığı $O(logn)$'dir.

Şekil 2.2'de örnek bir tamsayı dizisi ile Doğrusal Arama Algoritması adım adım gösterilmiştir.



Şekil 2.2: 7 Sayısı için İkili Arama Algoritması Çalışması

Yukarıdaki şekilde 7 sayısı için İkili Arama Algoritması'nın çalışması gösterilmiştir. 1.adımda dizinin ortasındaki eleman bulunmuştur. Aranan eleman 3'e eşit olmadığı ve 3'ten büyük olduğu için sağına bakılmıştır. 2.Adımda geriye kalan sıralı alt dizinin ortasındaki eleman 4'tür. Bir sonraki adımda ise 4'ün sağına gidilerek, aranan elemana ulaşılmıştır.

Şekil 2.2'de her adımda sarı arka plan dolgusu ile gösterilen elemanlar taranmaktadır. 3. adımda etrafı yeşil çerçeve ile kaplanan eleman ise aranılan elemandır.

2.3. SIRALAMA ALGORİTMALARI

Sıralama algoritmalarının kullanım amacı dağınık halde bulunan veri setinin düzenli hale getirilmesidir. Sıralama işlemi belirli kriterlere göre gerçekleştirilir. Bunlardan en önemlisi küçükten büyüğe, büyükten küçüğe ve alfabetik sıralamadır. Sıralama algoritmaları da arama algoritmaları gibi bilgisayar biliminin önemli konularından birini oluşturmaktadır. Her sıralama algoritması için en iyi durum, en kötü durum ve ortalama durum şu şekilde ifade edilebilir:

- En iyi durum: Sıralanması istenen veri kümesinin istenilen şekilde sıralı olması durumudur.
- En kötü durum: Sıralanması istenen veri kümesinin istenilenin tam tersi şeklinde sıralı olması durumudur.
- Ortalama Durum: Bu iki durum haricinde kalan tüm durumlardır.

Bu tez kapsamında, literatürde en çok kullanılan sıralama algoritmaları arasında yer alan, Eklemeli Sıralama Algoritması, Seçmeli Sıralama Algoritması, Kabarcık Sıralama Algoritması, Çabuk Sıralama Algoritması, Birleştirme Sıralama Algoritması, Gnome Sıralama Algoritması ve Yığın Sıralama Algoritması ele alınmıştır.

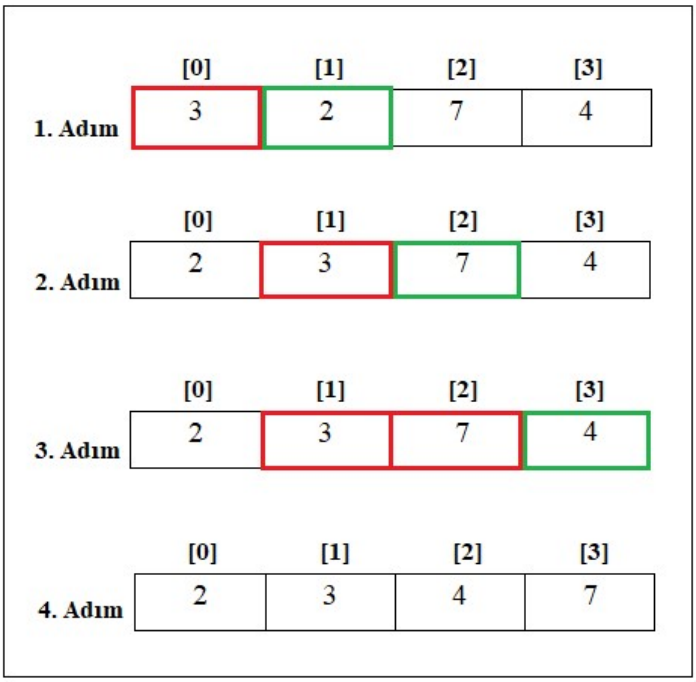
2.3.1. Eklemeli Sıralama Algoritması

En basit ifadeyle dizinin elemanlarının seçilip, sıralı alt diziye eklendiği algoritmadır. Algoritma adım adım aşağıdaki gibi ifade edilebilir:

1. Dizinin ikinci elemanı seçilir ve birinci elemanla karşılaştırılır.
2. Eğer ikinci eleman küçükse, yer değiştirilir. (Bu işlem sonucunda sıralı alt dizi oluşturulmuş olur.)
3. Sıradaki eleman seçilir ve sıralı alt dizide olması gereken yere eklenir.
4. Son eleman da alt diziye aktarıldığında, dizi sıralı hale getirilmiş olur.

Bu algoritma, en kötü durumunda $O(n^2)$, en iyi durumunda ise $O(n)$ zaman karmaşıklığına sahiptir.

Şekil 2.3'te örnek bir tamsayı dizisi ile Eklemeli Sıralama Algoritması adım adım gösterilmiştir. Şekilde her adımda yeşil çerçeve ile çevrilmiş elemanlar ile kırmızı çerçeve ile çevrilen elemanlar karşılaştırılmaktadır. İlk adımda dizinin 1.indeksinde yer alan eleman ile 0.indeksinde yer alan eleman karşılaştırılmış ve yer değiştirme işlemi gerçekleştirilmiştir. Sıradaki adımda ise kırmızı ve yeşil çerçeve ile çevrili elemanlar karşılaştırılmış ve yer değiştirme koşulu sağlanmadığı için sonraki adıma geçilmiştir. Sonraki adımda ise yeniden yer değiştirme işlemi yapılıp dizi sıralı hale getirilmiştir.



Şekil 2.3: Örnek bir tamsayı dizisi için Eklemeli Sıralama Algoritması Çalışması

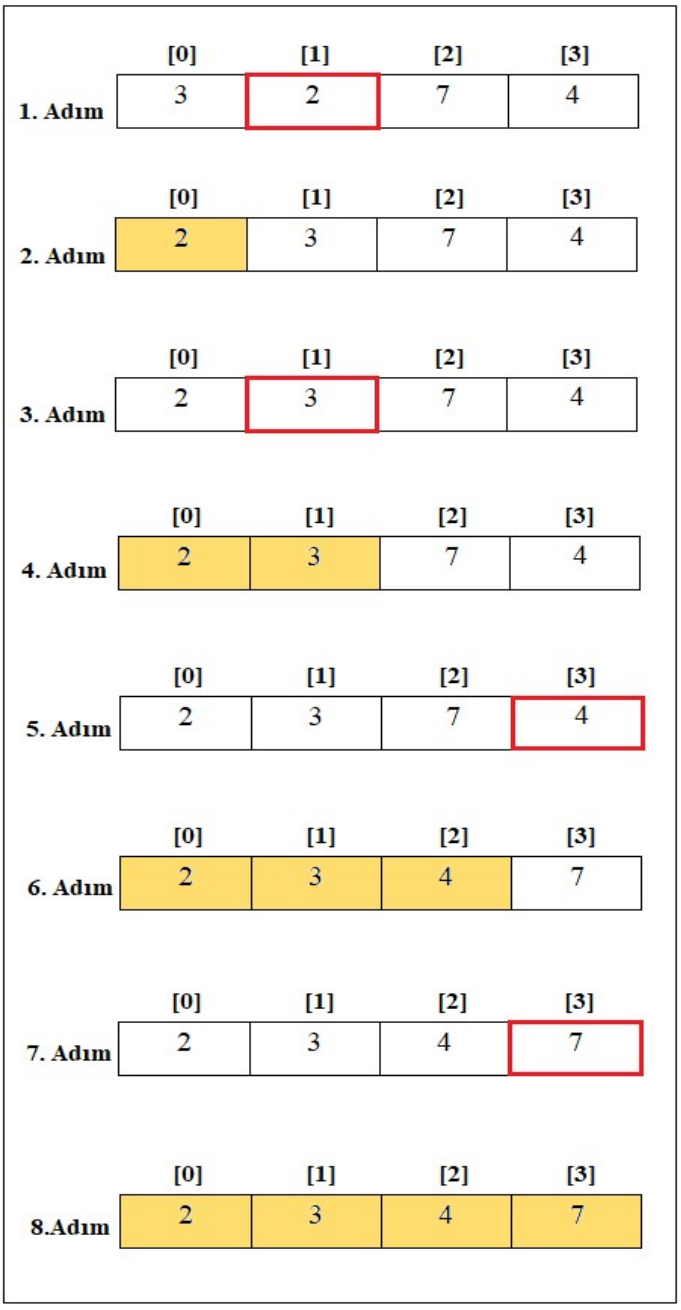
2.3.2. Seçmeli Sıralama Algoritması

Bu algoritmada dizinin en küçük elemanı seçilip ilk indekse yerleştirilir. Her döngünün sonunda indeks bir arttırılarak tüm dizinin sıralı hale gelmesi sağlanır. Algoritma adım adım aşağıdaki gibi ifade edilebilir:

1. İndeks numarası 0 olarak atanır.
2. Dizinin en küçük elemanı seçilir.
3. Dizinin indeks numarasındaki elemanıyla yer değiştirilir.
4. İndeks numarası bir arttırılır.
5. Dizinin son elemanına gelene kadar 2. adıma geri dönülür.

Bu algoritma, en kötü durumunda da en iyi durumunda da $O(n^2)$ zaman karmaşıklığına sahiptir. Bunun sebebi, sıralanmaya çalışılan dizinin tamamen sıralı olması halinde bile, indeks numarasının eleman sayısına uygun olacak şekilde n kez değişecek olması ve her indeks için dizinin en küçük elemanının seçilip yerinin değiştirilmesidir. Diğer bir deyişle, dizinin sıralı olması ve olmaması koşullarının ikisinde de n^2 sayıda işlem yapılacaktır.

Şekil 2.4'te örnek bir tamsayı dizisi ile Seçmeli Sıralama Algoritması adım adım gösterilmiştir.



Şekil 2.4: Örnek bir tamsayı dizisi için Seçmeli Sıralama Algoritması Çalışması

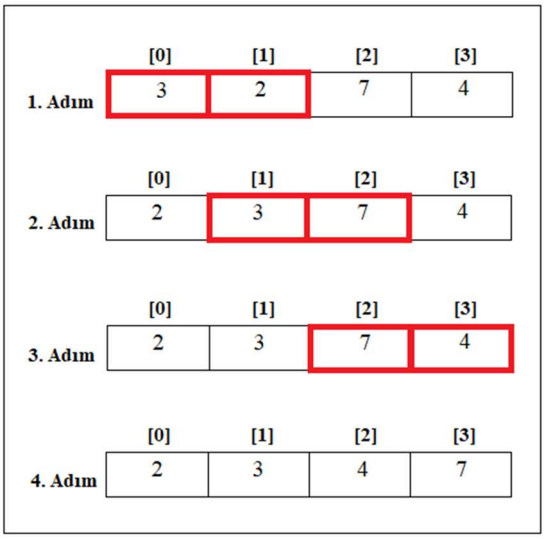
Şekilde ilk adımda dizinin en küçük elemanı seçilip ilk indeksteki sayı ile değiştirilmiştir. Sonraki adımlarda sarı arka plan dolgusu ile gösterilen elemanlar hedeflenen sıralı alt diziyi göstermektedir. Kırmızı çerçeveli elemanlar ise sıralı alt dizi haricindeki en küçük elemanlardır.

2.3.3. Kabarcık Sıralama Algoritması

Dizilerin sıralanmasında kullanılan en basit algoritmalardan biridir. Dizideki ardışık elemanların birbirleriyle karşılaştırılması üzerine kurulmuş bir sıralama algoritmasıdır. Algoritma adım adım aşağıdaki gibi ifade edilebilir:

1. Dizinin ilk iki elemanını alınır.
2. Alınan iki elemanı birbiriyle karşılaştırılır.
3. Eğer ilk eleman büyükse yer değiştirilir.
4. Değilse, sıradaki elemana geçilir.
5. Dizinin sonuna kadar kontrol edilir.
6. Bütün dizi elemanlarını karşılaştırana kadar ikinci adıma geri dönülür.

Bu algoritmada, en kötü durumda $(n\ x\ n-1)$ işlem yapılacağı için $O(n^2)$ zaman karmaşıklığı vardır. En iyi durumda ise $O(n)$ zaman karmaşıklığı vardır. Şekil 2.5'te Kabarcık Sıralama Algoritmasının 4 elemanlı bir tamsayı dizisi için adım adım uygulanma şekli gösterilmiştir. Şekilde Kabarcık Sıralama Algoritmasının adımları görülmektedir. Her adımda gösterilen kırmızı çerçeveli elemanlar birbirleriyle karşılaştırılan elemanlardır.



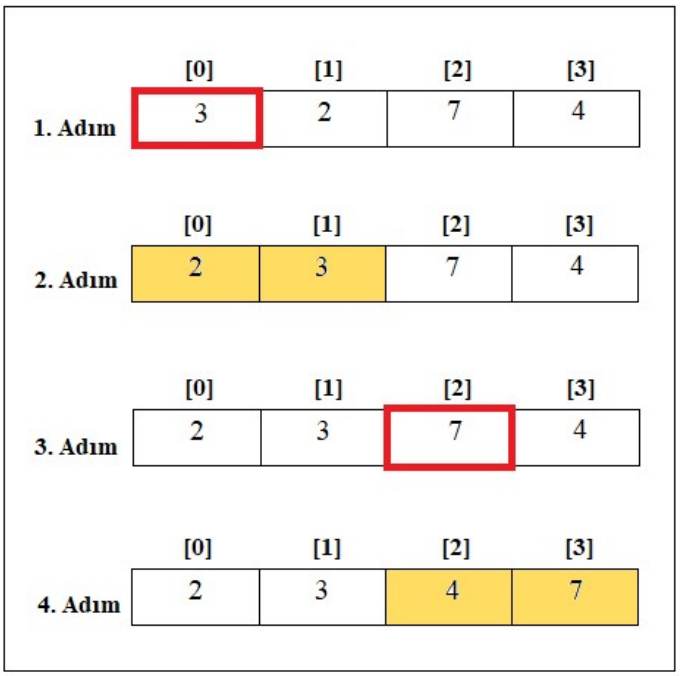
Şekil 2.5: Örnek bir tamsayı dizisi için Kabarcık Sıralama Algoritması Çalışması

2.3.4. Çabuk Sıralama Algoritması

Bu algoritma, Böl ve Fethet stratejisini izleyen rekürsif bir algoritmadır. Algoritma adım adım aşağıdaki gibi ifade edilebilir:

1. Dizi elemanlarından bir pivot seçilir.
2. Pivottan küçük elemanlar sola, büyük elemanlar sağa yerleştirilir.
3. Soldaki ve sağdaki alt dizilerden birer adet pivot seçilir.
4. Elde edilen yeni diziler tek elemanlı kalana kadar 2. adıma geri dönülür.
5. Tek başına bırakılan elemanlar birleştirilerek sıralı dizi oluşturulur.

Çabuk Sıralama Algoritması en iyi durumunda $O(nlog(n))$ zaman karmaşıklığına sahiptir. Bunun sebebi ise her bölme işlemi için $log(n)$ kez işlem yapılmasıdır. En kötü durumda ise $O(n^2)$ zaman karmaşıklığı vardır. Aradaki farklılığın sebebi, pivot seçiminden kaynaklanmaktadır. Eğer, en küçük veya en büyük eleman pivot seçilirse, her bölme işleminde diziden bir eleman ayrılacaktır. n elemanlı bir diziyi bölmek için $n-1$ kere tekrar edilmelidir. $n-1$ x n işleminden $O(n^2)$ olur. Aşağıdaki şekilde Çabuk Sıralama Algoritmasının 4 elemanlı bir tamsayı dizisi için adım adım uygulanma şekli gösterilmiştir.



Şekil 2.6: Örnek bir tamsayı dizisi için Çabuk Sıralama Algoritması Çalışması

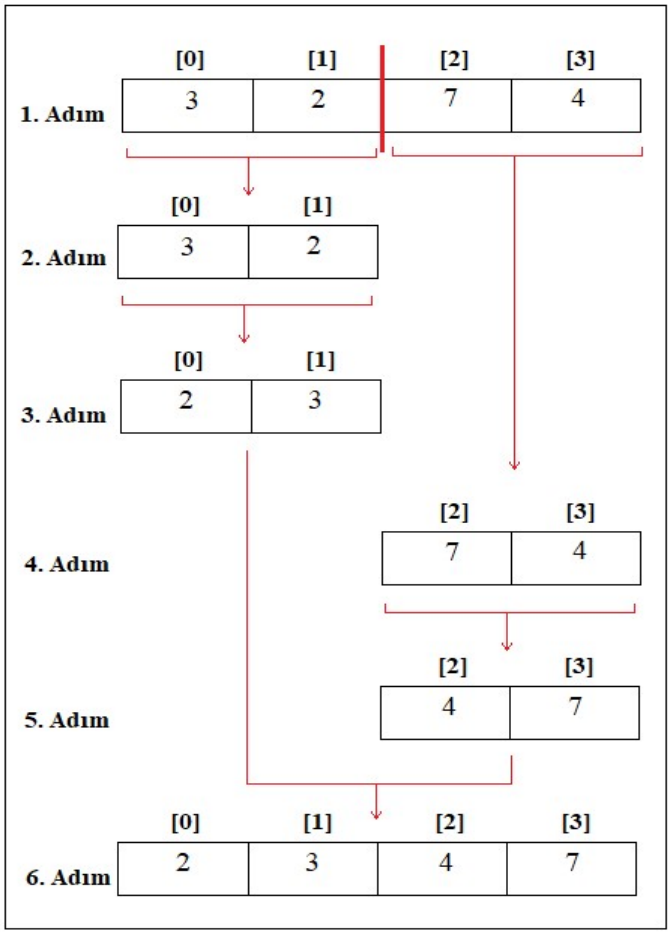
Yukarıdaki şekilde ilk adımda pivot olarak seçilen eleman dizinin ilk elemanı olan 3'tür. Pivot geriye kalan tüm elemanlarla karşılaştırılarak doğru yere yerleştirilir. 2.adımda yer değiştirme işlemi gerçekleştirilmiştir. Sonraki adımda ise eski pivotun sağında kalan alt dizinin yeni pivotu 7 elemanıdır. Sağ alt dizinin sıralı duruma gelmesi için pivotun sağındaki eleman ile yer değişmesi gerekmektedir. Bu işlem 4. adımda gerçekleşmiştir. Böylece dizi sıralı hale gelmiştir. 2. ve 4. adımlarda sarı arka plan dolgusu ile gösterilen elemanlar sıralı alt dizileri göstermektedir.

2.3.5. Birleştirme Sıralama Algoritması

Çabuk Sıralama Algoritması'nda olduğu gibi Birleştirme Sıralama Algoritması da Böl ve Fethet stratejisini izleyen rekürsif bir algoritmadır. Algoritma adım adım aşağıdaki gibi ifade edilebilir:

1. Dizi ikiye bölünür.
2. Oluşan alt diziler tek elemanlı hale gelene kadar 1. adım tekrarlanır.
3. Tek elemanlı diziler sıralanarak birleştirilmeye başlanır.
4. Tüm elemanlar eklenene kadar 3. adıma geri dönülür.

Bu algoritmanın zaman karmaşıklığı en iyi senaryoda da en kötü senaryoda da $O(nlogn)$' dir. Bunun sebebi ise, sıralı dizi bile işleme alınsa, dizinin parçalanması ve tekrar birleştirilmesidir. Şekil 2.7'de Birleştirme Sıralama Algoritmasının 4 elemanlı bir tamsayı dizisi için adım adım uygulanma şekli gösterilmiştir. Şekilde ilk adımda dizinin ilk hali görünmektedir. Dizinin sıralı hale gelebilmesi için önce ortadan iki alt diziye ayrılması gerekmektedir. Sonraki adımda sol alt dizi de kendi içinde ikiye ayrılmıştır ve sıralı hale gelebilmesi için bir sonraki adımda gösterilen yer değiştirme işlemi gerçekleşmiştir. Sonraki iki adımda ise aynı işlem sağ alt diziye uygulanmıştır. Dizi 5.adımda sıralı gözükse de sıralama işlemi henüz bitmemiştir. Alt diziler sıralı hale geldikten sonra birleştirilerek dizi sıralı hale getirilmiştir.



Şekil 2.7: Örnek bir tamsayı dizisi için Birleştirme Sıralama Algoritaması Çalışması

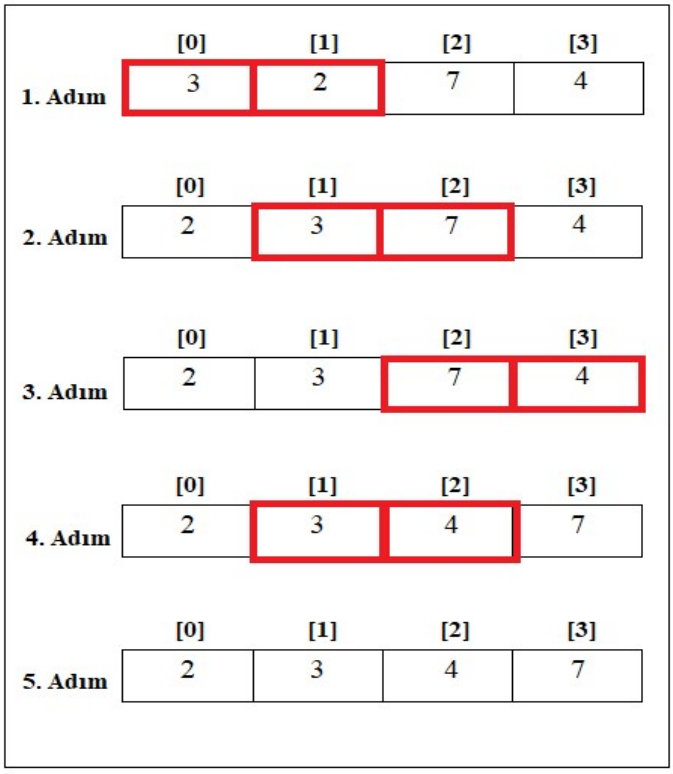
2.3.6. Gnome Sıralama Algoritması

Gnome Sıralama Algoritması, Eklemeli Sıralama Algoritması ve Kabarcık Sıralama Algoritması'na benzer bir yapıya sahiptir. Algoritma adım adım aşağıdaki gibi ifade edilebilir:

1. Dizinin ilk iki elemanı ele alınır.
2. Eğer soldaki eleman büyükse yer değiştirilir.
3. Yer değiştirilen elemanın doğru yeri bulunana kadar ters yöne doğru ilerlenir.
4. Dizinin sonuna gelene kadar bu 2. ve 3. adımlar uygulanır.

Bu algoritma $O(n^2)$ zaman karmaşıklığına sahiptir. Eğer liste sıralı ise en iyi durumundaki zaman karmaşıklığı $O(n)$ olur. Aşağıdaki şekilde (Şekil 2.8) Gnome Sıralama Algoritmasının 4 elemanlı bir tamsayı dizisi için adım adım uygulanmışı gösterilmiştir. Şekilde kırmızı ile gösterilen elemanlar birbirleriyle karşılaştırılan elemanlardır. Şeklin çok büyümemesi ve okunurluğun azalmaması açısından geriye dönük kontrol adımları gösterilmemiştir. Gnome

Sıralama Algoritması her zaman iki elemanın karşılaştırılması ve eğer gerekliyse yer değiştirmesi ile uygulanmaktadır. Her yer değiştirme işleminden sonra geriye dönük kontrol adımları gerçekleşmektedir.

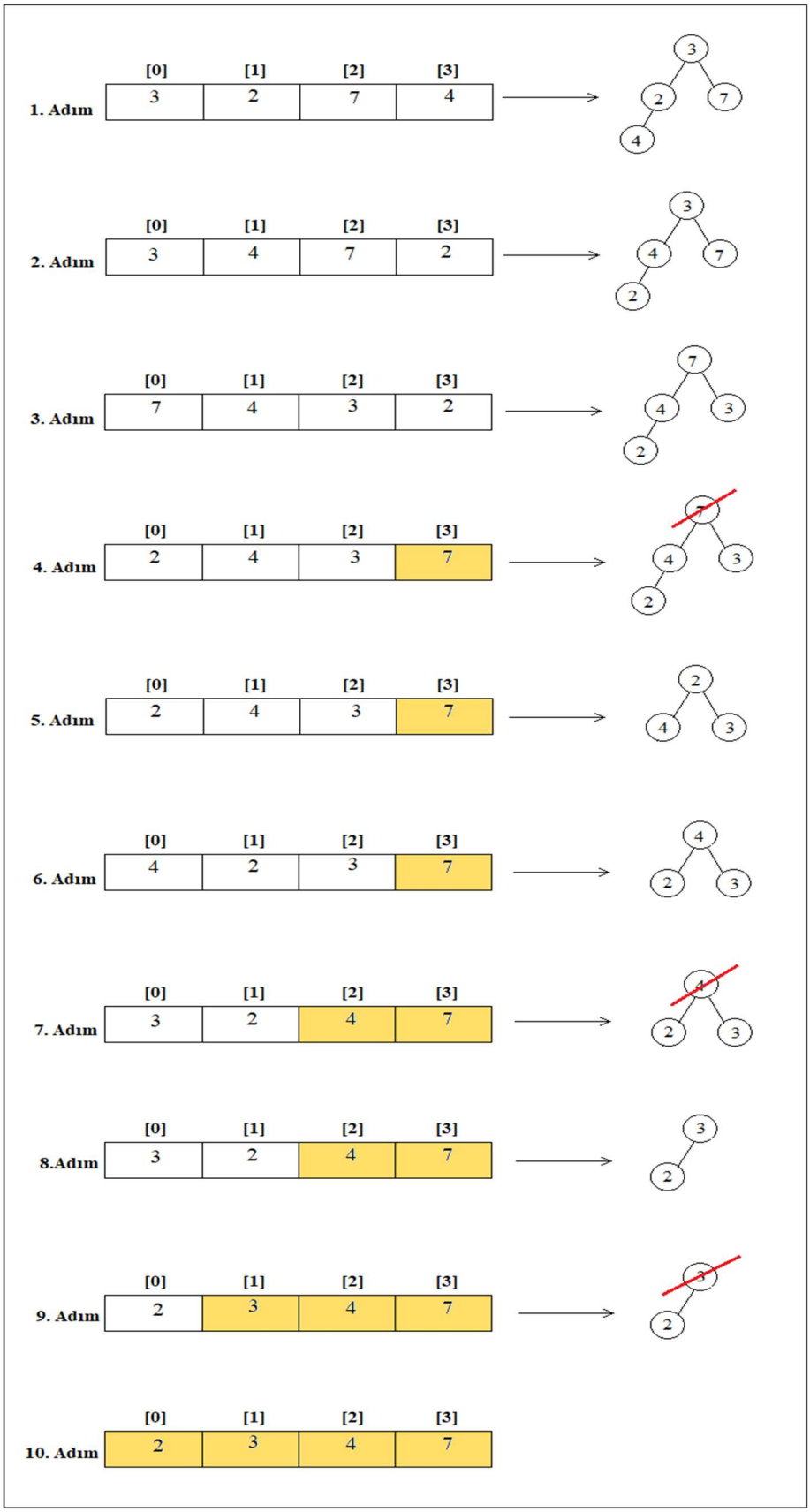


Şekil 2.8: Örnek bir tamsayı dizisi için Gnome Sıralama Algoritması Çalışması

2.3.7. Yığın Sıralama Algoritması

Yığın Sıralama Algoritması, diğer sıralama algoritmalarından farklı olarak bir ağaç yapısı üzerinde gerçekleştirilir. Yığın ağaçları olarak adlandırılan bu ağaçlara eleman eklerken veya bu ağaçlardan eleman çıkartılırken kök elemanı her zaman en büyük eleman olacak şekilde rotasyon yapılır. Dizinin elemanları yığın ağaçlarına aktarılır ve ağacın en üstündeki elemanlar diziye tekrar aktarılarak sıralama yapılır. Algoritma adım adım aşağıdaki gibi ifade edilebilir:

1. Yığın ağacı oluşturulur.
2. Dizinin elemanları yığın ağacına aktarılır.
3. Yığın ağacının kök elemanı diziye aktarılır.
4. Aktarma işleminden sonra rotasyonlar yapılarak, yığın ağacının en büyük elemanının tekrar kök elemanı olması sağlanır.
5. Yığın ağacı boşalana kadar adım 3'e geri dönülür.



Şekil 2.9: Örnek bir tamsayı dizisi için Yığın Sıralama Algoritması Çalışması

Yığın sıralama algoritması en iyi durumunda $O(n)$ zaman karmaşıklığına sahipken, en kötü durumunda ise $O(nlog(n))$ zaman karmaşıklığına sahiptir. Şekil 2.9'da Yığın Sıralama Algoritmasının 4 elemanlı bir tamsayı dizisi için adım adım uygulanışı gösterilmiştir.

Şekil 2.9. incelendiğinde ilk adımda dizinin elemanları yığın ağacına aktarılmıştır. Ardından yığın ağacında gerekli rotasyonlar yapılıp Maksimum Yığın Ağacına dönüştürülmüştür. 4.adımda ise ağacın kök elemanı olan 7'nin dizinin en büyük elemanı olduğu belirlenip, sıralı alt diziye aktarılmıştır. Sonraki işlemler yığın ağacında 1 eleman kalana dek devam etmiştir. Yığın ağacında son eleman kaldığında algoritma sona ermiştir.

2.4. POLİNOMSAL OLMAYAN ALGORİTMALAR

Literatürdeki problemler, genel olarak, polinomsal problemler ve polinomsal olmayan problemler olarak ikiye ayrılırlar. Bu ayrım problemlerin, deterministik olmayan Turing makinesi üzerinde polinomsal zamanda çözülebilmesine veya çözülememesine göre yapılmaktadır. P (Polynomial - Polinomsal) sınıfındaki problemlerin çözümü, polinom olarak ifade edilebilen bir zamanda gerçekleştirilebilirken, NP (Nondeterministic-Polynomial - Polinomsal Olmayan) sınıfındaki problemlerin çözümü polinom olarak ifade edilebilen bir zamanda gerçekleştirilebilmektedir. NP sınıfındaki problemlere örnek olarak, bu tezde uygulanan Gezgin Satıcı Problemi, Sırt Çantası Problemi, Hanoi Kuleleri Problemi, Düğüm Kapsama Problemi, Hamilton Yolu Problemi, Alt Küme Toplamı Problemi ve Çizelgeleme Problemi verilebilir.

NP sınıfındaki problemler, genellikle aritmetik yöntemler, sayısal analiz yöntemi ve optimizasyon algoritmaları ile çözülmektedir. Ancak, bu problemlerde, değişken sayısı arttıkça, sayısal analiz ve aritmetik yöntemlerin kullanımı zor olmakta ve çözümün elde edilmesi çok uzun sürmektedir [1].

NP problemleri çözümünde veri sayısı arttıkça verimlilik düşmektedir. Bu nedenle birçok araştırmacı tarafından çözüm algoritmaları incelenmektedir. NP sınıfına ait problemlerin çözümüyle ilgili yapılan bilimsel araştırmalar, optimizasyon algoritmalarının, sayısal analiz yöntemlerine ve aritmetik yöntemlere göre çok daha iyi sonuçlar verdiğini göstermektedir. Bu tezde NP problemler arasından Gezgin Satıcı Problemi, Sırt Çantası Problemi ve Hanoi Kuleleri Problemi üzerinde çalışılmıştır ve bu problemlerin çözülmesi için birer algoritma geliştirilerek

algoritmaların farklı programlama dilleri ile yapılan simülasyonlarındaki performansları karşılaştırılmıştır.

2.4.1. Gezgin Satıcı Problemi Çözüm Algoritması

Gezgin Satıcı Problemi literatürde en çok kullanılan NP sınıfına ait problemlerden biridir [2-5]. Gezgin Satıcı Problemi'nde düğümler, düğümler arası uzaklıklar ve satıcı -bulunmaktadır. Bu problemde, satıcının amacı her düğüme sadece bir kere uğrayarak [6], tüm düğümleri en kısa olacak şekilde dolaşmaktır. Sadece başlangıç düğümüne iki kere uğranır. Problemin karmaşıklığı düğüm sayısı ve düğümler arası yollarla doğru orantılıdır.

Literatürdeki çalışmalardan, Gezgin Satıcı Problemi'nin çözümünde, birçok farklı yöntemin kullanıldığı ve bu tezde kullanılan Geri İzleme (backtracking) Algoritması'ndan daha iyi sonuçlar veren çalışmalar olduğu görülmüştür [7-11]. Buna rağmen, tezde Geri İzleme Algoritması'nın kullanılmasının sebebi bu tip algoritmaların bilgisayara daha çok işlem yaptırması dolayısıyla tez içerisinde yapılacak analizlerin (CPU, bellek kullanımı ve çalışma süreleri ölçümü) daha net bir biçimde ortaya konulacağı düşüncesidir.

Bu tezde Gezgin Satıcı Problemi çözülürken, rekürsif bir tasarım gerçekleştirilmiştir. Çözüm esnasında, ilk olarak çizgelerin uzaklık matrisleri çıkartılmıştır. Ardından boolean (ikili) türünde diziler oluşturulup ilk elemanları hariç diğer elemanları "false" (yanlış) olarak atanmıştır. Eğer o noktaya uğrandıysa, bu türdeki dizi üzerinde, o noktayı gösteren indeks "true" (doğru) olarak değiştirilir. Böylece uğranılan noktaya bir kez daha uğranmaz. Fonksiyon tüm olasılıkları dener ve en iyi çözümü cevap olarak geri döndürür.

2.4.2. Sırt Çantası Problemi Çözüm Algoritması

Sırt Çantası Problemi, Gezgin Satıcı Problemi'ne benzer şekilde NP sınıfına ait bir problemdir ve bir 0/1 problemi niteliği taşımaktadır [12, 13]. Bu problemde seçilen eşyalar 1, seçilmeyen eşyalar ise 0 olarak işaretlenmektedir. Sırt Çantası Problemi'nde, çantanın taşıma kapasitesini aşmadan, olabilecek en değerli kombinasyon ile çantanın doldurulması hedeflenmektedir. Her eşyanın bir ağırlığı ve değeri vardır.

Bu tezde Sırt Çantası Problemi çözülürken, eşyaların ağırlıkları ve değerleri iki farklı tamsayı türünde veri içeren diziye aktarılmıştır. Diğer bir deyişle, ağırlık ve değer dizisinin aynı

indeksteki değerleri tek bir eşyaya aittir. Problemi çözecek olan fonksiyon rekürsif olarak yazılıp, parametre olarak çantanın taşıma kapasitesi, ağırlık dizisi, değer dizisi ve eşya sayısını almaktadır.

2.4.3. Hanoi Kuleleri Problemi Çözüm Algoritması

Hanoi Kuleleri Problemi ilk olarak Fransız matematikçi Édouard Lucas tarafından 1883 yılında bulunmuştur [14]. Hanoi Kuleleri Problemi, en basit haliyle 3 direk üzerine farklı sayıda disklerin yerleştirilmesi amacını taşır ve temelde 3 basit kural içermektedir. Bu kurallar aşağıdaki gibidir [15]:

1. Bir diskin üzerine kendisinden daha büyük bir disk yerleştirilemez.
2. Aynı anda sadece bir disk üzerinde işlem yapılabilir.
3. İşlemler, disklerin tamamı, başlangıç durumunda olduğu gibi, başlangıç direğinden farklı bir direğe taşınana kadar devam eder.

n diskli Hanoi kuleleri probleminin çözülebilmesi için 2^{n-1} adıma ihtiyaç duyulmaktadır.

Bu tezde Hanoi Kuleleri Problemi rekürsif bir fonksiyon yazılarak çözülmüştür. Parametre olarak disk sayısını ve direk isimlerini almaktadır. Problemin çözümü için performans analizleri, 5, 10, 15 ve 20 disk için gerçekleştirilirken, direk sayısı 3 olarak sabit tutulmuştur.

2.5. LİTERATÜR TARAMASI

Literatürde, programlama dillerinin birbirleri ile karşılaştırılması işlemini gerçekleştiren pek çok farklı çalışma yer almaktadır. Bu kısımda bu çalışmaların bir özeti sunulmuştur.

[16] çalışmasında, ACME, Aesop, C2, Darwin, MetaH, Rapide, SADL, UniCon, Weaves ve Wright yazılım mimarisi tanımlama dilleri farklı anlaşılabilir özellikler, birleştirilebilirlik, arıtma ve izlenebilirlik, heterojenlik, ölçeklenebilirlik, geliştirilebilirlik, dinamizm, kısıtlamalar, fonksiyonel olmayan özellikler, ADL konfigürasyonları gibi farklı noktalara göre karşılaştırılmıştır. Sonuç olarak, her dilin birbirlerine farklı açılardan üstünlük sağladıkları görülmüştür.

[17] çalışmasında, C, C++, Java, Perl, Python, Rexx ve Tcl olmak üzere 7 farklı programlama dili; çalışma süreleri, kod uzunlukları, program güvenilirlikleri, program yapıları ve bellek

tüketimleri açısından karşılaştırılmıştır. Sonuç olarak, Perl, Python, Rexx ve Tcl programlama dillerinde program yazma süresinin, C, C++ ve Java dillerinde program yazmanın yaklaşık olarak yarısı olduğu görülmüştür. Program güvenilirlikleri açısından programlama dilleri arasında büyük bir farklılık gözlemlenememiştir. Tipik komut dizisi programlarının, C ve C++ programlama dillerine göre iki kat bellek tükettiği görülmüştür.

[18] çalışmasında, Java, Python ve Ruby dilleri, iki farklı işletim sistemi (Windows ve Linux) üzerinde; çalışma süreleri, bellek kullanımları, geliştirilen uygulamaların satır sayıları ve karakter sayıları açısından karşılaştırılmıştır. Bu karşılaştırmada, Java programlama dilinin Python ve Ruby programlama dillerine göre daha hızlı sonuca ulaştığı, bellek kullanımları açısından, Ruby programlama dilinin en az bellek tüketen, Python programlama dilinin ise en çok bellek tüketen programlama dili olduğu görülmüştür.

[19] çalışmasında, C++, Fortran 2008, Java, Julia, Python, Matlab, Mathematica ve R programlama dilleri Mac ve Windows ortamlarında karşılaştırılmıştır. Ortaya çıkan sonuçlara göre, C++ ve Fortran programlama dillerinin diğerlerine göre daha hızlı olduğu, Java programlama dilinin C++ programlama dilinden 2,2 ile 2,69 kat daha yavaş olduğu ancak farklı işletim sistemlerinde çalışırken bir fark olmadığı, Julia'nın Java'dan az bir farkla daha hızlı olduğu ve Matlab'tan 4 kat daha hızlı olduğu, Matlab'ın C++'tan 9 ile 11 kat daha yavaş olduğu, Python'ın C++'tan 44 ile 45 kat daha yavaş olduğu, R'ın C++'tan 475 ile 491 kat daha yavaş olduğu görülmüştür.

[20] çalışmasında, C#, Java, Go, Python ve Node.js programlama dilleri kullanılarak ölçeklenebilir web uygulamaları için yük testleri uygulanmıştır. Uygulanan testler sonucunda, en kararlı şekilde isteklere cevap veren uygulamanın Go dili ile geliştirilen WebAPI uygulaması olduğu C# ve Java dillerinin birbirlerine çok yakın performans sergiledikleri, en düşük performansı ise Python programlama dilinde yazılan WebAPI uygulamasının gösterdiği görülmüştür. Node.js uygulaması ise en iyi dördüncü uygulama olmuştur. Uygulamaların en hızlı cevap süreleri karşılaştırıldığında ise en hızlı cevap süresine sahip uygulamaların Python ve Node.js programlama dilleri kullanılarak yazıldığı tespit edilmiştir. Locust uygulamasında uygulanan testler sonucunda ise, 750 sanal kullanıcının üstüne çıkıldığında hem ortalama cevap süresinde, hem de cevaplanabilen toplam istek sayısında düşüş başladığı görülmüştür.

Literatürde kimi zaman en popüler ve genel amaçlı programlama dilleri karşılaştırılırken kimi zaman yalnızca belli bir uzmanlığa hitap eden daha dar amaçlı programlama dilleri ele alınmıştır. Bu gibi çalışmaların amacı, o konu ile ilgili çalışmakta olan araştırmacıları hedef alarak özel olarak o konu ile ilgili en iyi programlama dilinin bulunması için çaba göstermektir. Bu duruma bir örnek olarak gösterilebilecek olan, [21] çalışmasında üç farklı paralel fonksiyonel programlama dili olan PMLS, GPH ve Eden, çalışma süreleri ve kullandıkları çekirdek sayıları açısından karşılaştırılmıştır. Bu üç dilde, matris çarpımı, doğrusal sistem çözücüsü ve ışın takipçisi programları çalıştırılarak karşılaştırma işlemi gerçekleştirilmiştir. Karşılaştırmalar sonucunda, PMLS'in diğerlerine oranla en yavaş çalışan dil olduğu, GPH'ın PMLS ile aynı yapıda olmasına rağmen, problemi paralelleştirme aşamasında, geliştiriciye daha fazla kontrol olanağı sağladığı, Eden'in ise diğer ikisine göre çok daha fazla kontrol olanağı sağladığı ama geliştiriciye de daha fazla efor sarfettirdiği görülmüştür. Yine [22] çalışmasında da, genel özelliğe sahip olmayan, yalnızca veritabanı sistemleri için kullanılan sorgulama dilleri olan Cypher, Gremlin ve SQL karşılaştırılmıştır. Karşılaştırmaya daha sonra, Java programlama dili kullanılarak veritabanına nesne ekleme işlemi de eklenmiştir. Sonuç olarak, en hızlı çözümü Gremlin'in sağladığı görülmüştür. Gremlin'den sonra sırasıyla Java ve Cypher gelmiştir. Ayrıca her biri RESTful web servisler kullanılarak da test edilmiştir. Cypher'ın programlama ve SQL bilen biri için Gremlin'den çoğu konuda daha iyi olduğu anlaşılmıştır. Ancak performans açısından Gremlin'in daha iyi olduğu da belirtilmiştir. Java programlama dili kullanılarak veritabanına veri girişi yapılırken, Cypher'ın yarı yarıya yavaş olduğu da görülmüştür.

Bazı çalışmalarda belli bir ortak özelliğe sahip kişiler tarafından gerçekleştirilen programlar baz alınarak karşılaştırmalar yapılmıştır. Örneğin [23] çalışmasında, yeni başlayan mühendislik fakültesi öğrencileri üç farklı gruba ayırılıp, programlama derslerinde C, Matlab ve Python programlama dilleri öğretilmiştir. Bu sayede bu dillerin öğrenme kolaylığı tespit edilmeye çalışılmıştır. Sonuç olarak Python programlama dilinin öğrenmeye en elverişli dil olduğu, Matlab'ın ise C'ye göre programlama yapısını daha iyi öğrettiği görülmüştür. Benzer şekilde [24] çalışmasında, bir müzeyi ziyaret eden 260 ziyaretçiden basit robotik programlar tasarlamaları istenmiştir. Bu programları önce Lego parçaları gibi üzerinde programlama ile ilgili etiketleri olan parçaları birleştirerek tasarlayıp sonrasında grafik arayüzünde gerçekleştirmeleri beklenmiştir. Analizler gerçekleştirilirken ziyaretçiler, önce çocuk ve yetişkin olarak daha sonra da bu iki grup cinsiyetlerine göre sınıflandırılmıştır. Grafik

arayüzündeki çalışmalar, somut parçalar kullanılarak gerçekleştirilen çalışmalara göre sadece yetişkin erkeklerde daha başarılı sonuçlar vermiştir. Geriye kalan tüm ziyaretçi tiplerinde somut parçalar kullanılarak daha iyi sonuçlar elde edilmiştir.

Literatürdeki bazı çalışmalarda, programlama dilleri yalnızca belli bir alandaki programlar ele alınarak karşılaştırılmıştır. Örneğin [25] çalışmasında; C, C++, C#, Java, Perl ve Python programlama dilleri, yalnızca biyoenformatikte sıkça kullanılan çeşitli programlar gerçeklenilerek karşılaştırılmıştır. Bu karşılaştırmalar çalışma süreleri, bellek kullanımları, kod uzunlukları açısından gerçekleştirilmiştir. Ayrıca karşılaştırmalar Windows ve Linux ortamlarında ayrı ayrı ele alınmıştır. Analizler sonucunda, C ve C++ programlama dilleri en az bellek kullanımına ve en hızlı çalışma süresine sahip programlama dilleri en hızlı diller olmuştur. Ayrıca C ve C++ programlama dillerinin kod uzunluklarının diğer programlama dillerine göre daha büyük olduğu görülmüştür. Programlama dillerinin performanslarının işletim sistemlerine göre belirgin bir fark göstermediği tespit edilmiştir.

Bu tez çalışmasında da yukarıdaki çalışmalara benzer olarak çeşitli matematiksel problemler ve algoritmaların, farklı programlama dilleri ve platformlar kullanılarak ve farklı kriterler için karşılaştırılması hedeflenmiştir.

3. MALZEME VE YÖNTEM

Bir bilgisayar yazılımının performansı bir çok etkene bağlıdır. Bu etkenler temel olarak iki ana kategoride incelenebilir. Bunlardan ilki programlama ekibinin sahip olduğu kişisel özellikler, ikincisi ise bu tezde ele alındığı gibi teknik özelliklerdir. Bu tez kapsamında, bilgisayar yazılımlarının performansını etkileyen teknik özellikler olarak programlama dilleri, programların geliştirildiği platformlar ve algoritmaların tasarım şekilleri ele alınmıştır.

3.1. PROGRAMLAMA DİLLERİ

Bu tez çalışmasında ele alınan algoritmalar, C, C#, Java ve Python programlama dilleri kullanılarak gerçeklenmiş ve bu programlama dillerinin performans analizleri gerçekleştirilmiştir. Bu tezde bu dillerin seçilme sebebinin anlaşılabilmesi açısından bu kısımda C, C#, Java ve Python programlama dillerinden kısaca bahsedilmiştir.

3.1.1. C Programlama Dili

C programlama dili Dennis Ritchie ve Ken Thompson tarafından 1972 yılında geliştirilmiş genel amaçlı bir programlama dilidir [26]. C programlama dili makine dilinden daha gelişmiş ancak modern dillere göre daha ilkel bir dil olarak kabul edilir. Bu durum programcılara esneklik katarken, bazı durumlarda çeşitli zorluklar da çıkartabilmektedir. C programlama dilinin sözdizimi sonrasında gelen bir çok programlama diline ilham vermiştir.

3.1.2. C# Programlama Dili

C# programlama dili Microsoft bünyesinde, başında Anders Hejlsberg bulunan bir geliştirme ekibinin tarafından geliştirilmiştir [27]. C# programlama dili, C programlama dilinin aksine dönüştürülme işlemine giren ve ara bir dile (Microsoft Ara Dili) dönüşen, nesne yönelimli bir programlama dilidir. Orta seviyeli bir dil olarak kabul edilen C#, hem doğal dillere hem de makine diline yakın bir özellik gösterir. Ayrıca, C# programlama dilinin sahip olduğu Çöp Toplayıcısı sayesinde geliştirilen uygulamaların bellek kullanımları azaltılır.

C# programlama dilinde yazılan bir programın çalışabilmesi için, çalıştırılacak bilgisayarlarda .NET Framework'ün bulunması gerekmektedir. C# programlama dili kullanılarak, mobil

uygulamalar, bulut uygulamaları, veri tabanı uygulamaları, masaüstü uygulamaları, web uygulamaları gibi çeşitli uygulamalar geliştirilebilmektedir.

3.1.3. Java Programlama Dili

Java programlama James Gosling tarafından 1995 yılında geliştirilmeye başlanmıştır ve hala gelişimini sürdürmektedir [28]. C# gibi nesne yönelimli programlama dili olan Java'nın C#'tan en önemli farkı üzerinde çalıştığı platformdan bağımsız bir programlama dili olmasıdır. C# programlama dili sadece Windows platformunda çalışabilirken, Java programlama dili ile yapılan bir uygulamanın bir bilgisayarda çalıştırılabilmesi için Java Sanal Makinesi (Java Virtual Machine – JVM) bulunması yeterlidir. Yine, C# programlama dilinde olduğu gibi Java programlama dilinde de Çöp Toplayıcısı bulunmaktadır.

Java programlama dilinin bir diğer avantajı, pek çok farklı uygulamaya uygun olmasıdır. Diğer bir deyişle, Java programlama dili kullanılarak, web uygulamaları, web servisleri, masaüstü uygulamaları, mobil uygulamalar gibi çeşitli uygulamalar geliştirilebilmektedir.

3.1.4. Python Programlama Dili

Python programlama dili Guido van Rossum tarafından 1990 yılında geliştirilmeye başlanmıştır [29]. Python programlama dili günümüzde modern diller arasında en popüler dillerden biri haline gelmiştir. Bunun en büyük sebeplerinden biri çok geniş ve kapsamlı bir kütüphanesi olması, ayrıca Python programlama dili kullanılarak yazılan kodları okumanın daha kolay olmasıdır. C# ve Java programlama dillerinde olduğu gibi, Python programlama dili de Çöp Toplayıcısına sahiptir. Python programlama dilinde geliştirilen uygulamalar da Java programlama dili gibi bir çok platformda çalıştırılabilir. Diğer bir deyişle, Python programlama dili, ağ programlama, web uygulamaları, oyun programlama, veri tabanı uygulamaları gibi çeşitli çalışmalara olanak sağlamaktadır.

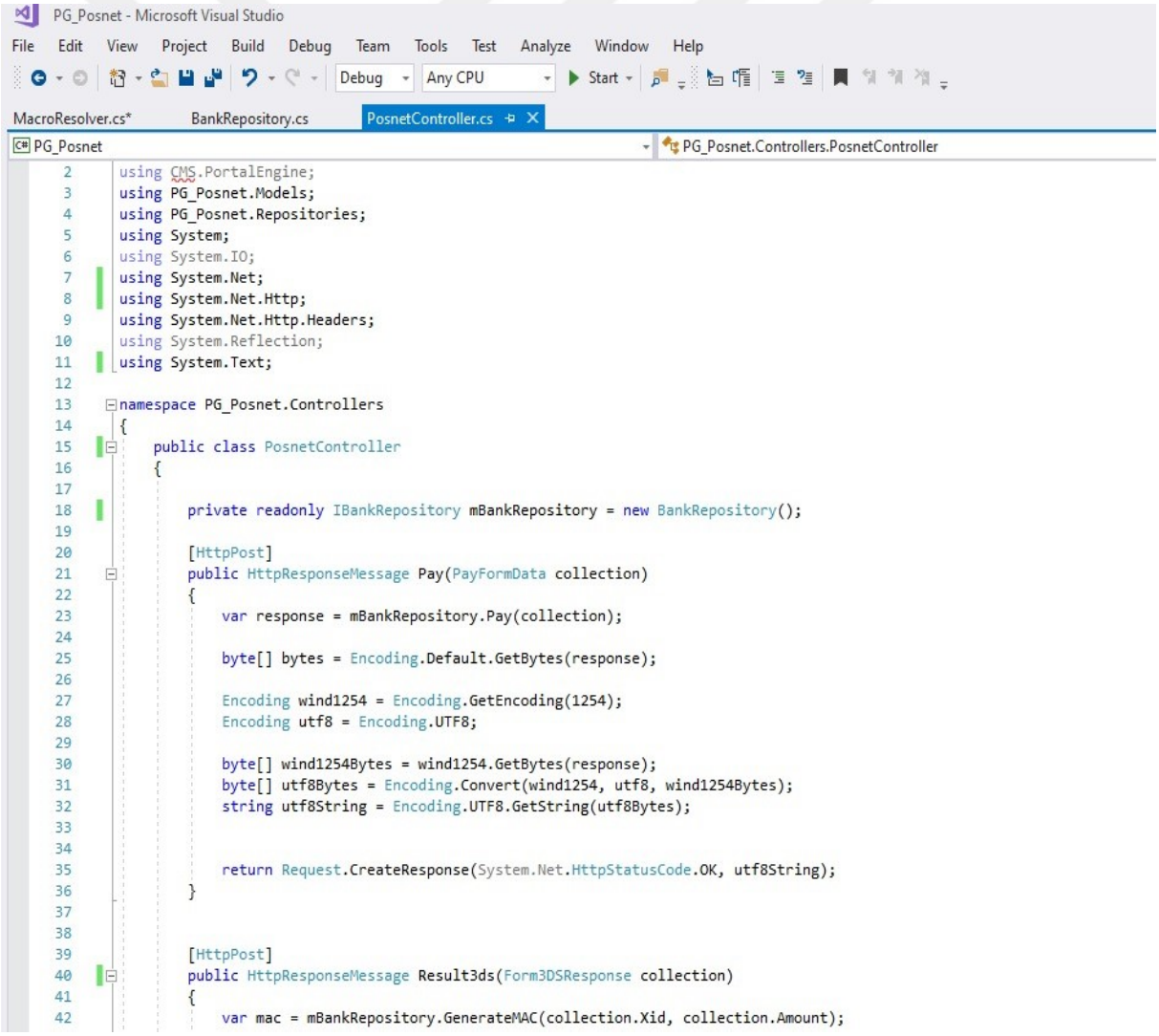
3.2. PROGRAMLAMA PLATFORMLARI

Bu tez çalışmasında ele alınan algoritmalar, farklı programlama dilleri kullanılarak gerçekleştirildiği için farklı programlama platformlarının da kullanılması gerekmiştir. C programlama dili ile kodlanan programlar, Visual Studio ve Code::Blocks platformlarında; C# programlama dili ile kodlanan programlar, Visual Studio platformunda; Java programlama

dili ile kodlanan programlar Net Beans platformunda; Python programlama dili ile kodlanan programlar ise JetBrains PyCharm platformunda gerçeklenmiştir. Bu bölümde bu programlama platformları hakkında kısaca bilgi verilmiştir.

3.2.1. Visual Studio

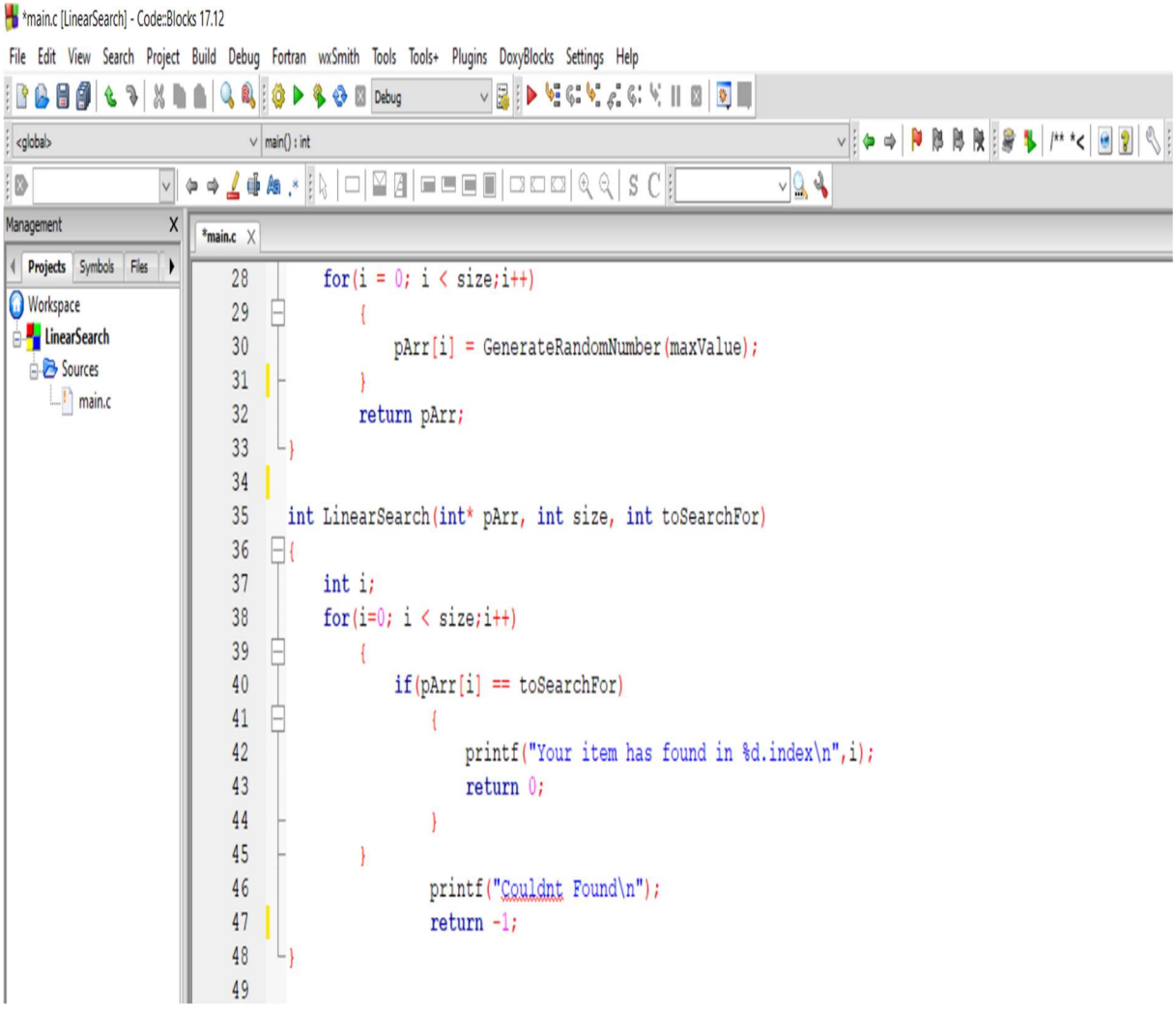
Visual Studio Microsoft tarafından geliştirilen Windows İşletim Sistemine uyumlu, üzerinde C, C++, C#, Ruby, Node.js ve JavaScript gibi pek çok programlama dilinin çalışabildiği çok yönlü bir programlama platformudur [30]. Bu tez çalışmasında C ve C# programlama dilleri kullanılarak geliştirilen tüm algoritmalar Visual Studio ortamında gerçeklenmiştir. Şekil 3.1'de bir Visual Studio ortamı görüntüsü verilmiştir.



Şekil 3.1: Visual Studio Platformundan Alınan Ekran Görüntüsü

3.2.2. Code::Blocks

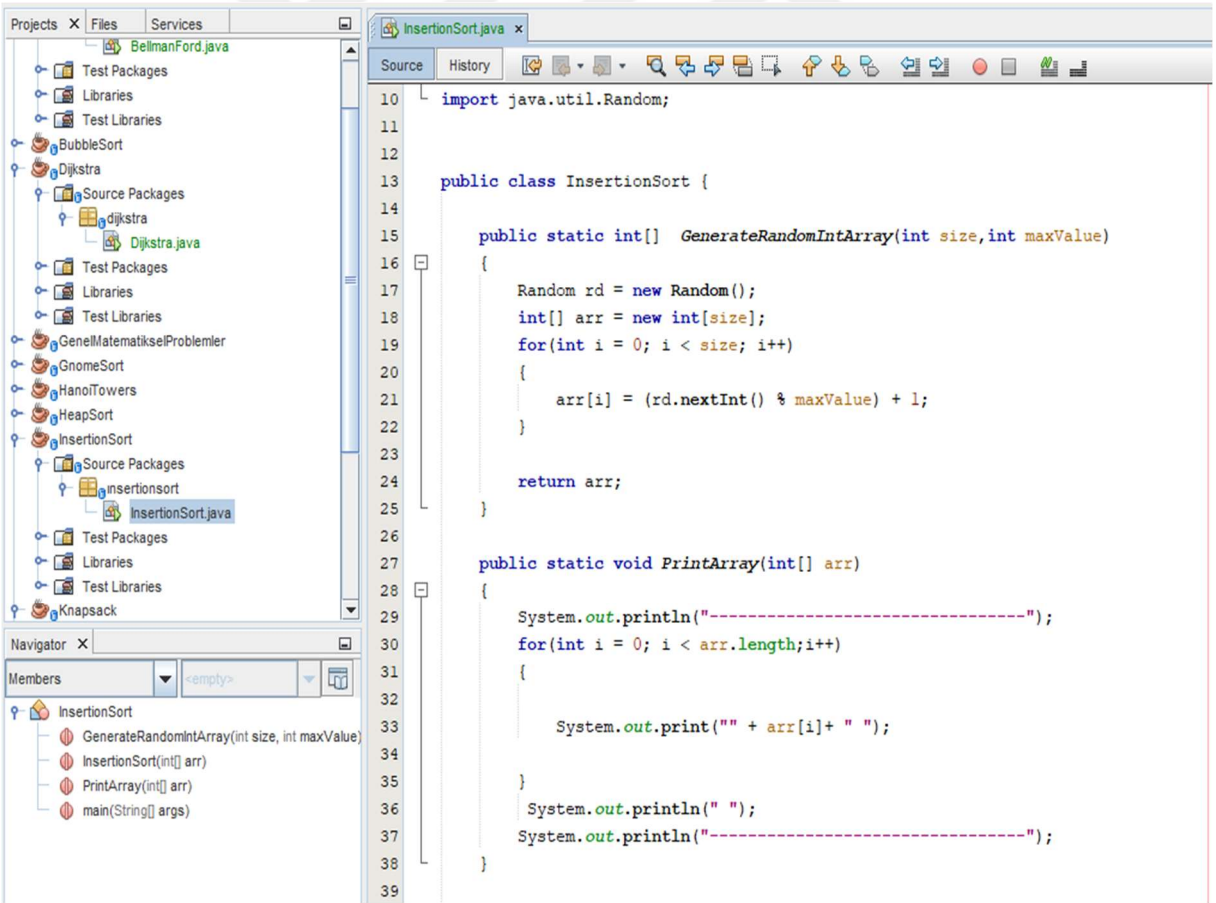
Code::Blocks, açık kaynak kodlu, C, C++ ve Fortran dillerinde yazılım geliştirilebilecek bir platformdur [31]. Bu tez çalışmasında C programlama dili kullanılarak geliştirilen tüm algoritmalar, Visual Studio ortamının yanı sıra Code::Blocks ortamında gerçeklenmiştir. Bunun sebebi, Visual Studio platformunun .NET Framework'ü üzerinde çalışmasının, C programlama dili kullanılarak kodlanan programların performansına etki edebileceğinin düşünülmesidir. Böyle bir etkinin mevcut olup olmadığının görülmesi ve gerçekleştirilen programların programlama platformuna bağımlı olup olmadığının tespit edilmesi için iki farklı programlama platformunun kullanılması, tezin doğruluğu açısından uygun görülmüştür. Şekil 3.2'de bir Code::Blocks ortamı görüntüsü verilmiştir.



Şekil 3.2: Code::Blocks Platformundan Alınan Ekran Görüntüsü

3.2.3. NetBeans

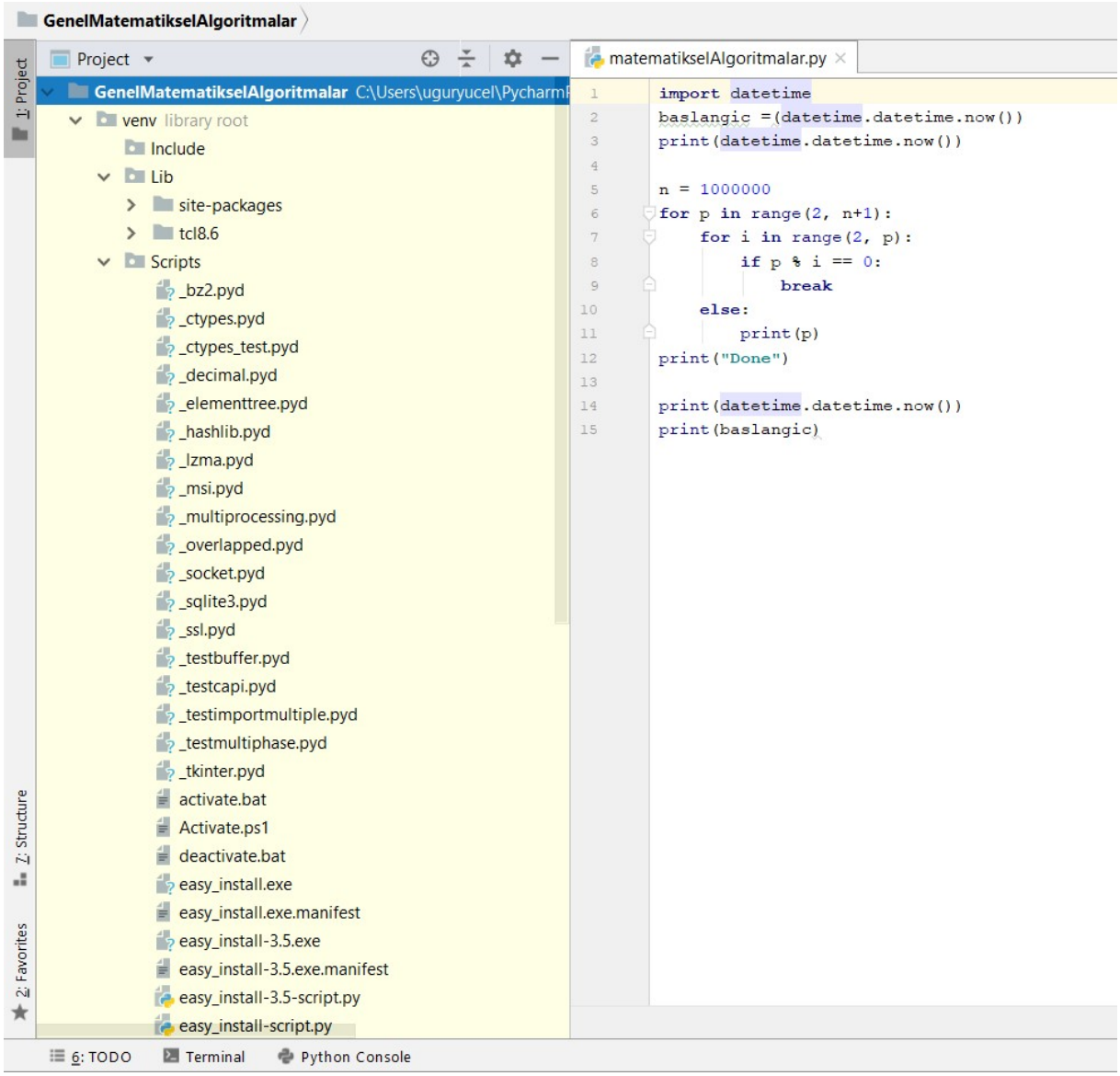
NetBeans, Apache Software Foundations tarafından desteklenen bedava ve açık kaynak kodlu platformdur [32]. Masaüstü uygulamaları, mobil uygulamalar, web uygulamaları gibi çeşitli uygulamaların geliştirilebildiği bir ortam olan NetBeans, uygulama çeşitliğinde olduğu gibi, kullanılacak programlama dili çeşitliliğinde de kullanıcıya özgürlük sağlamaktadır. Bu sebeple birçok yazılım geliştiricinin sıklıkla kullanmakta olduğu bir platform olan NetBeans bu tezde de kendisine yer bulmuştur. Bu tez çalışmasında Java programlama dili kullanılarak geliştirilen tüm algoritmalar NetBeans ortamında gerçeklenmiştir. Şekil 3.3'te bir NetBeans ortamı görüntüsü verilmiştir.



Şekil 3.3: NetBeans Platformundan Alınan Ekran Görüntüsü

3.2.4. Jet Brains PyCharm

Jet Brains şirketi tarafından PyCharm bütün işletim sistemleri üzerinde çalışabilen Python dilinde program geliştirilmesini sağlayan platformdur [33]. Bu tez çalışmasında Python programlama dili kullanılarak geliştirilen tüm algoritmalar PyCharm ortamında gerçeklenmiştir. Şekil 3.4'te bir PyCharm ortamı görüntüsü verilmiştir.



Şekil 3.4: JetBrains PyCharm Platformundan Alınan Ekran Görüntüsü

3.3. ALGORİTMA TASARLAMA YAKLAŞIMLARI

Bir bilgisayar yazılımının performansını, yazılımın gerçeklendiği programlama dilleri ve programlama platformları gibi, yazılımın algoritmasının tasarlanma şekli de etkileyebilmektedir. Bir algoritma tasarlanırken algoritmanın iteratif ya da rekürsif olarak tasarlanması, algoritmanın karmaşıklığını dahi değiştirebilecek önemli bir husustur. Bu tez çalışmasında, bazı algoritmalar, iteratif ve rekürsif olarak gerçeklenmiş ve her yaklaşımın performans analizleri ayrı ayrı gerçekleştirilmiştir. Ayrıca bu iki yaklaşıma ek olarak, bazı problemlerin çözüm algoritmalarında kullanıldığı için Geri İzlemeli Algoritma tasarımı da ele alınmıştır.

3.3.1. İteratif Algoritma Tasarımı

Yazılım geliştirme esnasında bir işlem birden fazla kez tekrarlanacaksa döngü kullanımı tavsiye edilmektedir. Bunun sebebi ise aynı işlemin kodunun tekrar tekrar yazmak yerine bir döngü kullanılarak gerçekleştirilebiliyor olmasıdır. Döngü kullanımının bir diğer avantajı ise diziler, bağlı listeler, ağaçlar veya çizgeler gibi veri yapıları üzerinde kolayca işlem yapılabilmesidir. Bu tezde gerçeklenen hemen her algoritmanın kodu döngü içermektedir.

3.3.2. Rekürsif Algoritma Tasarımı

Rekürsif (özyinelemeli) algoritmalar kendi içerisinde kendini daha basit değerlerle tekrar eden algoritmalardır. Rekürsif algoritmalar tasarlanırken bir durma kriteri olmasına dikkat edilir. Rekürsif algoritmalar çalışırken her çağrılışlarında bir önceki değerlerinin bir yerlerde tutulması gerekmektedir. Bu değerler bilgisayar hafızasında yığın bellekte tutulmaktadır.

3.3.3. Geri İzlemeli Algoritma Tasarımı (Backtracking)

Geri izlemeli algoritmalar genellikle bir çözümü bulana kadar, bir sonraki adımı deneyerek devam eden ve çözümü sağlamayan durumları eleyerek devam eden rekürsif algoritmalardır. Çözümü sağlamayan durumları elemesi nedeniyle Geri İzlemeli Algoritma adını almıştır.

4. BULGULAR

Bu tezde gerçekleştirilen performans analizleri, programlama dilleri ve dizi boyutlarına göre; CPU kullanım yüzdesi, maksimum bellek kullanımı ve çalışma süresi açısından gerçekleştirilmiştir. Programlama dili olarak C, C#, Java ve Python dilleri; geliştirme ortamı olarak Visual Studio, Code::Blocks, NetBeans ve JetBrains PyCharm kullanılmıştır.

C programlama dili ile yapılan analizlerde algoritmalar iki farklı geliştirme ortamı (Visual Studio – Code::Blocks) kullanılarak kodlanmıştır. Visual Studio ortamının kullanılma sebebi, Performans Profil Oluşturucusu'na sahip olması ve bu tezde yapılan ölçümleri gerçekleştirmek için ortam sağlamasıdır. Code::Blocks ortamının kullanılma sebebi ise, C kodunun yazılabilmesi için daha elverişli bir ortam oluşturulmasıdır. CPU kullanımı ve bellek kullanımı değerleri ölçülürken Visual Studio programının Performans Profil Oluşturucusu kullanılmıştır.

C# programlama diline ait ölçümler, Visual Studio 2017 yazılım geliştirme ortamında alınmıştır. C# dilinin C dilinden en önemli farkı nesne yönelimli programlama dilleri ailesinin mensubu olmasıdır. Nesne yönelimli programlamada oluşturulan her sınıf Object sınıfının alt sınıfıdır. Programcı açısından işleri bir çok yönden kolaylaştırırken, bellek kullanımı açısından bilgisayar için durumu biraz zorlaştırmaktadır. C# programlama dilinin, yukarıda belirtildiği üzere bir nesne yönelimli programlama dili olması dolayısıyla, sıralama algoritmalarının C# programlama diliyle yapılmış performans analizlerini gösteren tablolarda, C programlama dilinin performans analizlerini gösteren tablolara ek olarak Çöp Toplama sütunu bulunmaktadır. Çöp Toplama işlemi manuel bellek yönetiminin tam tersidir. C dilinde malloc() fonksiyonu kullanılarak belleğe yerleştirilen bir değişken, free() edilmedikçe veya üzerine başka bir veri yazılmadıkça o değişken bellekte kalmaya devam eder. Ancak Çöp Toplama mekanizması bellekte olan ama uzun bir süredir kullanılmayan, kullanılıp işi biten nesneleri bellekten silerek yeni nesnelere yer açar.

Java programlama dili, yapı olarak C# programlama diliyle benzer özellik gösterdiğinden, bu tezde Java programlama diliyle gerçekleştirilen analizlerde tablo sütunları C# programlama diliyle yapılan analizlerdeki sütunlarla aynıdır. Bu analizlerde, NetBeans geliştirme ortamı kullanılmıştır. Bunun sebebi ise NetBeans Profil Oluşturucusu'nun bu tezde yapılan ölçümlerde gereken özellikleri sağlıyor olmasıdır.

Python programlama dili kullanılarak gerçekleştirilen analizlerde C# ve Java programlama dillerinin analizlerinden farklı olarak Windows ortamında çalışan Profil Oluşturucusu bulunamamıştır. Bu nedenle, Python programlama dili kullanılarak yapılan analizler sonucunda oluşturulan tablolar 2 sütundan oluşmaktadır. Bunlar sayı dizisinin eleman sayısı ve çalışma süresidir.

Her algoritma sınıfında programlama dilleri ve platformlar dışında o algoritma sınıfının performansını etkileyebilecek çeşitli kriterler bulunmaktadır. Performans analizleri gerçekleştirilirken algoritma sınıfının bu kriterleri de ele alınmıştır.

Arama ve sıralama algoritmalarında performans analizini etkileyen en önemli kriterlerden bir tanesi veri boyutudur. Bu durum göz önünde bulundurularak, bu tezde veriler bu iki algoritma sınıfının analizlerinde sayı dizisi şeklinde ele alınmıştır. Dizi boyutu ise 10^n $(n = 2, 3, 4, 5, 6)$ olacak şekilde, 100, 1000, 10000, 100000 ve 1000000 olarak test edilmiştir. Algoritmalar çalıştırılırken öncelikle üzerinde çalışılacak sayı dizileri oluşturulmuş ve dizinin elemanlarına 0 ile 1000 arasında rastgele değerler atanmıştır. Rastgele atama işlemi programlama dillerinin Random sınıflarını kullanarak veya rnd() ve benzeri fonksiyonlar kullanılarak gerçekleştirilmiştir. Arama algoritmalarında performans analizi yapılırken dizinin son elemanının indeksine, dizinin içerisinde bulunmadığından emin olunan 1001 değeri atanmıştır. Böylece arama işlemi en kötü duruma göre gerçekleştirilmiş olur. Sıralama algoritmalarında ise dizinin başarılı bir şekilde sıralandığını görebilmek için dizi önce karışık haliyle, daha sonra da sıralı haliyle ekrana yazdırılmıştır.

Polinomsal olmayan algoritmaların kriterleri kendi içlerinde çeşitlenmektedir. Gezgin Satıcı Problemi Çözüm Algoritması'nda, problemin karmaşıklığını arttıracak ve performansını etkileyecek olan veri düğüm sayısı olduğundan farklı düğüm sayıları için algoritma gerçeklenmeye çalışılmıştır. Yapılan analizlerde, düğüm sayısı 5, 10, 15, 20 olacak şekilde artırılmıştır. Literatürdeki çalışmalara bakıldığında düğüm sayısı 20'nin üzerine çıkıldığında problemin polinomsal zamanda çözülemediği görülmüştür. Bu problemin çözümünde Global Karınca Kolonisi Algoritması, Genetik Algoritmalar, gibi bir çok yaklaşım olmasına rağmen, bu tezde geri izleme yaklaşımı seçilmiştir. Bunun sebebi, geri izleme yaklaşımının belleği ve CPU'yu daha fazla kullanacak olmasından ötürü, Profil Oluşturucularının değerleri daha net bir biçimde belirleyebilecek olmasıdır.

Bilgisayar donanımının simülasyon sonuçlarını etkilememesi için her birinde aynı bilgisayar sistemi kullanılmıştır. Bu sistemin özellikleri aşağıdaki gibidir:

- Intel® Core i7-7700HQ, 2.8GHz İşlemci
- 16 GB RAM
- 64 Bit Windows 10 Pro İşletim Sistemi

Bu tezdeki simülasyonlar için C programlama dilinde yazılan kodlar Visual Studio 2017 Professional ve Code::Blocks 17.12 ortamlarında geliştirilmiş ve çalıştırılmıştır. Java programlama dilinde yazılan kodlar NetBeans platformunun 8.2 versiyonunda geliştirilmiş ve çalıştırılmıştır. C# programlama dilinde yazılan kodlar için de Visual Studio 2017 Professional ortamı kullanılmıştır. Python programlama dilinde yazılan kodlar için JetBrains PyCharm Community ortamının 2018.2.4 versiyonu kullanılmıştır.

4.1. GENEL MATEMATİKSEL ALGORİTMALARIN UYGULAMALARININ PERFORMANS ANALİZİ

Bu bölümde Faktöriyel Hesaplama Algoritması, Üs Alma Algoritması ve Asal Sayı Bulma Algoritması incelenmiş ve uygulamalarının performans analizleri gerçekleştirilmiştir.

4.1.1. Faktöriyel Hesaplama Algoritması Uygulamasının Performans Analizi

Bu kısımda Faktöriyel Hesaplama Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizler gerçekleştirilmiştir. Faktöriyel algoritması daha önce de belirtildiği üzere iteratif ve rekürsif olarak simule edilmiştir. İlk olarak C programlama dilindeki sonuçlar elde edilmiştir. 16! değeri C programlama dilindeki int değişken tipinin en üst değerini aşmayan en büyük sayı olduğu için üst sınır olarak alınmıştır. Sırasıyla, iteratif ve rekürsif tasarım simülasyonlarının C programlama dili kullanılarak gerçekleştirilen analiz sonuçları, Tablo 4.1 ve Tablo 4.2'de, C# programlama dili kullanılarak gerçekleştirilen analiz sonuçları Tablo 4.3 ve Tablo 4.4'te, Java programlama dili kullanılarak gerçekleştirilen analiz sonuçları Tablo 4.5 ve Tablo 4.6'da, Python programlama dili kullanılarak gerçekleştirilen analiz sonuçları Tablo 4.7 ve Tablo 4.8'de verilmiştir.

Tablo 4.1: İteratif Faktöriyel Hesaplama Algoritması için C Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (ms) (Visual Studio)	Çalışma Süresi (ms) (Code::Blocks)
5	*	*	216	44
10	*	*	308	46
15	*	*	438	46
16	*	*	508	46

* Çok küçük değer olduğu için tespit edilememiştir.

Tablo 4.2: Rekürsif Faktöriyel Hesaplama Algoritması için C Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (ms) (Visual Studio)	Çalışma Süresi (ms) (Code::Blocks)
5	*	*	271	45
10	*	*	341	45
15	13	6,1	421	45
16	13	6,3	606	46

* Çok küçük değer olduğu için tespit edilememiştir.

Yukarıdaki iki tablo incelendiğinde arama algoritmalarının ne kadar az kaynak tüketen algoritmalar olduğu görülmüştür. Çoğu zaman, bellek ve CPU kullanımları neredeyse 0'a yakındır. Bunun yanı sıra platformlar arasında bir karşılaştırma yapıldığında; Code::Blocks'un çok daha hızlı olduğu görülmüştür.

Tablo 4.3: İteratif Faktöriyel Hesaplama Algoritması için C# Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
5	13	13,2	0	26
10	14	13,3	0	45
15	14	13,3	0	52
16	17	13,3	0	53

Yukarıda C# programlama dili kullanılarak yapılan bu analiz sonucunda, sayı artsa da bellek kullanımlarında yüksek değişiklikler olmadığı gözlemlenmiştir ve bellek kullanımları düşük seviyelerde olduğu için Çöp Toplama Mekanizması'nın devreye girmediği görülmüştür.

Tablo 4.4: Rekürsif Faktöriyel Hesaplama Algoritması için C# Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
5	13	12,8	0	20
10	13	12,9	0	22
15	15	13,4	0	23
16	16	13,8	0	23

Yukarıda C# programlama dili kullanılarak yapılan bu analiz sonucunda, rekürsif algoritmanın, iteratif algoritmaya göre daha hızlı çalıştığı görülmüştür. Ayrıca bir önceki tabloya göre, sayılar arttıkça bellek kullanımı değerlerinin de arttığı bu tabloda daha net bir şekilde görülmektedir.

Tablo 4.5: İteratif Faktöriyel Hesaplama Algoritması için Java Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
5	*	10,58	0	12
10	*	10,58	0	13
15	*	10,58	0	14
16	*	10,58	0	14

* Çok küçük değer olduğu için tespit edilememiştir.

Tablo 4.6: Rekürsif Faktöriyel Hesaplama Algoritması için Java Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
5	*	10,58	0	4
10	*	10,58	0	9
15	*	10,58	0	25
16	*	10,58	0	25

* Çok küçük değer olduğu için tespit edilememiştir.

Yukarıdaki iki tablo incelendiğinde, CPU ve bellek kullanımlarının birbirlerine eşit olduğu görülmektedir. C# programlama dili kullanılarak gerçekleştirilen performans analizinde, özyinelemeli fonksiyonların, iteratif fonksiyonlara göre daha hızlı sonuçlandığı görülmüştür. Java programlama dilinde ise bu durum veri küçük iken aynı şekildeyken, veri büyüdükçe tam tersine dönmektedir.

Tablo 4.7: İteratif Faktöriyel Hesaplama Algoritması için Python Programlama Dili Performans Analizi.

Sayı	Çalışma Süresi (ms)
5	0,001
10	0,001
15	0,001
16	0,001

Tablo 4.8: Rekürsif Faktöriyel Hesaplama Algoritması için Python Programlama Dili Performans Analizi.

Sayı	Çalışma Süresi (ms)
5	0,001
10	0,001
15	0,001
16	0,001

Yukarıdaki iki tablo incelendiğinde, ikisinin de birebir aynı değerleri verdiği görülmektedir. Buna göre Python programlama dilinde, algoritmanın iteratif ya da rekürsif olmasının bir fark yaratmadığı gözlemlenmiştir. Ancak, Python programlama dili, diğer programlama dilleri ile karşılaştırıldığında, C, C# ve Java programlama dillerine göre küçük veriler üzerinde işlem yapma konusunda daha hızlı olduğu da gözlemlenmektedir.

4.1.2. Üs Alma Algoritması Uygulamasının Performans Analizi

Bu kısımda Üs Alma Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizler gerçekleştirilmiştir. Üs Alma Algoritması iteratif ve rekürsif olarak simule edilmiştir. Üs Alma Algoritması ölçümlerinde taban sayısı olarak 2 alınmıştır. Faktöriyel Hesaplama Algoritması ölçümleri sırasında da karşılaşılan int (integer – tamsayı) değişken tipinin en büyük değerini aşmayan en büyük sayı 30'dur. Sırasıyla iteratif ve rekürsif tasarım simülasyonlarının C programlama dili kullanılarak gerçekleştirilen analiz sonuçları, Tablo 4.9 ve Tablo 4.10'da, C# programlama dili kullanılarak gerçekleştirilen analiz sonuçları Tablo 4.11'de ve Tablo 4.12'de, Java programlama dili kullanılarak gerçekleştirilen analiz sonuçları Tablo 4.13 ve Tablo 4.14'te, Python programlama dili kullanılarak gerçekleştirilen analiz sonuçları Tablo 4.15 ve Tablo 4.16'da verilmiştir.

Tablo 4.9: İteratif Üs Alma Algoritması için C Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (ms) (Visual Studio)	Çalışma Süresi (ms) (Code::Blocks)
5	*	*	*	39
10	*	*	*	43
15	*	6,2	502	46
20	13	6,2	567	48
25	13	6,3	606	53
30	13	6,5	616	67

* Çok küçük değer olduğu için tespit edilememiştir.

Tablo 4.10: Rekürsif Üs Alma Algoritması için C Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (ms) (Visual Studio)	Çalışma Süresi (ms) (Code::Blocks)
5	*	*	448	42
10	*	*	508	51
15	*	*	564	54
20	13	5,8	610	56
25	13	5,9	633	60
30	13	6,2	1013	60

* Çok küçük değer olduğu için tespit edilememiştir.

Yukarıdaki iki tablo incelenecek olursa, Code::Blocks platformunun Visual Studio'dan daha hızlı sonuç verdiği bir kez daha görülmüştür. Hem iteratif hem de rekürsif olarak gerçekleştirilen üs alma fonksiyonu sonucunda, CPU ve bellek kullanım değerleri birbirine çok yakın olarak tespit edilmiştir.

Tablo 4.11: İteratif Üs Alma Algoritması için C# Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
5	14	12,7	0	17
10	14	12,9	0	22
15	15	12,9	0	23
20	15	12,9	0	24
25	15	13,4	0	25
30	17	13,5	0	29

Tablo 4.12: Rekürsif Üs Alma Algoritması için C# Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
5	13	13,2	0	24
10	14	13,2	0	25
15	14	13,3	0	25
20	15	13,3	0	25
25	15	13,4	0	26
30	15	13,4	0	28

Yukarıdaki iki tablo incelendiğinde, işlemin tekrarlanma sayısı küçük iken, iteratif algoritma rekürsif algoritmaya göre daha hızlı çalışmakta olduğu, tekrarlanma sayısı arttıkça bu durumun tam tersine döndüğü görülmüştür.

Tablo 4.13: İteratif Üs Alma Algoritması için Java Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
5	*	10,58	0	5
10	*	10,58	0	9
15	*	10,58	0	11
20	*	10,58	0	11
25	*	10,58	0	11
30	*	11,94	0	14

* Çok küçük değer olduğu için tespit edilememiştir.

Tablo 4.14: Rekürsif Üs Alma Algoritması için Java Programlama Dili Performans Analizi.

Sayı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
5	*	10,58	0	11
10	*	10,58	0	12
15	*	10,58	0	12
20	*	10,58	0	13
25	*	10,58	0	15
30	*	10,58	0	15

* Çok küçük değer olduğu için tespit edilememiştir.

Java dili kullanılarak yapılan bu analizler sonucunda, CPU kullanım değerlerinin çok küçük olduğu ve bellek kullanımlarının da olabilecek en küçük seviyede olduğu (iteratif algoritmanın uygulanması sırasında 2^{30} sayısındaki bellek kullanımı haricinde kalanların hepsi) görülmüştür. Tekrarlanma sayısı küçük iken iteratif algoritmanın, rekürsif algoritmaya göre daha hızlı iken, tekrarlama sayısı arttıkça rekürsif algoritmaların, iteratif algoritmalara göre hızlandığı gözlemlenmiştir.

Tablo 4.15: İteratif Üs Alma Algoritması için Python Programlama Dili Performans Analizi.

Sayı	Çalışma Süresi (ms)
5	0,001
10	0,001
15	0,001
20	0,001
25	0,001
30	0,001

Tablo 4.16: Rekürsif Üs Alma Algoritması için Python Programlama Dili Performans Analizi.

Sayı	Çalışma Süresi (ms)
5	0,001
10	0,001
15	0,023
20	0,1
25	0,1
30	0,1

Yukarıdaki tablolara bakıldığında, işlemin tekrarlanma sayısı arttıkça rekürsif algoritmanın yavaşladığı, ancak iteratif algoritmanın yavaşlamadığı görülmektedir.

4.1.3. Asal Sayı Bulma Algoritması Uygulamasının Performans Analizi

Asal Sayı Bulma Algoritması uygulanırken, iki adet fonksiyon yazılmıştır. İlki parametre olarak verilen sayının asal olup olmadığını kontrol eden, ikincisi ise üst sınır değerini parametre olarak alan ve 2'den itibaren o sayıya kadar olan bütün asal sayıları ekrana yazdıran fonksiyondur.

Bu kısımda Asal Sayı Bulma Algoritmasının C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla Tablo 4.17' de C programlama dili, Tablo 4.18'de C# programlama dili, Tablo 4.19'da Java programlama dili ve Tablo 4.20'de Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.17: Asal Sayı Bulma Algoritması için C Programlama Dili Performans Analizi.

Üst Sınır	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
100	12	*	580ms	80ms
1000	13	6,2	808ms	519ms
10000	13	6,3	1s 424ms	2s 424ms
100000	13	6,6	6s 795ms	11s 531ms
1000000	13	6,8	1dk 15s 808ms	1dk 47s 812ms

* Çok küçük değer olduğu için tespit edilememiştir.

Yukarıdaki tablo incelendiğinde, veri sayısı büyüdükçe Code::Blocks platformunda çalıştırılan asal sayı bulma fonksiyonunun, Visual Studio platformunda çalıştırılan platforma göre yavaşladığı gözlemlenmektedir.

Tablo 4.18: Asal Sayı Bulma Algoritması için C# Programlama Dili Performans Analizi.

Üst Sınır	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	14	13,0	0	23ms
1000	15	13,3	0	218ms
10000	15	13,5	0	1s 365ms
100000	18	13,8	0	9s 835ms
1000000	18	16,3	0	1dk 45s 230ms

Asal sayı bulma algoritması için C# programlama dili kullanılarak yapılan analiz sonucunda, üst sınır arttıkça, CPU kullanımlarının ve bellek kullanımlarının da arttığı görülmektedir. Bellek kullanımı üst sınırlara yaklaşmadığı için Çöp Toplama mekanizması devreye girmemiştir.

Tablo 4.19: Asal Sayı Bulma Algoritması için Java Programlama Dili Performans Analizi.

Üst Sınır	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	*	10,58	0	11ms
1000	*	10,58	0	22ms
10000	*	13,28	0	64ms
100000	*	14,62	0	1s 747ms
1000000	13,1	29,39	0	2dk 4s 671ms

* Çok küçük değer olduğu için tespit edilememiştir.

Yukarıdaki tablo ve bir önceki tablo karşılaştırılacak olursa, Java programlama dili kullanıldığında CPU kullanımının daha düşük ancak bellek kullanımının daha yüksek seviyelerde olduğu görülmüştür.

Tablo 4.20: Asal Sayı Bulma Algoritması için Python Programlama Dili Performans Analizi.

Üst Sınır	Çalışma Süresi
100	1ms
1000	12ms
10000	998ms
100000	1dk 32s 348ms
1000000	136dk 3s 950ms

Bu tablolara bakıldığında, Asal Sayı Bulma Algoritması implementasyonları arasında, üst sınır büyüdükçe en yavaş çalışan programın Python programlama dili kullanılarak yazılan program olduğu görülmüştür.

4.2. ARAMA ALGORİTMALARININ UYGULAMALARININ PERFORMANS ANALİZİ

Bu bölümde literatürde en çok kullanılan arama algoritmaları arasında yer alan Doğrusal Arama Algoritması ve İkili Arama Algoritması incelenmiş ve performans analizi gerçekleştirilmiştir.

4.2.1. Doğrusal Arama Algoritması Uygulamasının Performans Analizi

Bu kısımda Doğrusal Arama Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Sırasıyla, Tablo 4.21'de C programlama dili, Tablo 4.22'de C# programlama dili, Tablo 4.23'te Java programlama dili ve Tablo 4.24'te Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.21: Doğrusal Arama Algoritması için C Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (ms) (Visual Studio)	Çalışma Süresi (ms) (Code::Blocks)
100	*	*	676	19
1000	*	*	717	23
10000	*	*	730	28
100000	*	*	889	32
1000000	*	*	1106	54

* Çok küçük değer olduğu için tespit edilememiştir.

Bu tabloda, CPU ve bellek kullanımlarının değerleri tespit edilemediği için yalnızca çalışma süreleri karşılaştırılmıştır. Sonuçlara bakıldığında, Code::Blocks'un Visual Studio'ya göre daha hızlı çalıştığı görülmektedir.

Tablo 4.22: Doğrusal Arama Algoritması için C# Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
100	13	*	0	7
1000	14	*	0	17
10000	14	12,6	0	19
100000	16	13,9	0	25
1000000	17	21,1	1	68

* Çok küçük değer olduğu için tespit edilememiştir.

Yukarıdaki analiz sonucunda, Çöp Toplama Mekanizması'nın devreye 1 kere girdiği görülmüştür. Eleman sayısının 1000000 olmadığı durumlarda bellek kullanım değerleri düşük seviyelerde olduğu için çöp toplama işlemi yapılmamıştır. Eleman sayısı arttıkça, CPU ve bellek kullanımının arttığı gözlemlenmiştir.

Tablo 4.23: Doğrusal Arama Algoritması için Java Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
100	*	10,58	0	3
1000	*	10,58	0	4
1000	*	10,58	0	9
10000	*	10,58	0	34
100000	*	11,42	0	263

* Çok küçük değer olduğu için tespit edilememiştir.

Doğrusal arama algoritmasının, Java programlama dili kullanılarak gerçekleştirilen analizi sonucunda veri sayısı 100, 1000, 10000 iken Java programlama dilinde C# programlama diline göre daha hızlı olduğu gözlemlenmiştir. Ancak 100000 ve 1000000 iken C# programlama dili daha kısa sürede sonuç vermiştir. Ancak bellek kullanımları karşılaştırılacak olursa, 100 ve 1000 eleman sayılı analizler haricinde C# programlama dili performans analizinin daha çok bellek tükettiği fark edilmiştir.

Tablo 4.24: Doğrusal Arama Algoritması için Python Programlama Dili Performans Analizi.

Eleman Sayısı	Çalışma Süresi
100	1ms
1000	1.4ms
10000	16ms
100000	186ms
1000000	1s 815ms

Doğrusal arama algoritması programı, Python dili ile çalıştırıldığında, küçük veri setlerinde diğer dillere göre daha hızlı çalışırken, büyük veri setlerinde daha yavaş çalışmaktadır.

4.2.2. İkili Arama Algoritması Uygulamasının Performans Analizi

Bu kısımda İkili Arama Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. İkili Arama Algoritması, yalnızca sıralı diziler üzerinde uygulanabilen bir algoritma olduğundan analizler yapılırken dizilerin hepsi önce Çabuk Sıralama Algoritması kullanılarak ile sıralanmış, daha sonra arama işlemi başlatılmıştır. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.25'te C programlama dili, Tablo 4.26'da C# programlama dili, Tablo 4.27'de Java programlama dili ve Tablo 4.28'de Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.25: İkili Arama Algoritması için C Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (ms) (Visual Studio)	Çalışma Süresi (ms) (Code::Blocks)
100	*	*	607	51
1000	*	*	607	62
10000	14	*	676	95
100000	16	7,6	804	610
1000000	18	14,5	2520	1422

* Çok küçük değer olduğu için tespit edilememiştir.

Bu bölümde gerçekleştirilen analizlerin sonuçları, C programlama dili kullanılarak gerçekleştirilen diğer analiz sonuçları ile benzer özelliktedir. Veri sayısı küçük iken, CPU ve bellek kullanımlarının ölçüm alınamayacak kadar küçük olduğu görülürken, veri sayısı büyüdükçe bu değerlerle birlikte çalışma süresinin de arttığı görülmüştür.

Tablo 4.26: İkili Arama Algoritması için C# Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
100	10	*	0	19
1000	12	12,9	0	21
1000	13	13,3	0	30
10000	14	13,6	0	50
100000	14	21,1	1	2232

* Çok küçük değer olduğu için tespit edilememiştir.

C# programlama dili kullanılarak gerçekleştirilen analiz sonucunda, eleman sayısı arttıkça, CPU kullanımı yüzdesi, bellek kullanımı ve çalışma süresinin de beklenildiği üzere arttığı görülmüştür.

Tablo 4.27: İkili Arama Algoritması için Java Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi (ms)
100	*	10,58	0	5
1000	*	10,58	0	6
10000	*	10,58	0	8
100000	*	10,58	0	23
1000000	*	21,81	0	255

* Çok küçük değer olduğu için tespit edilememiştir.

Java programlama dili kullanılarak gerçeklenen analiz sonucunda, CPU kullanım yüzdeleri çok küçük olduğu için ölçüm alınamamıştır. Çalışma süreleri karşılaştırılacak olursa, Java programlama dilinin diğer dillere göre daha hızlı olduğu görülmektedir.

Tablo 4.28: İkili Arama Algoritması için Python Programlama Dili Performans Analizi.

Eleman Sayısı	Çalışma Süresi
100	1ms
1000	7ms
10000	59ms
100000	1sn 18ms
1000000	45sn 647ms

Yukarıda da belirtildiği gibi, İkili Arama Algoritması analiz değerlerinin, Doğrusal Arama Algoritması analiz değerlerine göre yüksek çıkmasının sebebi dizinin önce sıralanıp, sonra arama işleminin başlatılmasıdır. Tablo incelendiğinde, bir kez daha görüleceği üzere, küçük veri setleri üzerinde Python programlama dili kullanılarak çalıştırılan programların daha kısa sürdüğü, büyük veri setleri üzerinde çalıştırılan programların daha uzun sürdüğü görülmektedir.

4.3. SIRALAMA ALGORİTMALARININ UYGULAMALARININ PERFORMANS ANALİZİ

Sıralama algoritmalarının performans analizleri sırasında sayı dizileri önce oluşturulmuş, daha sonra sıralı olmayan halleri ekrana bastırılmış, sıralama fonksiyonu çağrılmış ve sıralı halleri tekrar ekrana bastırılmıştır.

Bu bölümde literatürde en popüler sıralama algoritmalarından Eklemeli Sıralama Algoritması, Seçmeli Sıralama Algoritması, Kabarcık Sıralama Algoritması, Çabuk Sıralama Algoritması, Birleştirmeli Sıralama Algoritması, Gnome Sıralama Algoritması ve Yığın Sıralama Algoritması'nın performans analizleri gerçekleştirilmiştir.

4.3.1. Eklemeli Sıralama Algoritması Uygulamasının Performans Analizi

Bu kısımda Eklemeli Sıralama Algoritması'nın C, C#, Java ve Python programlama dillerinde kodlanan simülasyonları ile performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.29'da C programlama dili, Tablo 4.30'da C# programlama dili, Tablo 4.31'de Java programlama dili ve Tablo 4.32'de Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.29: Eklemeli Sıralama Algoritması için C Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
100	*	*	1s 804ms	1s 24ms
1000	2	0,5	2s 608ms	5s 537ms
10000	13	0,9	14s 303ms	9s 379ms
100000	13	1,6	43s 275ms	40s 247ms
1000000	14	8,3	16dk 29s 264ms	15dk 55s 445ms

* Çok düşük değer olduğu için tespit edilememiştir.

Tablonun ilk satırında CPU kullanımı ve bellek kullanımı değerlerinin çok küçük olması sebebiyle profil oluşturucusu 0 değerini döndürdüğü görülmüştür. CPU kullanımının 100000 ve 1000000 eleman için değişmediği gözlenmiştir. Ancak CPU kullanımı her iki eleman sayısı için %14 iken, maksimum bellek kullanımı yaklaşık 5 katına çıkmıştır. Bu artışın sebebi, dizinin boyutu arttığı için belleğe yazılacak veri sayısının da artmasıdır.

Tabloya genel olarak bakıldığında, eleman sayısı arttıkça bellek kullanımının da arttığı gözlemlenmiştir. Ayrıca eleman sayısı arttıkça Visual Studio'da ölçülen çalışma süresinin Code::Blocks'ta ölçülen çalışma süresine göre arttığı da gözlemlenmiştir.

Tablo 4.30: Eklemeli Sıralama Algoritması için C# Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	4	5,9	0	261 ms
1000	7	6,4	0	1s 403ms
10000	11	8,8	1	6s 947 ms
100000	13	9,6	17	34s 272 ms
1000000	15	17,0	176	19dk 6s 71ms

Tabloya bakıldığında, C ile karşılaştırılacak olursa, Çöp Toplama Mekanizması'na rağmen C#'ın çok daha fazla bellek kullandığı görülmektedir. Dizinin eleman sayısı arttıkça, CPU kullanımı ve bellek kullanımı da artmaktadır. Bellek kullanımı arttıkça da C# Çöp Toplayıcısı devreye girerek bellek kullanımını olabildiğince düşük değerlerde tutmaktadır.

Tablo 4.31: Eklemeli Sıralama Algoritması için Java Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	*	10,6	0	13ms
1000	*	10,6	0	23ms
10000	*	10,6	0	197ms
100000	9.9	29,79	0	2s 393ms
1000000	12.8	72,21	2	2dk 27s 243ms

* Çok küçük değer olduğu için tespit edilememiştir.

Görüldüğü üzere, Java programlama dilinin işlemci kullanma yüzdesi, C ve C# programlama dillerinin işlemci kullanma yüzdesine göre daha düşüktür. Ancak bellek tüketimleri ele alınacak olursa, tam tersi bir durum söz konusudur. Java programlama dilinin diğer dillere göre bir avantajı da çalışma süresidir.

Tablo 4.32: Eklemeli Sıralama Algoritması için Python Programlama Dili Performans Analizi.

Eleman Sayısı	Çalışma Süresi
100	20ms
1000	86ms
10000	6s 284ms
100000	11dk 42s 802ms
1000000	17 saat 10dk 23s 947ms*

* Program bu süre boyunca çalıştırılmış, ancak sonuç alınamadığı için durdurulmuştur.

Python programlama dili kullanılarak gerçekleştirilen performans analizinin sonucunda veri sayısı küçük iken diğer analiz sonuçlarına göre daha kısa sürelerde bittiği, ancak veri sayısı büyük iken de çok uzun sürelerde bitmediği gözlemlenmiştir.

4.3.2. Seçmeli Sıralama Algoritması Uygulamasının Performans Analizi

Bu kısımda Seçmeli Sıralama Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.33'te C programlama dili, Tablo 4.34'te C# programlama dili, Tablo 4.35'te Java programlama dili ve Tablo 4.36'da Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.33: Seçmeli Sıralama Algoritması için C Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
100	*	*	1s 163ms	775ms
1000	2	0,5	2s 397ms	1s 859ms
10000	13	0,9	12s 631ms	15s 198ms
100000	13	1,6	48s 81ms	49s 848ms
1000000	14	8,3	24dk 48s 641ms	23dk 47s

* Çok düşük değer olduğu için tespit edilememiştir.

Tablonun ilk satırında CPU kullanımı ve bellek kullanımı değerlerinin çok küçük olması sebebiyle Profil Oluşturucusu'nun 0 değerini döndürdüğü görülmüştür. Seçmeli Sıralama Algoritması'nın C programlama dilinde yapılan analizlerinde olduğu gibi, algoritmanın CPU kullanımı 10000 ve 100000 elemanlı dizide değişmezken bellek kullanımı artmıştır.

Tablo 4.34: Seçmeli Sıralama Algoritması için C# Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	4	5,9	0	141ms
1000	7	6,4	0	1s 399ms
10000	12	8,8	1	7s 991ms
100000	13	9,6	17	42s 693ms
1000000	16	16,8	181	28dk 23s 74ms

C# programlama dilinde gerçekleştirilen analizler sonucunda eleman sayısı arttıkça CPU kullanımının ve bellek kullanımının da arttığı gözlemlenmiştir.

Tablo 4.35: Seçmeli Sıralama Algoritması için Java Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	*	10,58	0	6ms
1000	*	10,58	0	10ms
10000	*	10,58	0	148ms
100000	12,6	30,73	0	5s 982ms
1000000	12,8	72,07	2	9dk 23s 955ms

* Çok düşük değer olduğu için tespit edilememiştir.

Java programlama dili kullanılarak gerçekleştirilen analizlerde Çöp Toplayıcısının C# programlama dili kullanılarak yapılan analizlerdeki kadar sıklıkla devreye girmediği görülmüştür. Bunun sonucunda ise bellek kullanımlarındaki büyük fark ortaya çıkmıştır. Ancak süre olarak C ve C# programlama dillerine göre çok daha kısa sürmüştür.

Tablo 4.36: Seçmeli Sıralama Algoritması için Python Programlama Dili Performans Analizi.

Eleman Sayısı	Çalışma Süresi
100	20ms
1000	86ms
10000	6s 284ms
100000	11dk 42s 802ms
1000000	17 saat 10dk 23s 947ms*

* Program bu süre boyunca çalıştırılmış, ancak sonuç alınamadığı için durdurulmuştur.

Python programlama dili kullanılarak gerçekleştirilen Seçmeli Sıralama Algoritması analizi sonucunda Eklemeli Sıralama Algoritması'nda olduğu gibi veri sayısı 1000000 iken sonuç verememiştir. Bunun dışında, diğer programlama dillerine göre küçük veri setleri üzerinde çok daha hızlı sonuç vermiştir.

4.3.3. Kabarcık Sıralama Algoritması Uygulamasının Performans Analizi

Bu kısımda Kabarcık Sıralama Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.37'de C programlama dili, Tablo 4.38'de C# programlama dili, Tablo 4.39'da Java programlama dili ve Tablo 4.40'ta Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.37: Kabarcık Sıralama Algoritması için C Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
100	*	*	956ms	880ms
1000	2	0,8	2s 767ms	6s 13ms
10000	13	0,9	10s 393ms	16s 425ms
100000	14	1,6	2dk 18s	1dk 49s 179ms
1000000	16	8,3	158dk 54s 263ms	84dk 2s 879ms

* Çok küçük değer olduğu için tespit edilememiştir.

Şu ana kadar analizi yapılan algoritmalara oranla 1000000 eleman sayısındaki süre artışı, algoritmanın eleman sayısı arttıkça verimliliğinin düştüğünü göstermektedir. Ancak eleman sayısı küçük iken de diğer algoritmalarla karşılaştırıldığında daha hızlı çalışmaktadır.

Tablo 4.38: Kabarcık Sıralama Algoritması için C# Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	4	5,9	0	183ms
1000	7	6,4	0	1s 381ms
10000	11	8,8	1	7s 136ms
100000	13	9,6	17	1dk 1s 487ms
1000000	14	16,3	176	73dk 32s 829ms

C# programlama dilinde gerçekleştirilen analizler sonucunda eleman sayısı arttıkça CPU kullanımının ve bellek kullanımının da arttığı gözlemlenmiştir.

Tablo 4.39: Kabarcık Sıralama Algoritması için Java Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	*	10,6	0	7ms
1000	*	10,6	0	39ms
10000	*	10,6	0	261ms
100000	12,5	30,73	0	19s 381ms
1000000	12,5	71,43	2	33dk 2s 965ms

* Çok küçük değer olduğu için tespit edilememiştir.

Tablonun ilk üç satırında CPU kullanımı ve bellek kullanımı değerlerinin değişmediği gözlemlenmektedir. Daha sonra eleman sayıları arttıkça bellek kullanımı ve CPU kullanımı değerlerinin ilk üç satıra oranla arttığı görülmektedir.

Tablo 4.40: Kabarcık Sıralama Algoritması için Python Programlama Dili Performans Analizi.

Eleman Sayısı	Çalışma Süresi
100	6ms
1000	126ms
10000	10s 598ms
100000	19dk 54s 396ms
1000000	12 saat 20dk 57s 710ms*

* Program bu süre boyunca çalıştırılmış, ancak sonuç alınamadığı için durdurulmuştur.

Python programlama dili kullanılarak gerçekleştirilen Seçmeli Sıralama Algoritması analizi sonucunda Eklemeli Sıralama Algoritması'nda olduğu gibi veri sayısı 1000000 iken sonuç verememiştir. Bunun dışında, diğer programlama dillerine göre küçük veri setleri üzerinde çok daha hızlı sonuç vermiştir.

4.3.4. Çabuk Sıralama Algoritması Uygulamasının Performans Analizi

Bu kısımda Çabuk Sıralama Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.41'de C programlama dili, Tablo 4.42'de C# programlama dili, Tablo 4.43'te Java programlama dili ve Tablo 4.44'te Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.41: Çabuk Sıralama Algoritması için C Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
100	*	*	1sn295ms	445ms
1000	2	0,78	3s 369ms	2s 648ms
10000	5	0,9	8s 424ms	15s 465ms
100000	9	1,6	21s 831ms	36s 162ms
1000000	14	9,3	2dk 40s 15ms	4dk 2s 879ms

* Çok küçük değer olduğu için tespit edilememiştir.

Tablonun ilk satırında CPU kullanımı ve bellek kullanımı değerlerinin çok küçük olması sebebiyle Profil Oluşturucusu'nun 0 değerini döndürdüğü görülmüştür. C# programlama dilinde gerçekleştirilen analiz sonucunda, 100 ve 1000 eleman sayılı diziler üzerindeki çalışma süresi diğer algoritmalardaki değerlere daha yakın olsa da, eleman sayısı arttıkça sürenin diğer algoritmalara oranla kısaldığı gözlemlenmiştir.

Tablo 4.42: Çabuk Sıralama Algoritması için C# Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	4	5,9	0	260ms
1000	4	6,4	0	1s 935ms
10000	4	8,8	1	7s 687ms
100000	5	9,8	17	31s 497ms
1000000	13	17,0	176	4dk 7s 12ms

Yukarıdaki tabloda C# programlama dili kullanılarak gerçekleştirilen analizlerin sonuçları bulunmaktadır. Eleman sayısı arttıkça, CPU kullanımı yüzdesi, bellek kullanımı, çöp toplama sayısı ve çalışma süresinin de arttığı gözlemlenmektedir.

Tablo 4.43: Çabuk Sıralama Algoritması için Java Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	*	10,6	0	16ms
1000	*	10,6	0	33ms
10000	*	10,6	0	117ms
100000	*	24,02	0	1s 200ms
1000000	3,4	72,57	2	11s 143ms

Java programlama dili kullanılarak gerçekleştirilen analizler sonucunda, CPU kullanımının diğer dillere oranla düştüğü gözlemlenmektedir. Ayrıca süreler karşılaştırıldığında ise Java programlama dilinin diğer dillere büyük bir üstünlüğü olduğunu söylenebilir. Java programlama dilinin dezavantajı ise bellek kullanımı değerlerinin diğer dillere göre yüksek olmasıdır.

Tablo 4.44: Çabuk Sıralama Algoritması için Python Programlama Dili Performans Analizi.

Eleman Sayısı	Çalışma Süresi
100	1ms
1000	20ms
10000	179ms
100000	Maximum Recursion Depth Exceeded hatası alındı – 2s 48ms
1000000	Maximum Recursion Depth Exceeded hatası alındı – 19s 398ms

Python progralama dilinde rekürsif fonksiyonlar çalıştırılırken, girdi sayısı 10^4'ü geçerse Maximum Recursion Depth Exceeded (Maksimum Özyineleme Derinliği Aşıldı) hatası alınmaktadır. Ancak "sys.setrecursionlimit()" fonksiyonu kullanılarak bu hatanın önüne

geçilebilmektedir. Bu fonksiyon sayesinde yeni özyineleme limiti 10^7'ye çıkarıldığı için 100000 ve 1000000 elemanlı sayı dizilerinde sonuç alınabilmiştir.

4.3.5. Birleştirmeli Sıralama Algoritması Uygulamasının Performans Analizi

Bu kısımda Birleştirmeli Sıralama Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.45'te C programlama dili, Tablo 4.46'da C# programlama dili, Tablo 4.47'de Java programlama dili ve Tablo 4.48'de Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.45: Birleştirmeli Sıralama Algoritması için C Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
100	*	*	687ms	501ms
1000	4	3,88	2s 935ms	2s 960ms
10000	13	288,8	12s 38ms	11s 890ms
100000	TAŞMA HATASI	TAŞMA HATASI	TAŞMA HATASI	30s 15ms
1000000	TAŞMA HATASI	TAŞMA HATASI	TAŞMA HATASI	TAŞMA HATASI

* Çok küçük değer olduğu için tespit edilememiştir.

C programlama dili çok büyük sayılar üzerinde çalışılmaya elverişli bir programlama dili değildir. Bunu sebebi, C dilindeki değişken türleri ile ifade edilebilecek sayı aralığının çok yüksek olmamasıdır. 100 ve 1000 elemanlı diziler üzerinde yapılan performans analizi sonuçlarına bakılacak olursa, bellek kullanımının daha önce hiç görülmemiş değerlere çıktığı fark edilir. Bu nedenle alınan taşma hataları normal karşılanabilir.

Tablo 4.46: Birleştirmeli Sıralama Algoritması için C# Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	6	5,9	0	371ms
1000	6	6,4	0	2s 143ms
10000	8	8,8	1	7s 263ms
100000	8	10,5	19	27s 698ms
1000000	13	24,2	205	3dk 55s 133ms

Yukarıdaki tablo ve C# programlama dili kullanılarak gerçekleştirilen diğer analizler incelendiğinde, bellek kullanımı ve çöp toplama sayılarının en yüksek değerlerine çıktığı gözlemlenmiştir.

Tablo 4.47: Birleştirmeli Sıralama Algoritması için Java Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	*	10,6	0	5ms
1000	*	10,6	0	20ms
10000	*	11,94	0	139ms
100000	*	11,94	0	716ms
1000000	5,2	71,57	2	14s 452ms

* Çok küçük değer olduğu için tespit edilememiştir.

C ve C# dili kullanılırken gerçekleştirilen analizlerde, bellek kullanımlarının diğer analizlere göre daha yüksek çıktığı gözlemlenmişti. Ancak Java programlama dili için özel bir durum söz konusu değildir.

Tablo 4.48: Birleştirmeli Sıralama Algoritması için Python Programlama Dili Performans Analizi.

Eleman Sayısı	Çalışma Süresi
100	1ms
1000	15ms
10000	191ms
100000	1s 782ms
1000000	19s 983ms

Diğer programlama dilleri kullanılarak gerçekleştirilen analizlerde olduğu gibi, Python programlama dili kullanılarak yapılan bu analiz sonucunda da Birleştirmeli Sıralama Algoritması'nın en hızlı sonuçlanan algoritmalardan biri olduğu doğrulanmıştır.

4.3.6. Gnome Sıralama Algoritması Uygulamasının Performans Analizi

Bu kısımda Gnome Sıralama Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.49'da C programlama dili, Tablo 4.50'de C# programlama dili, Tablo 4.51'de Java programlama dili ve Tablo 4.52'de Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.49: Gnome Sıralama Algoritması için C Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
100	*	*	1s 222ms	513ms
1000	4	0,78	3s 227ms	2s 561ms
10000	10	0,85	8s 778ms	12s 842ms
100000	13	1,5	41s 89ms	42s 594ms
1000000	14	8,3	20dk 35s 666ms	24dk 19s 32ms

* Çok küçük değer olduğu için tespit edilememiştir.

C programlama dili kullanılarak gerçekleştirilen analizler sonucunda, eleman sayısı küçük iken Code::Blocks geliştirme ortamında daha kısa sonuç alınırken, eleman sayısı büyüdükçe Visual Studio geliştirme ortamında daha kısa sonuçlar alındığı gözlemlenmiştir.

Tablo 4.50: Gnome Sıralama Algoritması için C# Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	5	5,9	0	5ms
1000	5	6,5	0	20ms
10000	13	8,8	1	139ms
100000	14	9,6	17	716ms
1000000	14	16,4	176	14s 452ms

Gnome Sıralama Algoritması için gerçekleştirilen analizler, C# programlama dili kullanılarak gerçekleştirilen diğer analizlerle benzer sonuçlar göstermektedir.

Tablo 4.51: Gnome Sıralama Algoritması için Java Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	*	10,6	0	6ms
1000	*	10,6	0	23ms
10000	*	13,28	0	275ms
100000	12,5	32,07	0	8s 555ms
1000000	12,6	62,89	2	11dk 57s 340ms

* Çok küçük değer olduğu için tespit edilememiştir.

Java programlama dili kullanılarak gerçekleştirilen analizlerin sonucu, Java programlama dili kullanılarak yapılan diğer analizlerde olduğu gibi, eleman sayısı arttıkça bellek kullanımında ve çalışma süresinde artış olduğu görülmüştür.

Tablo 4.52: Gnome Sıralama Algoritması için Python Programlama Dili Performans Analizi.

Eleman Sayısı	Çalışma Süresi
100	3ms
1000	169ms
10000	15s 693ms
100000	31dk 50s 447ms
1000000	12 saat 3dk 46s 92ms*

* Program bu süre boyunca çalıştırılmış, ancak sonuç alınamadığı için durdurulmuştur.

Python programlama dillerinde gerçekleştirilen analizlere bakıldığında, diğer programlama dillerindekinden farklı olarak, Gnome Sıralama Algoritması'nın diğer algoritmalara oranla çok daha yavaş çalıştığı görülmektedir.

4.3.7. Yığın Sıralama Algoritması Uygulamasının Performans Analizi

Bu kısımda Yığın Sıralama Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.53'te C programlama dili, Tablo 4.54'te C# programlama dili, Tablo 4.55'te Java programlama dili ve Tablo 4.56'da Python programlama dili için elde edilen sonuçlar gösterilmiştir.

Tablo 4.53: Yığın Sıralama Algoritması için C Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
100	*	*	913ms	450ms
1000	4	0,78	1s 558ms	2s 219ms
10000	6	0,85	6s 638ms	12s 449ms
100000	11	1,6	18s 962ms	31s 295ms
1000000	13	8,5	2dk 55s 29ms	3dk 36s 431ms

* Çok küçük değer olduğu için tespit edilememiştir.

Tablo 4.54: Yığın Sıralama Algoritması için C# Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	4	5,9	0	259ms
1000	4	6,4	0	1s 387ms
1000	4	8,6	1	13s 551ms
10000	8	9,6	17	42s 145ms
100000	14	17,0	176	4dk 50s 171ms

Yığın Sıralama algoritması için C programlama dili kullanılarak gerçekleştirilen analiz sonuçları, C programlama dili kullanılarak gerçekleştirilmiş diğer analizlerle benzer sonuç göstermektedir. Aynı şekilde C# programlama dili kullanılarak gerçekleştirilen performans analizi, C# programlama dili kullanılarak gerçekleştirilen diğer analizlerle benzer sonuçlar göstermektedir.

Tablo 4.55: Yığın Sıralama Algoritması için Java Programlama Dili Performans Analizi.

Eleman Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
100	*	10,6	0	6ms
1000	*	10,6	0	25ms
10000	*	11,94	0	158ms
100000	*	21,33	0	1s 446ms
1000000	5,1	67,6	2	13s 552ms

* Çok küçük değer olduğu için tespit edilememiştir.

Java programlama dili kullanılarak gerçekleştirilen diğer analizlerden farklı olarak CPU kullanımı değeri sadece 1000000 eleman sayılı dizi için ölçülebilmiştir.

Tablo 4.56: Yığın Sıralama Algoritması için Python Programlama Dili Performans Analizi.

Eleman Sayısı	Çalışma Süresi
100	0,3ms
1000	14ms
10000	188ms
100000	2s 340ms
1000000	25s 174ms

Yığın Sıralama Algoritması için Python programlama dili kullanılarak gerçekleştirilen analiz sonuçları, Çabuk Sıralama Algoritması ve Birleştirmeli Sıralama Algoritması için yapılan analizler gibi hızlı sonuç vermiştir. Bunun başlıca sebebi ise en kötü durum senaryolarının O(nlogn) olmasıdır. Bunun anlamı ise sıralama için en kötü senaryo bile olsa diğer algoritmalara (en kötü durum senaryosu $O(n^2)$ olan) göre daha az işlem yapılmasıdır.

4.4. POLİNOMSAL OLMAYAN ALGORİTMALARIN PERFORMANS ANALİZİ

Bu bölümde Gezgin Satıcı Problemi, Sırt Çantası Problemi ve Hanoi Kuleleri Problemi incelenmiş, bu problemlerin çözümü için birer algoritma geliştirilmiş ve bu algoritmaların performans analizi gerçekleştirilmiştir.

4.4.1. Gezgin Satıcı Problemi Çözüm Algoritması Uygulamasının Performans Analizi

Bu kısımda Gezgin Satıcı Problemi Çözüm Algoritması'nın C, C#, Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.57'de C programlama dili, Tablo 4.58'de C# programlama dili, Tablo 4.59'da Java programlama dili ve Tablo 4.60'ta Python programlama dili için elde edilen sonuçlar verilmiştir.

Tablo 4.57: Gezgin Satıcı Problemi Çözüm Algoritması için C Programlama Dili Performans Analizi.

Düğüm Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
5	*	*	68ms	46ms
10	3	*	981ms	716ms
15	16	0,85	176dk 48s 632ms	161dk 10s 524ms
20	20	0,86	5700dk.**	***

* Çok küçük değer olduğu için tespit edilememiştir.

** Program bu süre boyunca çalıştırılmış, ancak sonuç alınamadığı için durdurulmuştur.

*** Code::Blocks'ta Program bitmeden süre ölçülemediği için bu değer elde edilememiştir.

C programlama dili kullanılarak gerçekleştirilen performans analizi sonucunda, şehir sayısı 20 iken 5700 dakika geçtikten sonra bile sonuç alınamamıştır. Sonuç alınamamasının sebebi, şehir sayısı arttıkça uzaklık matrisinin elemanları da arttığı için ihtimal sayısının üssel olarak artması şeklinde ifade edilebilir.

Tablo 4.58: Gezgin Satıcı Problemi Çözüm Algoritması için C# Programlama Dili Performans Analizi.

Düğüm Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
5	5,77	5,7	0	4ms
10	6	5,9	0	61ms
15	18	6,1	0	4 saat 27dk 3s
20	21	12,3	0	68 saat 24dk 18s *

* Program bu süre boyunca çalıştırılmış, ancak sonuç alınamadığı için durdurulmuştur.

C# programlama dili kullanılarak yapılan analiz sonucunda, 20 şehirli Gezgin Satıcı Problemi Çözüm Algoritması için sonuç alınamamıştır. Burada, Çöp Toplama Mekanizması'nın devreye girmediği dikkat çekmektedir. Çünkü bellek kullanımları düşük seviyededir. Ancak Gezgin Satıcı Problemi Çözüm Algoritması için CPU kullanım yüzdelerinin özellikle 15 ve 20 şehirli çözümlerde yüksek olduğu görülmüştür.

Tablo 4.59: Gezgin Satıcı Problemi Çözüm Algoritması için Java Programlama Dili Performans Analizi.

Düğüm Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
5	*	10,6	0	3ms
10	0,3	11,94	0	1s
15	12,6	15,97	0	46dk 46s
20	13,5	70,51	3	104 saat 51dk 25s **

* Çok küçük değer olduğu için tespit edilememiştir.

** Program bu süre boyunca çalıştırılmış, ancak sonuç alınamadığı için durdurulmuştur.

Java programlama dilinde gerçekleştirilen performans analizi sonucunda, C ve C# programlama dilleri kullanılarak gerçekleştirilen performans analizlerine göre daha kısa sürelerde programın çalışıp, çözüm ürettiği görülmektedir. Ancak bellek kullanımları karşılaştırılacak olursa, Java programlama dilinin bellekte çok daha fazla yer tükettiği gözlemlenmektedir. Çöp Toplayıcısının 3 kere devreye girmesine rağmen 70 MB'a yakın bellek kullanılmıştır.

Tablo 4.60: Gezgin Satıcı Problemi Çözüm Algoritması için Python Programlama Dili Performans Analizi.

Düğüm Sayısı	Çalışma Süresi
5	1ms
10	700ms
15	93 saat 6dk*
20	-

* Program bu süre boyunca çalıştırılmış, ancak sonuç alınamadığı için durdurulmuştur.

Analiz esnasında 15 şehirli Gezgin Satıcı Problemi Çözüm Algoritması, 93 saate yakın bir süre çalıştırılmasına rağmen sonuç alınamamıştır. Bu nedenle 20 şehirli Gezgin Satıcı Problemi Çözüm Algoritması çalıştırılmamıştır. Bu problem üzerinde Python programlama dilinin C, C# ve Java programlama dillerine göre çok daha yavaş olduğu görülmüştür.

4.4.2. Sırt Çantası Problemi Çözüm Algoritması Uygulamasının Performans Analizi

Bu kısımda Sırt Çantası Problemi Çözüm Algoritması'nın C, C# Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda paylaşılmıştır. Sırasıyla, Tablo 4.61'de C programlama dili, Tablo 4.62'de C# programlama dili, Tablo 4.63'te Java programlama dili ve Tablo 4.64'te Python programlama dili için elde edilen sonuçlar verilmiştir.

Tablo 4.61: Sırt Çantası Problemi Çözüm Algoritması için C Programlama Dili Performans Analizi.

Veri Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
10	13	*	601ms	51ms
25	15	6,6	709ms	105ms
50	16	6,7	1s 222ms	1s 39ms
75	17	6,7	9s 331 ms	2s 716ms
100	19	7,0	8dk 6s 12ms	1dk 49s 526ms

* Çok küçük değer olduğu için tespit edilememiştir.

C programlama dili kullanılarak gerçekleştirilen analiz sonucunda, problemin karmaşıklığı arttıkça, CPU Kullanımı, bellek kullanımı ve çalışma sürelerinin arttığı görülmektedir. Code::Blocks platformunda çalıştırılan programın, Visual Studio platformunda çalıştırılan programa göre daha hızlı çalıştığı görülmüştür.

Tablo 4.62: Sırt Çantası Problemi Çözüm Algoritması için C# Programlama Dili Performans Analizi.

Veri Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
10	13	12,7	0	799ms
25	15	12,9	0	815ms
50	15	13,3	0	1s 14ms
75	19	13,3	0	4s 137ms
100	19	13,5	0	3dk 11s 75ms

C# programlama dili kullanılarak Sırt Çantası Problemi Çözüm Algoritması için gerçekleştirilen performans analizi, C# programlama dili kullanılarak gerçekleştirilen diğer analizlerle benzer sonuçlar göstermektedir.

Tablo 4.63: Sırt Çantası Problemi Çözüm Algoritması için Java Programlama Dili Performans Analizi.

Veri Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
10	*	11,94	0	8ms
25	*	13,28	0	11ms
50	*	14,42	0	72ms
75	*	14,77	0	855ms
100	12,6	14,77	0	34s 756ms

* Çok küçük değer olduğu için tespit edilememiştir.

Java programlama dili kullanılarak gerçekleştirilen bu analiz sonucunda, veri sayısı arttıkça bellek kullanımlarının da arttığı görülmektedir. CPU kullanımı yüzdesi için sadece 100

elemanlı veri setinden sonuç alınabilmiştir. Bellek kullanımı değerleri de düşük seviyede olduğu için Çöp Toplama Mekanizması devreye girmemiştir.

Tablo 4.64: Sırt Çantası Problemi Çözüm Algoritması için Python Programlama Dili Performans Analizi.

Veri Sayısı	Çalışma Süresi
10	1ms
25	76ms
50	8s 419ms
75	2dk 4s 581ms
100	122dk 16s 58ms

Python programlama dili kullanılarak gerçekleştirilen bu analiz sonucunda, bu programlama dilinde yazılan programların diğer programlama dillerinde yazılan programlara oranla daha uzun sürede tamamlandığı görülmüştür.

4.4.3. Hanoi Kuleleri Problemi Çözüm Algoritması Uygulamasının Performans Analizi

Bu kısımda Hanoi Kuleleri Problemi Çözüm Algoritması'nın C, C# Java ve Python programlama dillerinde performans analizleri gerçekleştirilmiştir. Elde edilen sonuçlar aşağıdaki tablolarda ifade edilmiştir. Sırasıyla, Tablo 4.65'te C programlama dili, Tablo 4.66'da C# programlama dili, Tablo 4.67'de Java programlama dili ve Tablo 4.68'de Python programlama dili için elde edilen sonuçlar verilmiştir.

Tablo 4.65: Hanoi Kuleleri Problemi Çözüm Algoritması için C Programlama Dili Performans Analizi.

Disk Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çalışma Süresi (Visual Studio)	Çalışma Süresi (Code::Blocks)
5	*	6,3	613ms	58ms
10	13	6,7	1s 214ms	2s 821ms
15	13	7,0	4s 635ms	5s 694ms
20	13	7,0	1dk 9s 310ms	1dk 20s 566ms

* Çok küçük değer olduğu için tespit edilememiştir.

Hanoi Kuleleri Probleminin Çözüm Algoritması için C programlama dili kullanılarak yapılan bu analiz sonucunda, disk sayısı arttıkça, Visual Studio platformunun Code::Blocks platformuna göre daha kısa sürede sonuç vermeye başladığı görülmektedir.

Tablo 4.66: Hanoi Kuleleri Problemi Çözüm Algoritması için C# Programlama Dili Performans Analizi.

Disk Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
5	10	12,8	0	810ms
10	12	13,1	0	1s 213ms
15	14	16,5	1	6s 473ms
20	17	16,8	57	1dk 8s 179ms

Yukarıdaki tablo incelendiğinde, Çöp Toplama Mekanizması'nın, bellek kullanımını alt seviyelerde tutmak için bir çok kez devreye girdiği görülmektedir.

Tablo 4.67: Hanoi Kuleleri Problemi Çözüm Algoritması için Java Programlama Dili Performans Analizi.

Disk Sayısı	CPU Kullanımı %	Maksimum Bellek Kullanımı (MB)	Çöp Toplama Sayısı	Çalışma Süresi
5	*	*	0	14ms
10	*	*	0	59ms
15	3,3	19,99	0	2s 369ms
20	6,5	69,98	2	56s 378ms

* Çok küçük değer olduğu için tespit edilememiştir.

Yukarıdaki iki tablo karşılaştırıldığında, bellek kullanımı değerlerinde Java programlama dilinin disk sayısı arttıkça, C# programlama diline göre daha fazla bellek kullandığı görülür. Ancak CPU kullanım yüzdeleri ve çalışma süreleri karşılaştırıldığında Java programlama dilinin, C# programlama diline göre daha üstün olduğu gözlemlenmektedir.

Tablo 4.68: Hanoi Kuleleri Problemi Çözüm Algoritması için Python Programlama Dili Performans Analizi.

Disk Sayısı	Çalışma Süresi
5	1ms
10	12ms
15	356ms
20	7s 603ms

Python programlama dili ile gerçeklenen bu analiz sonucunda, bir kez daha Python programlama dilinin küçük veriler üzerinde diğer üç programlama diline göre daha hızlı sonuç verdiği ortaya çıkmıştır.

5. TARTIŞMA VE SONUÇ

Bu tez çalışmasında, literatürde sıklıkla kullanılan çeşitli matematiksel algoritmaların farklı programlama dilleri ve farklı yazılım geliştirme platformları kullanılarak yapılan implementasyonları üzerinde performans analizleri gerçekleştirilmiş ve çeşitli kriterlere göre karşılaştırmalar yapılmıştır. Bu karşılaştırma işlemleri C, C#, Java ve Python programlama dilleri kullanılarak gerçekleştirilmiştir. Uygun bir karşılaştırma yapılabilmesi için uygulamalar, programlama dillerinde ek bir kütüphane kullanılmadan veya farklı ek bir iş yapılmadan, gelen istekleri karşılayarak, basit cevaplar dönecek şekilde tasarlanmıştır. Bu basit cevaplar alınırken sırasıyla CPU kullanımı yüzdesi, maksimum bellek kullanımı miktarı, çöp toplama sayısı ve çalışma süresi ölçülmüştür. Yazılan programlar, Visual Studio, Code::Blocks, Net Beans ve JetBrains PyCharm platformları üzerinde geliştirilmiştir. Ayrıca algoritmaların bir kısmı iteratif ve rekürsif olmak üzere iki ayrı yaklaşım kullanılarak modellenmiş ve bu sayede algoritma tasarım yaklaşımlarının etkisi incelenmeye çalışılmıştır.

Bu tez çalışmasında, ilk olarak genel matematiksel algoritmalardan; faktöriyel hesaplama, üs alma ve asal sayı bulma işlemlerini gerçekleştiren algoritmalar ele alınmış ve bu algoritmaların performans analizleri gerçekleştirilmiştir. Bu analizler gerçekleştirilirken faktöriyel hesaplama ve üs alma işlemlerini gerçekleştiren algoritmaların hem iteratif hem rekürsif versiyonları yazılmıştır ve bu versiyonlar kendi aralarında da karşılaştırılmıştır. İkinci olarak bilgisayar biliminin en önemli problemlerinden birine çözüm getiren arama algoritmaları incelenmiştir. Literatürde en çok kullanılan arama algoritmalarından, Doğrusal Arama Algoritması ve İkili Arama Algoritması, bu kısımda performans analizleri gerçekleştirilen algoritmalardır. Üçüncü olarak, tıpkı arama algoritmaları gibi bilgisayar biliminin, önemli sorunlarına çözüm üreten sıralama algoritmaları incelenmiştir. Eklemeli Sıralama Algoritması, Seçmeli Sıralama Algoritması, Kabarcık Sıralama Algoritması, Çabuk Sıralama Algoritması, Birleştirmeli Sıralama Algoritması, Gnome Sıralama Algoritması ve Yığın Sıralama Algoritması olmak üzere yedi farklı sıralama algoritmasının performans analizleri gerçekleştirilmiştir. Bu tez çalışmasında yapılan dördüncü ve son işlem ise polinomsal olmayan problemlerin çözüm algoritmalarının karşılaştırılmasıdır. Bu kısımda ele alınan algoritmalar da Gezgin Satıcı Problemi Çözüm Algoritması, Sırt Çantası Problemi Çözüm Algoritması ve Hanoi Kuleleri Problemi Çözüm Algoritması'dır.

Bu tez çalışmasında yapılan performans analizleri neticesinde aşağıdaki sonuçların elde edildiği görülmüştür:

- Genel matematiksel algoritmalar için,
  - C programlama dilinin, en az bellek kullanan programlama dili olduğu,
  - C ve Java'nın en az CPU kullanan programlama dilleri olduğu,
  - çalışma süreleri açısından Java'nın genel olarak verimlilik gösterdiği,
  - çalışma süreleri açısından Python'un dengesiz bir davranış sergilediği; küçük veri sayısı üzerinde oldukça hızlı iken, veri sayısı büyüdükçe üssel bir yavaşlama gösterdiği,
  - C programlama dilinde, Code::Blocks platformunun, Visual Studio platformuna göre genellikle daha hızlı sonuç verdiği,
  - özellikle Asal Sayı Bulma Algoritması için, veri sayısı arttıkça sürenin de arttığı,
- Arama Algoritmaları için,
  - C programlama dili için, Code::Blocks platformunun Visual Studio platformundan daha hızlı bir yapı gösterdiği,
  - C programlama dilinin en az bellek tüketen programlama dili olduğu,
  - C# programlama dilinde (Visual Studio) çöp toplama mekanizmasının devreye girdiği,
- Sıralama Algoritmaları için,
  - Java programlama dilinin, çoğu zaman sonuca en çabuk ulaşan programlama dili olduğu,
  - beklenildiği gibi, zaman karmaşıklığı O(nlog(n)) olan arama algoritmalarının, zaman karmaşıklığı $O(n^2)$ olan algoritmalara göre çok daha hızlı sonuç verdiği,
  - C# programlama dilinde, bellek kullanımını azaltmak için çöp toplama mekanizmasının sık sık devreye girdiği,
  - veri sayısı arttıkça, CPU kullanımının, bellek kullanımının ve çalışma süresinin de arttığı,
  - C ve Java'nın en az CPU kullanan programlama dilleri olduğu,
  - küçük miktardaki veri sayısı üzerinde, C programlama dilinin en az bellek kullanan programlama dili olduğu,
  - Birleştirmeli Sıralama Algoritması için işlem sayısı arttıkça çöp toplama sayısının da arttığı,

- Polinomsal Olmayan Problemlerin Çözüm Algoritmaları için,
    - veri sayısı küçük iken, programlardan çok hızlı sonuç alındığı,
    - veri sayısı arttıkça, çalışma sürelerinin üstel olarak arttığı,
    - Java programlama dilinin en çok bellek tüketen programlama dili olduğu,
    - veri sayısı küçük iken, Code::Blocks platformunun, büyük iken Visual Studio platformunun daha hızlı çalıştığı,
    - Python programlama dilinin, veri sayısı küçük iken daha hızlı sonuç verdiği, ancak veri sayısı büyüdükçe, diğer dillere göre çok daha yavaş sonuç verdiği

görülmüştür.

Bu tez çalışması, yukarıda ifade edilen sonuçlar sayesinde, çeşitli programlama dillerinin ve platformlarının seçimi ile ilgili yazılım geliştiricilere bir rehber niteliği taşımayı hedeflemiştir. Sonuç olarak herhangi bir algoritma gerçeklenirken hiçbir programlama dilinin her kriterde en başarılı ya da en başarısız olmadığı, yazılım geliştiricilerin kısıt ve önceliklerine göre bir programlama dili seçmesi gerektiği, bununla ilgili olarak da yazılım geliştiricilerin kısıt ve önceliklerini başarılı bir şekilde tespit etmelerinin yazılım geliştirme açısından büyük önem arz ettiği tespit edilmiştir.

Bu çalışma, algoritma sayısı arttırılarak, karşılaştırma ölçütleri arttırılarak ya da farklı işletim sistemleri işin içine katılarak geliştirilebilir. İleride yazılım metriklerinin ve Mac ve Linux işletim sistemlerinin de bu çalışmaya dahil edilmesi planlanmaktadır.

KAYNAKLAR

[1]. Çiğdem U., 2011, *Np Problemlerde Ve Sistem Tasarımında DNA Hesaplamanın Kullanılması*, Yüksek Lisans Tezi, Fırat Üniversitesi Fen Bilimleri Enstitüsü

[2]. Agatz N., Bouman P., Schmidt M., 2018, Optimization approaches for the traveling salesman problem with drone, *Transportation Science,* 52(4), 965-981.

[3]. Anderson R., Ashlagi I., Gamarnik D., Roth A.E., 2015, Finding long chains in kidney exchange using the traveling salesman problem, *Proceedings of the National Academy of Sciences* 112(3), 663-668.

[4]. Ezugwu, A. E. S., Adewumi, A. O., Frîncu, M. E., 2017, Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem, *Expert Systems with Applications*, 77, 189-210.

[5]. Malaguti, E., Martello, S., Santini, A., 2018, The traveling salesman problem with pickups, deliveries, and draft limits, *Omega*, 74, 50-58.

[6]. Lin, S., 1965, Computer solutions of the traveling salesman problem, *Bell System Technical Journal,* 44(10),2245-2269.

[7]. Dorigo, M., Gambardella, L. M., 1997, Ant colonies for the travelling salesman problem, B*iosystems*, 43(2), 73-81.

[8]. Wang, K. P., Huang, L., Zhou, C. G., Pang, W., 2003, Particle swarm optimization for traveling salesman problem, *Proceedings of the international conference on machine learning and cybernetics,* 1583-1585.

[9]. Nguyen, H. D., Yoshihara, I., Yamamori, K., Yasunaga, M., 2007, Implementation of an effective hybrid GA for large-scale traveling salesman problems, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1), 92-99.

[10]. Baraglia, R., Hidalgo, J. I., Perego, R., 2001, A hybrid heuristic for the traveling salesman problem, *IEEE Transactions on evolutionary computation*, 5(6), 613-622.

[11]. Thanh, P. D., Binh, H. T. T., Lam, B. T., 2013, A survey on hybridizing genetic algorithm with dynamic programming for solving the traveling salesman problem, *2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR)*, 66-71.

[12]. Han, A., 2006, DNA Computing model for the 0/1 knapsack problem, *Proceedings of th*e *Sixth International Conference on Hybrid Intelligent Systems (HIS'06)*, 18.

[13]. Abdel-Basset, M., El-Shahat, D., Sangaiah, A. K., 2019, A modified nature inspired meta-heuristic whale optimization algorithm for solving 0–1 knapsack problem, *International Journal of Machine Learning and Cybernetics*, 10(3), 495-514.

[14]. Hosch W.L., 2019, *Tower of Hanoi | puzzle*, https://www.britannica.com/topic/Tower-of-Hanoi#ref1083058 [Ziyaret Tarihi: 27 Kasım 2019].

[15]. Şeker, Ş. E., 2019, *Hanoi Kuleleri (Towers of Hanoi)* http://bilgisayarkavramlari.sadievrenseker.com/2012/03/15/hanoi-kuleleri-towers-of-hanoi/,[Ziyaret Tarihi: 27 Kasım 2019].

[16]. Medvidovic, N., Taylor, R. N. 2000, A classification and comparison framework for software architecture description languages. *IEEE Transactions on software engineering*, 26(1), 70-93.

[17]. Prechelt, L., 2000, An empirical comparison of seven programming languages. *Computer,* 33(10), 23-29.

[18]. Şahin, M., 2007, Java, Python Ve Ruby Dillerinin Performans Karşılaştırması. *Akademik Bilişim Konferansı Bildiri Kitapçığı,* 529-532.

[19]. Aruoba, S. B., Fernández-Villaverde, J. 2014, A Comparison Of Programming Languages In Economics. *National Bureau of Economic Research*, 39(3), 245-273.

[20]. Kemer, E., Samli, R., 2019, Performance comparison of scalable rest application programming interfaces in different platforms. *Computer Standards and Interfaces,* 66, 103355.

[21]. Loidl, H. W., Rubio, F., Scaife, N., Hammond, K., Horiguchi, S., Klusik, U., Loogen R., Michaelson, G. J., Pena, R., Priebe, S., Trinder, P. W., Rebón, Á. J., 2003, Comparing parallel functional languages: Programming and performance. *Higher-Order and Symbolic Computation,* 16(3), 203-251.

[22]. Holzschuher, F., Peinl, R., 2013, Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j. *Proceedings of the Joint EDBT/ICDT Workshops,* 195-204.

[23]. Fangohr, H., 2004, A comparison of C, MATLAB, and Python as teaching languages in engineering, *Proceedings of International Conference on Computational Scienc*e, 1210-1217.

[24]. Horn, M. S., Solovey, E. T., Crouser, R. J., Jacob, R. J., 2009, Comparing the use of tangible and graphical programming languages for informal science education. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* 975-984

[25]. Fourment, M., Gillings, M. R. 2008, A comparison of common programming languages used in bioinformatics. *BMC bioinformatics*, 9(1), 82.

[26]. Liman E., 2019*, C Dili Tarihçesi,* https://www.erkanliman.com/programlama/c/c-dili-tarihcesi.html, [Ziyaret Tarihi: 27 Kasım 2019].

[27]. Keleştemur A., 2019, *C# Dilinin Tarihçesi,* https://gelecegiyazanlar.turkcell.com.tr/konu/c-sharp/egitim/c-101/c-dilinin-tarihcesi, [Ziyaret Tarihi: 27 Kasım 2019].

[28]. Anonim, 2019, *Java Programlama Dilinin Tarihi ve Gelişimi - Bilgi Kulvarı,* https://www.bilgikulvari.com/java-programlama-dilinin-tarihi-ve-gelisimi/, [Ziyaret Tarihi: 27 Kasım 2019].

[29]. Anonim, 2019, *Python Programlama Diline Genel Bakış | BackLink.Name - İnternet ve Teknoloji Blog,* https://backlink.name/blog/python-programlama-diline-genel-bakis, [Ziyaret Tarihi: 27 Kasım 2019].

[30]. Anonim, 2019, *Visual Studio Nedir? Visual Studio Kurulumu - Resimli Anlatım | Mehsatek.* http://www.mehsatek.com/visual-studio-nedir-visual-studio-kurulumu/, [Ziyaret Tarihi: 27 Kasım 2019].

[31]. Anonim, 2019, *FAQ-General - CodeBlocks.* http://wiki.codeblocks.org/index.php/FAQ-General#Q:_What_is_Code::Blocks.3F, [Ziyaret Tarihi: 27 Kasım 2019].

[32]. Anonim, 2019, *NetBeans ve www.netbeans.org'a hoşgeldiniz,* https://netbeans.org/index_tr.html, [Ziyaret Tarihi: 27 Kasım 2019].

[33]. Anonim, 2019, *PyCharm: the Python IDE for Professional Developers by JetBrains,* https://www.jetbrains.com/pycharm/, [Ziyaret Tarihi: 27 Kasım 2019].

ÖZGEÇMİŞ

Kişisel Bilgiler	
Adı Soyadı	Uğur Osman YÜCEL
Doğum Yeri	Göztepe/İSTANBUL
Doğum Tarihi	28.05.1994
Uyruğu	☑ T.C. ❒ Diğer:
Telefon	
E-Posta Adresi	uyucel94@gmail.com
Web Adresi	



Eğitim Bilgileri	
Lisans	
Üniversite	Bahçeşehir Üniversitesi
Fakülte	Mühendislik ve Doğa Bilimleri Fakültesi
Bölümü	Yazılım Mühendisliği (EN)
Mezuniyet Yılı	2017

Yüksek Lisans	
Üniversite	İstanbul Üniversitesi-Cerrahpaşa
Enstitü Adı	Lisansüstü Eğitim Enstitüsü
Anabilim Dalı	Bilgisayar Mühendisliği Anabilim Dalı
Programı	Bilgisayar Mühendisliği Programı

Makale ve Bildiriler

1. Aydın Güven Z.B., Yücel U.O., Şamlı R., 2019, Measurement In Software Engineering, The Importance Of Software Metrics, In: Zeynep A. (ed), IGI Global, İstanbul, 166-182.
2. Yücel U.O., Güven Aydın Z.B., Şamlı R., 2019, A Survey About Object Oriented Software Metrics. International Conference On Engineering and Natural Science, Prag, Çekya 12-16 Temmuz, 232-234.
3. Güven Aydın Z.B., Yücel U.O., Şamlı R., 2019, An Overview About Software Metric Tools. International Conference on Engineering and Natural Science, Prag, Çekya, 12-16 Temmuz, 225-231.
4. Mumcu M.C., Yücel U.O., 2018, TEA Encyription Algorithms Design and Applications. 5. Uluslararası Multidisipliner Çalışma Kongresi, Antalya, Türkiye, 2-3 Kasım, 79.