

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**204 DATA STRUCTURES (3+1)**

**2020–2021 FALL SEMESTER**

**PROJECT-3 REPORT**

**(Search Tree, Heap, Hash Table, Bike Rental System)**

**DELIVERY DATE**

31/01/2021

**PREPARED BY**

05180000091,Kutay Avcı

İçindekiler

[1.a Durak nesnelerini oluşturma ve ağaca ekleme 3](#_Toc61463807)

[1.b Ağaç derinliği ve ağaçtaki bilgilerin ekrana listelenmesi 3](#_Toc61463808)

[1.b.1 Kaynak Kod 8](#_Toc61463809)

[1.b.2 Ekran görüntüleri 8](#_Toc61463810)

[1.c Verilen müşteri ID’si için bilgi listeleme 10](#_Toc61463811)

[1.c.1 Kaynak Kod 10](#_Toc61463812)

[1.c.2 Ekran görüntüleri 11](#_Toc61463813)

[1.d Kiralama İşlemi 12](#_Toc61463814)

[2.a Hash Tablosuna Ekleme 12](#_Toc61463815)

[2.b Hash Tablosu Güncelleme 12](#_Toc61463816)

[3.a Heap Veri Yapısı Tasarlama 12](#_Toc61463817)

[3.a.1 Ön Çalışma 13](#_Toc61463818)

[3.a.2 Kaynak Kod 17](#_Toc61463819)

[3.b Max Heap düğüm yerleştirme 17](#_Toc61463820)

[3.c Heap bilgi çekme 20](#_Toc61463821)

[3.c.1 Kaynak Kod 20](#_Toc61463822)

[3.c.2 Ekran görüntüleri 21](#_Toc61463823)

[4.a Simple sorting algoritması 22](#_Toc61463824)

[4.b Advanced sorting algoritması 24](#_Toc61463825)

[4.c Sıralama algoritmalarının karşılaştırılması 24](#_Toc61463826)

[4.d Görselleştirme araçları 25](#_Toc61463827)

[Özdeğerlendirme Tablosu 26](#_Toc61463828)

ARAMA AĞAÇLARI, YIĞINLAR VE HASH TABLOSU: BİSİKLET KİRALAMA SİSTEMİ

// Visual Studio, 2019 16.8.0, C#

## 1.a Durak nesnelerini oluşturma ve ağaca ekleme

## public class Durak

## {

## private string DurakAdı;

## private int BosPark;

## private int TandemBisiklet;

## private int NormalBisiklet;

## public List<Musteri> musteri;

## public Durak(String DurakAdı, int BosPark, int TandemBisiklet, int NormalBisiklet) //Constructor method

## {

## this.DurakAdı = DurakAdı;

## this.BosPark = BosPark;

## this.TandemBisiklet = TandemBisiklet;

## this.NormalBisiklet = NormalBisiklet;

## musteri = new List<Musteri>() ;

## }

## //Getter and Setter methods:

## public String getDurakAdı()

## {

## return DurakAdı;

## }

## public void setDurakAdı(String DurakAdı)

## {

## this.DurakAdı = DurakAdı;

## }

## public int getBosPark()

## {

## return BosPark;

## }

## public void setBosPark(int BosPark)

## {

## this.BosPark = BosPark;

## }

## public int getTandemBisiklet()

## {

## return TandemBisiklet;

## }

## public void setTandemBisiklet(int TandemBisiklet)

## {

## this.TandemBisiklet = TandemBisiklet;

## }

## public int getNormalBisiklet()

## {

## return NormalBisiklet;

## }

## public void setNormalBisiklet(int NormalBisiklet)

## {

## this.NormalBisiklet = NormalBisiklet;

## }

## public string getMusteri() //gets every element of array musteri.

## {

## string s = "";

## foreach (var m in musteri)

## s += m.ToString();

## return s;

## }

## public string ToStringHeap()

## {

## return "Durak Adı: " + DurakAdı + "\tBoş Park Sayısı: " + BosPark + "\tTandem Bisiklet Sayısı: " + TandemBisiklet + "\tNormal Bisiklet Sayısı: " + NormalBisiklet;

## }

## public override string ToString()

## {

## return "Durak Adı: " + DurakAdı + "\tBoş Park Sayısı: " + BosPark + "\tTandem Bisiklet Sayısı: " + TandemBisiklet + "\tNormal Bisiklet Sayısı: " + NormalBisiklet + "\n" + getMusteri();

## }

Durak Ağacı Sınıfı:

public void insert(Durak newdata)//insert a new Durak into tree

{

TreeNode newNode = new TreeNode();

if (newdata.getBosPark() <= newdata.getNormalBisiklet() + newdata.getTandemBisiklet())//if BosPark is lower than sum of NormalBisiklet and TandemBisiklet then increase BosPark by length of musteri

newdata.setBosPark(newdata.getBosPark() + musteriSayısı);

else // in the other case set BosPark to sum of NormalBisiklet and TandemBisiklet

newdata.setBosPark(newdata.getNormalBisiklet() + newdata.getTandemBisiklet());

for (int i = 0; i < musteriSayısı; i++)

{

if (newdata.getNormalBisiklet() != 0)

newdata.setNormalBisiklet(newdata.getNormalBisiklet() - 1);

else

if (newdata.getTandemBisiklet() == 0)

{

Console.WriteLine(newdata.getDurakAdı() + " Durağında Bisiklet kalmadı.");

break;

}

else

newdata.setTandemBisiklet(newdata.getTandemBisiklet() - 1);

}

newNode.data = newdata;

MusteriEkle(newNode);

if (root == null)

{

root = newNode;

}

else//insert element into in rules of binary search tree structure.

{

TreeNode current = root;

TreeNode parent;

while (true)

{

parent = current;

if (string.Compare(newdata.getDurakAdı(), current.data.getDurakAdı()) == -1)

{

current = current.leftChild;

if (current == null)

{

parent.leftChild = newNode;

return;

}

}

else

{

current = current.rightChild;

if (current == null)

{

parent.rightChild = newNode;

return;

}

}

}

}// end else not root

}// end insert()

public int musteriSayısı = 0;

public int toplamMusteri = 0;

Random random = new Random();

public void MusteriEkle(TreeNode newNode)//inserts new Musteri into the musteri list.

{

musteriSayısı = random.Next(1, 11);

toplamMusteri += musteriSayısı;

for (int i = 0; i < musteriSayısı; i++)//adds random number of musteri into list.

{

int ID = random.Next(1, 21);//creates random ID.

string saat = random.Next(0, 24) + ":" + random.Next(0, 60);//creates random time.

newNode.data.musteri.Add(new Musteri(ID, saat));

}

}

## }

## 1.b Ağaç derinliği ve ağaçtaki bilgilerin ekrana listelenmesi

### 1.b.1 Kaynak Kod

### class TreeNode

### {

### public Durak data;

### public TreeNode leftChild;

### public TreeNode rightChild;

### public void displayNode()

### {

### Console.WriteLine(data);

### }

### public int Height()

### {

### //return 1 when leaf node is found

### if (this.leftChild== null && this.rightChild == null)

### {

### return 1; //found a leaf node

### }

### int left = 0;

### int right = 0;

### //recursively go through each branch

### if (this.leftChild != null)

### left = this.leftChild.Height();

### if (this.rightChild != null)

### right = this.rightChild.Height();

### //return the greater height of the branch

### if (left > right)

### {

### return (left + 1);

### }

### else

### {

### return (right + 1);

### }

### }

### }

DurakAğacı Sınıfı:

public void inOrder(TreeNode localRoot) //prints tree inOrder.

{

if (localRoot != null)

{

inOrder(localRoot.leftChild);

localRoot.displayNode();

inOrder(localRoot.rightChild);

}

}

public int Height()//gives the height of tree.

{

//if root is null then height is zero

if (root == null)

{ return 0; }

return root.Height();

}

### 1.b.2 Ekran görüntüleri

## 

## 

## 1.c Verilen müşteri ID’si için bilgi listeleme

### 1.c.1 Kaynak Kod

DurakAğacı Sınıfı:

string s = "";

public string IDSearch(TreeNode localRoot,int ID) //return Kiralanılan Durak and Kiralama Saati of given ID.

{

if (localRoot == null)

return "Tree is Empty";

else {

IDSearch(localRoot.leftChild, ID);

for (int i = 0; i < localRoot.data.musteri.Count; i++) {

if (localRoot.data.musteri[i].getMusteriID() == ID)

{

s +="Kiralanılan Durak: " + localRoot.data.getDurakAdı() + "\tKiralanılan Saat: " + localRoot.data.musteri[i].getKiralamaSaati() + "\n";

}

}

IDSearch(localRoot.rightChild, ID);

}

if (s.Equals(""))

return "Müşteri Bisiklet Kiralamamış.";

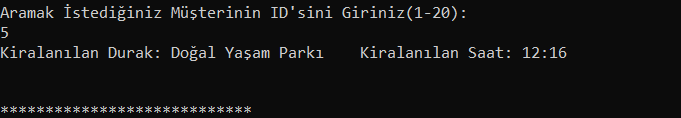
Console.WriteLine(s);

s = "";

return s;

}

### 1.c.2 Ekran görüntüleri



## 1.d Kiralama İşlemi

## public void Rent(TreeNode localRoot,string DurakAdı,string ID) //Rents NormalBisiklet from given DurakAdı for given ID.

## {

## if (localRoot == null)

## {

## return;

## }

## Rent(localRoot.leftChild, DurakAdı, ID);//calls left child recursively

## if (localRoot.data.getDurakAdı().Equals(DurakAdı,StringComparison.OrdinalIgnoreCase))//check if given DurakAdı exist in the array.

## {

## if (localRoot.data.getNormalBisiklet() == 0)//check if there is any NormalBisiklet.

## {

## Console.WriteLine("Durakta Normal Bisiklet Mevcut Değildir.");

## return;

## }

## localRoot.data.setNormalBisiklet(localRoot.data.getNormalBisiklet() - 1);//Rents a NormalBisiklet for given ID and increase the number of BosPark.

## string saat = random.Next(0, 24) + ":" + random.Next(0, 60);

## localRoot.data.musteri.Add(new Musteri(Convert.ToInt32(ID), saat));

## localRoot.data.setBosPark(localRoot.data.getBosPark()+1);

## localRoot.data.setNormalBisiklet(localRoot.data.getNormalBisiklet() - 1);

## }

## Rent(localRoot.rightChild, DurakAdı, ID);//calls right child recursively

## }

## 2.a Hash Tablosuna Ekleme

## Hashtable hash = new Hashtable();

## for (int i = 0; i < matris.Count; i++)

## {

## int[] values = { Convert.ToInt32(matris[i][1]), Convert.ToInt32(matris[i][2]), Convert.ToInt32(matris[i][3]) };

## hash.Add(matris[i][0], values);

## }

## 2.b Hash Tablosu Güncelleme

## foreach(int[] a in hash.Values)

## {

## if (a[0] > 5)

## {

## a[0] -= 5;

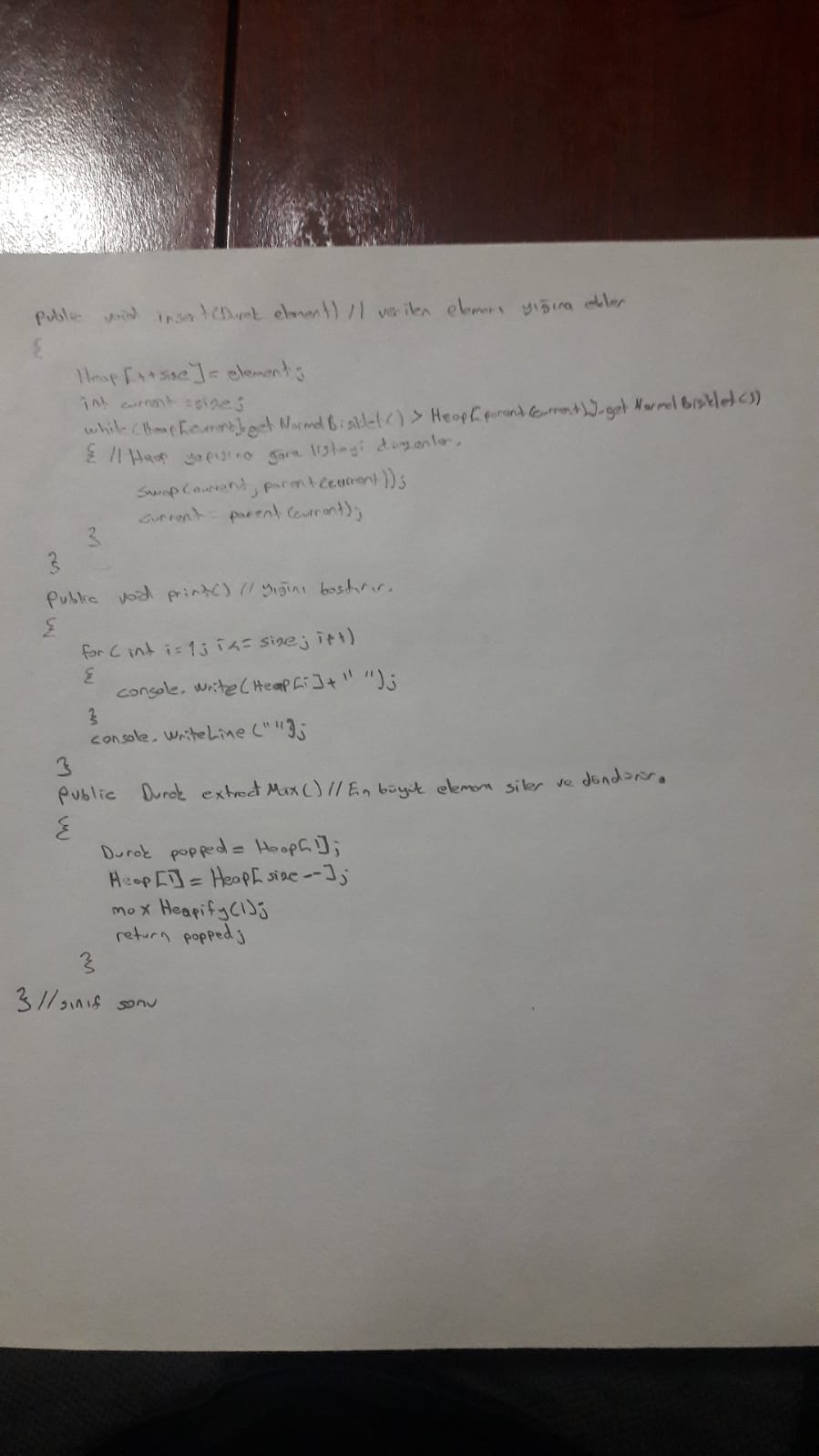
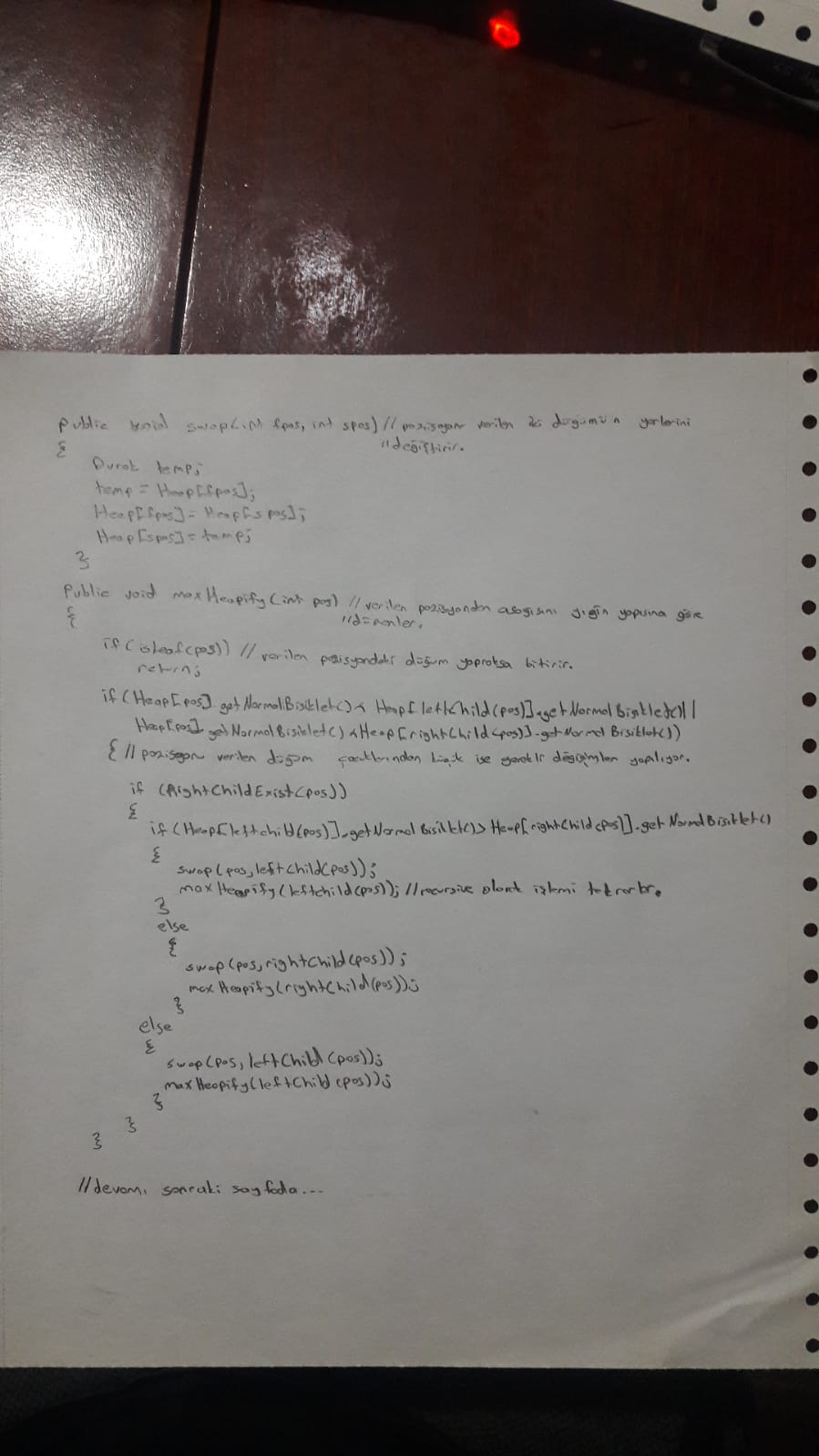
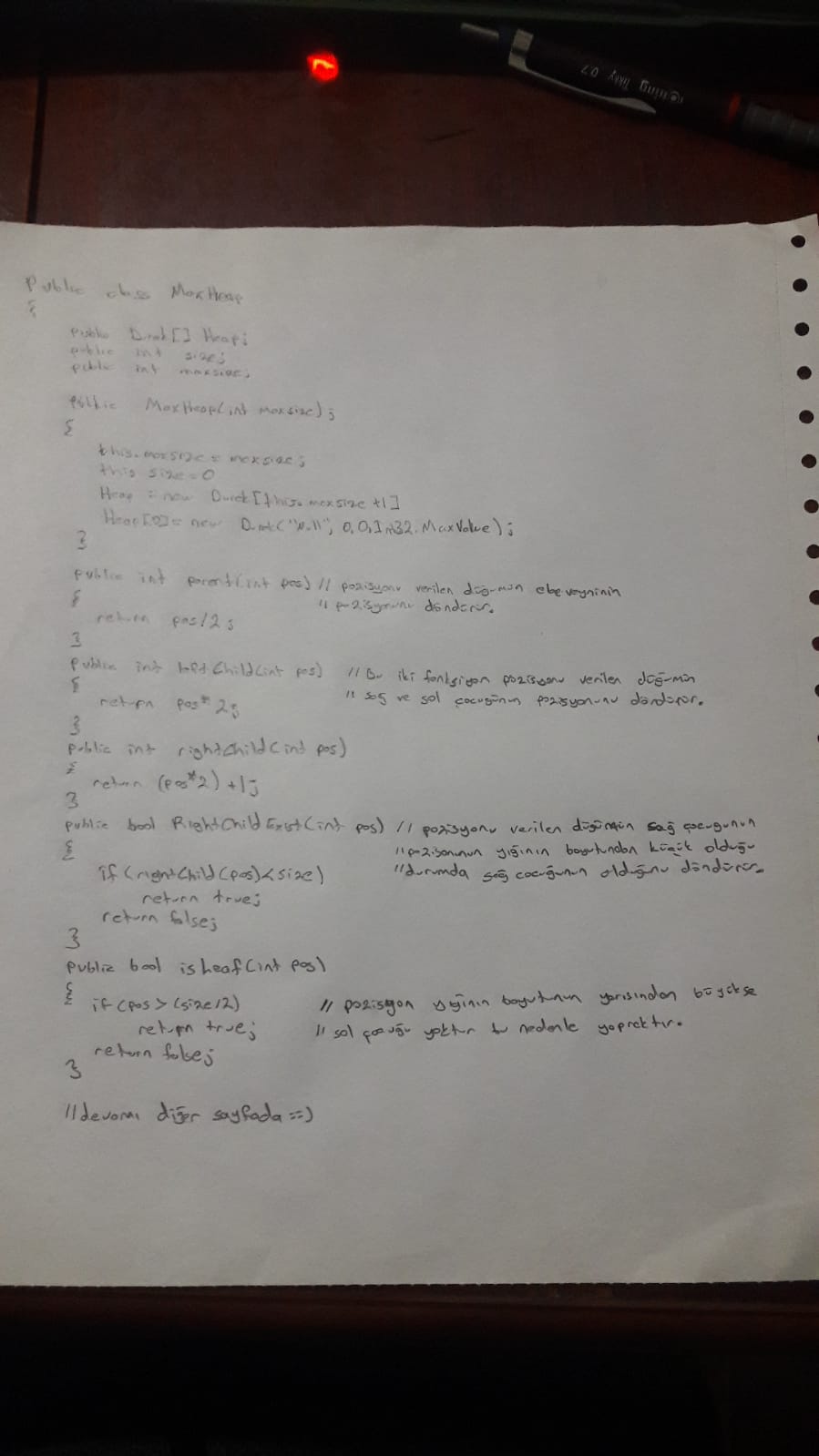
## a[2] += 5;

## }

## }

## 3.a Heap Veri Yapısı Tasarlama

### 3.a.1 Ön Çalışma

.

### 3.a.2 Kaynak Kod

## public class MaxHeap

## {

## public Durak[] Heap;

## public int size;

## public int maxsize;

## // Constructor to initialize an

## // empty max heap with given maximum

## // capacity.

## public MaxHeap(int maxsize)

## {

## this.maxsize = maxsize;

## this.size = 0;

## Heap = new Durak[this.maxsize + 1];

## Heap[0] = new Durak("Null",0,0,Int32.MaxValue) ;

## }

## // Returns position of parent

## public int parent(int pos)

## {

## return pos / 2;

## }

## // Below two functions return left and

## // right children.

## public int leftChild(int pos)

## {

## return (2 \* pos);

## }

## public int rightChild(int pos)

## {

## return (2 \* pos) + 1;

## }

## public bool RightChildExist(int pos) // Check if right child exist

## {

## if (rightChild(pos) < size)

## return true;

## return false;

## }

## // Returns true of given node is leaf

## public bool isLeaf(int pos)

## {

## if (pos > (size / 2))

## {

## return true;

## }

## return false;

## }

## public void swap(int fpos, int spos)

## {

## Durak tmp;

## tmp = Heap[fpos];

## Heap[fpos] = Heap[spos];

## Heap[spos] = tmp;

## }

## // A recursive function to max heapify the given

## // subtree. This function assumes that the left and

## // right subtrees are already heapified, we only need

## // to fix the root.

## public void maxHeapify(int pos)

## {

## if (isLeaf(pos))

## return;

## if (Heap[pos].getNormalBisiklet() < Heap[leftChild(pos)].getNormalBisiklet() ||

## Heap[pos].getNormalBisiklet() < Heap[rightChild(pos)].getNormalBisiklet())

## {

## if (RightChildExist(pos))

## {

## if (Heap[leftChild(pos)].getNormalBisiklet() > Heap[rightChild(pos)].getNormalBisiklet())

## {

## swap(pos, leftChild(pos));

## maxHeapify(leftChild(pos));

## }

## else

## {

## swap(pos, rightChild(pos));

## maxHeapify(rightChild(pos));

## }

## }

## else

## {

## swap(pos, leftChild(pos));

## maxHeapify(leftChild(pos));

## }

## }

## }

## // Inserts a new element to max heap

## public void insert(Durak element)

## {

## Heap[++size] = element;

## // Traverse up and fix violated property

## int current = size;

## while (Heap[current].getNormalBisiklet() > Heap[parent(current)].getNormalBisiklet())

## {

## swap(current, parent(current));

## current = parent(current);

## }

## }

## public void print()

## {

## for (int i = 1; i <= size ; i++)

## {

## Console.Write(Heap[i] + " ");

## }

## Console.WriteLine();

## }

## // Remove an element from max heap

## public Durak extractMax()

## {

## Durak popped = Heap[1];

## Heap[1] = Heap[size--];

## maxHeapify(1);

## return popped;

## }

## }

## 3.b Max Heap düğüm yerleştirme

## public void insert(Durak element)

## {

## Heap[++size] = element;

## // Traverse up and fix violated property

## int current = size;

## while (Heap[current].getNormalBisiklet() > Heap[parent(current)].getNormalBisiklet())

## {

## swap(current, parent(current));

## current = parent(current);

## }

## }

public void swap(int fpos, int spos)

{

Durak tmp;

tmp = Heap[fpos];

Heap[fpos] = Heap[spos];

Heap[spos] = tmp;

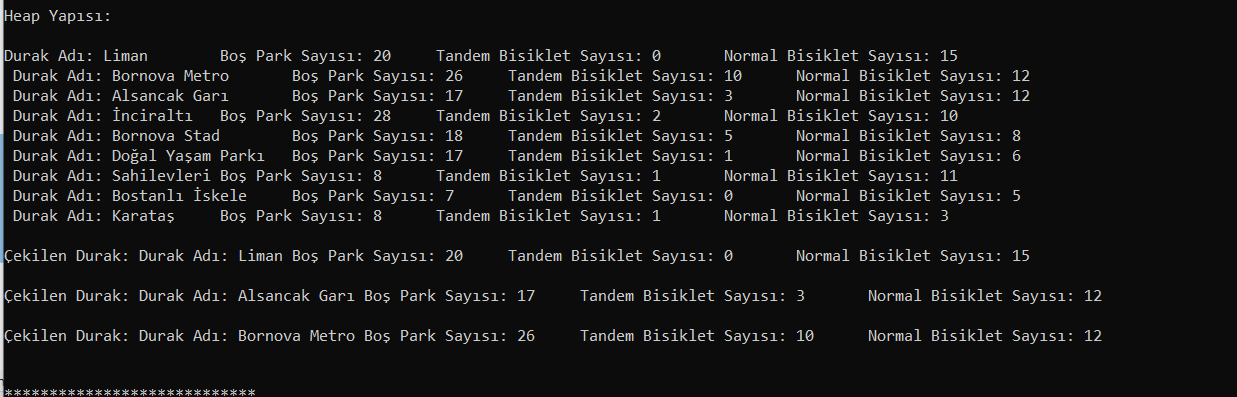
}

## 3.c Heap bilgi çekme

### 3.c.1 Kaynak Kod

3a kaynak kodları içinde mevcuttur.

### 3.c.2 Ekran görüntüleri



## 4.a Simple sorting algoritması

public static void selectionSort(int[] a)

{

for(int i = 0; i < a.Length - 1; i++)

{

int min = i;

for(int j = i + 1; j < a.Length; j++)

{

if (a[j] < a[min])

min = j;

}

swap(i, min, a);

}

}

public static void swap(int fpos,int spos,int[] a)

{

int temp = a[fpos];

a[fpos] = a[spos];

a[spos] = temp;

}

## 4.b Advanced sorting algoritması

## public static void quickSort(int[] a)

## {

## reQuickSort(a,0,a.Length-1);

## }

## 

## public static void reQuickSort(int[]a,int left,int right)

## {

## if (right <= left) //eğer sağ pointer sol pointerdan küçükse liste sıralıdır.

## return;

## else

## {

## int pivot = a[right];//pivot listenin son elemanı olarak belirlenir

## int repivot = partition(a, left, right, pivot);//daha sonra yeni listeye göre sıralama yapılır ve yeni pivot seçilir

## reQuickSort(a, left, repivot - 1);//pivotun sağı ve solu kendi aralarında yeni pivot seçerek tekrar sıralanır

## reQuickSort(a, repivot + 1, right);//ve bu işlem recursive olarak devam eder

## }

## }

## public static int partition(int[] a,int left,int right,int pivot)

## {

## int leftPtr = left - 1;

## int rightPtr = right;

## while (true)

## {

## while (a[++leftPtr] < pivot)//pivottan büyük eleman bulana kadar indexi arttırır

## ;

## while (rightPtr > 0 && a[--rightPtr] > pivot)//pivottan küçük eleman bulana kadar indexi azaltır

## ;

## if (leftPtr >= rightPtr)//sol pointer sağ pointera eşit veya büyük olduğunda döngüyü durdurur

## break;

## else

## swap(leftPtr, rightPtr, a);//sağ ve sol pointerları olması gerektiği gibi değiştirir

## }

## swap(leftPtr, right,a);

## return leftPtr;//son halinde sol pointerı yeni pivot olarak döndürür

## }

## 4.c Sıralama algoritmalarının karşılaştırılması

Selection Sort iç içe 2 for döngüsü içerdiği için zaman karmaşıklığı O(n^2)’dir.Quick Sort ise n kere döndüğü için n ve her seferde 2 parçaya böldüğümüz için log2(n) olacağından zaman karmaşıklığı O(nlog2(n)) şeklindedir.Quick sort en kötü durumda ise O(n^2) olacaktır.Küçük dizilerin sıralanmasında basit sıralama algoritmaları daha yararlı olacaktır.

## 4.d Görselleştirme araçları

Algoritmaların görselleştirilmesi, görselleştirilen algoritmaların kavranması için güzel bir araç.Sözel olarak oturmayan bazı yerleri görsel olarak görmek anlaşılmasını kolaylaştırıyor.Etkileşimli Görselleştirme Araçları ise bana göre videolardan daha etkili çünkü gösterim hızını ve adımları istediğimiz gibi kontrol edebiliyoruz ve bu algoritmanın ve veri yapısının anlaşılmasını çok kolaylaştırıyor.

# 

# Özdeğerlendirme Tablosu

**Özdeğerlendirme Tablosu**

|  |  |  |  |
| --- | --- | --- | --- |
| **Proje 2 Maddeleri** | **Puan** | **Tahmini Not** | **Açıklama** |
| **1 a) Durak nesnelerini oluşturma ve ağaca ekleme** | **10** | **10** | **Yapıldı** |
| **1 b) Ağaç derinliği ve ağaçtaki bilgilerin ekrana listelenmesi** | **10** | **10** | **Yapıldı** |
| **1 c) Verilen müşteri ID’si için bilgi listeleme** | **10** | **10** | **Yapıldı** |
| **1 d)** **Kiralama İşlemi** | **10** | **10** | **Yapıldı** |
| **2 a) Hash Tablosuna Ekleme** | **10** | **10** | **Yapıldı** |
| **2 b) Hash Tablosu Güncelleme** | **5** | **10** | **Yapıldı** |
| **3 a) Heap Veri Yapısı Tasarlama** | **5** | **10** | **Yapıldı** |
| **3 b) Max Heap düğüm yerleştirme** | **5** | **10** | **Yapıldı** |
| **3 c) Heap bilgi çekme** | **5** | **10** | **Yapıldı** |
| **4 a) Simple sorting algoritması** | **4** | **10** | **Yapıldı** |
| **4 b) Advanced sorting algoritması** | **4** | **10** | **Yapıldı** |
| **4 c) Sıralama algoritmalarının karşılaştırılması** | **4** | **9** | **Yapıldı** |
| **4 d) Görselleştirme araçları** | **8** | **9** | **Yapıldı** |
| **5) Özdeğerlendirme Tablosu** | **10** | **10** | **Yapıldı** |
| **Toplam** | **100** | **98** | **Yapıldı** |

**Açıklama kısmında yapıldı, yapılmadı bilgisi ve hangi maddelerin nasıl yapıldığı (ve nelerin yapılmadığı / yapılamadığı) yazılmalıdır. Tahmini not kısmına da ilgili maddeden kaç almayı beklediğinizi yazmalısınız.**