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

02/02/2021

PREPARED BY

05190000061, Oktay Kaloğlu

İçindekiler

ARAMA AĞAÇLARI, YIĞINLAR VE HASH TABLOSU: BİSİKLET KİRALAMA SİSTEMİ

//Visual Studio Community 2019, 16.8.1 and C# used to develop this program.

1.a Durak nesnelerini oluşturma ve ağaca ekleme

```
static (BinarySearchTree,List<Durak>) for1A(String[] duraklar)
        {

            BinarySearchTree ağaç = new BinarySearchTree();
            List<Durak> durakListesi = new List<Durak>();
            //durakların bilgilerinin alınması
            String[] durağınBilgisi = new String[4];

            for (int i=0;i<duraklar.Length;i++) {//ana giriş verisindeki bütün elemanların durak bilgilerinin tek tek gezilmesi
                int count = 0;
                int countForArray = 0; //geçici dizinin elemanlarının gezilebilmesi için.


                for (int y =0;y<duraklar[i].Length;y++)//durağın bilgilerinin oluşturulması
                {
                    if (duraklar[i][y]==',') {//stringde virgülün aranması
                        durağınBilgisi[countForArray]=duraklar[i].Substring(count,(y-count));//virgüle kadarki karakterlerin string
olarak diziye eklenmesi

                        count = y+2;//virgülden sonraki boşluğun eklenmemesi için
                        countForArray++;
                    }

                }
                durağınBilgisi[countForArray] = duraklar[i].Substring(count,duraklar[i].Length-count);//son karakterlerin geçici diziye
eklenmesi

                Durak durak = new Durak(durağınBilgisi[0], int.Parse(durağınBilgisi[1]), int.Parse(durağınBilgisi[2]),
int.Parse(durağınBilgisi[3]));//durak nesnesinin oluşturulması

                Durak kopyaDurak = new Durak(durağınBilgisi[0], int.Parse(durağınBilgisi[1]), int.Parse(durağınBilgisi[2]),
int.Parse(durağınBilgisi[3]));//ileride ağaçta yapıcağım değişikliklerin hash table etkilememesi için bir kopyasını veriyorum.
                durakListesi.Add(kopyaDurak);
                Random random = new Random();


                int rastgeleSayı = random.Next(1,11);
                for(int y =0;y< rastgeleSayı; y++)//List tipinde bir veri yapısı içine 1 ile 10 adet arasında random sayıda rastgele
Müşterteri eklenmesi
                {
                    Müşteri müşteri = new Müşteri();
                    durak.müşteriler.Add(müşteri);
                }

                ağaç.AddToTree( durak);


            }
class Müşteri
    {
        public int müşteriID = 0;

        public int saat = 0;
        public int dakika = 0;
        public Müşteri() {
            Random random = new Random();
            this.saat = random.Next(0, 24);
            this.dakika = random.Next(0, 60);
            this.müşteriID = random.Next(1,21);
        }
        public Müşteri(int ID)
        {
            this.müşteriID = ID;

            Random random = new Random();
            this.saat = random.Next(0, 24);
            this.dakika = random.Next(0, 60);
        }
        public String getZaman()
        {
            return saat.ToString() + ":" + dakika.ToString(); ;
        }

        public String ToString()
        {
            return müşteriID+" "+getZaman();
        }

    }
    class Durak
    {
        public String durakAdı = "";
        public int boşPark = 0;
        public int tandemBisiklet = 0;
        public int normalBisiklet = 0;
```

```
        public List<Müşteri> müşteriler;

        public Durak(String nam,int par,int tan,int bis)
        {
            this.durakAdı = nam;
            this.boşPark = par;
            this.tandemBisiklet = tan;
            this.normalBisiklet = bis;
            this.müşteriler = new List<Müşteri>();
        }

        public String ToString()
        {
            return durakAdı+" "+boşPark+" "+tandemBisiklet+" "+normalBisiklet;
        }



    }

    class BinarySearchTree{

        public Node root;

        internal class Node
        {
            internal Durak durak;
            internal Node parent;
            internal Node leftChild;
            internal Node rightChild;
            internal Node()
            {
                this.durak    =null;
                this.rightChild = null;
                this.leftChild= null;
                this.parent = null;
            }

        }
        public BinarySearchTree()
        {
            this.root = new Node();
        }
        public void AddToTree(Durak dur) {//büyük ise sağ çocuk
                                //recursive de yazıalbilir

            Node eklenecekNode = new Node();
            eklenecekNode.durak = dur;
            if (root.durak!=null)//kök boş değilse girilen eleman ağaç içerisinde gezilerek eklenmelidir.
            {
                Node tempNode = root;
                while (true)
                {

                    if (String.Compare(tempNode.durak.durakAdı, dur.durakAdı, true, new CultureInfo("tr-TR")) == -1)//-1 ise sağında
kalmaktadır

                    {
                        if (tempNode.rightChild == null)//sağ çocuk boş ise ekleme buraya yapılmalıdır.
                        {
                            tempNode.rightChild = eklenecekNode;
                            eklenecekNode.parent = tempNode;
                            break;
                        }
                        else//boş olan node bulunana kadar bu kontrol işlemleri devam ettirilmeilidir.
                        {
                            tempNode = tempNode.rightChild;
                        }
                    }
                    else//sol taraf
                    {
                        if (tempNode.leftChild== null)//sol çocuk boş ise ekleme buraya yapılmalıdır.
                        {
                            tempNode.leftChild= eklenecekNode;
                            eklenecekNode.parent = tempNode;
                            break;
                        }
                        else//boş olan node bulunana kadar bu kontrol işlemleri devam ettirilmeilidir.
                        {
                            tempNode = tempNode.leftChild;
                        }
                    }
                }
            }
            else//kök boş ise köke eklenecektir
            {
                root.durak=dur;
            }

        }

        public int DerinlikBul(Node tempNode, int sayı = 0)
        {

            if (tempNode!=null)//temp node null olana kadar öncelikle en sol çocuğa ulaşıçak ve null olmadan önceki seviyeyi döndürücek.
                                //en soldaki çocuğun derinliğini sağındaki kardeşinin en derin çoğunun derinliği ile karşılaştırıp sırası
ile yukarı çıkıcaktır.
            {
                int ikisideEşitGirsin = sayı+1;
```

```
            int sol = DerinlikBul(tempNode.leftChild, ikisideEşitGirsin);
            int sağ = DerinlikBul(tempNode.rightChild, ikisideEşitGirsin);
            if (sağ > sol)
            {
                return sağ;
            }
            return sol;

        }

        return sayı-1;//nulldan önceki derinliğin döndürülmesi

    }


    public void AğaçYaz(Node tempNode)
    {
        if (tempNode!= null)
        {
            AğaçYaz(tempNode.leftChild);
            AğaçYaz(tempNode.rightChild);
            Console.WriteLine(tempNode.durak.ToString());
            foreach (Müşteri müş in tempNode.durak.müşteriler)
            {
                Console.WriteLine(müş.ToString());
            }
            Console.WriteLine("---------");


        }
    }
    public void SearchFoID(int id,Node tempNode)//post order treversal
    {

        if (tempNode!= null)
        {
            SearchFoID(id,tempNode.leftChild);
            SearchFoID(id,tempNode.rightChild);
            Console.WriteLine(tempNode.durak.ToString());
            foreach (Müşteri müş in tempNode.durak.müşteriler)
            {
                if (müş.müşteriID==id)
                {
                    Console.WriteLine(tempNode.durak.durakAdı + " saat : " + müş.getZaman());
                }
            }
        }
    }
    public void Kiralama(String durak,int id)
    {
        Node temp = this.root;
        while (true)
        {
            if (temp!=null) {
                if (temp.durak != null & durak != null)
                {

                    int sonuç = String.Compare(temp.durak.durakAdı, durak, true, new CultureInfo("tr-TR"));
                    if (sonuç == -1) {
                        temp = temp.leftChild;
                    }
                    else if (sonuç == 1)
                    {
                        temp = temp.rightChild;
                    }
                    else//doğru durak bulunmuştur
                    {
                        Müşteri müş = new Müşteri(id);
                        temp.durak.müşteriler.Add(müş);
                        temp.durak.boşPark++;
                        temp.durak.normalBisiklet--;
                        break;//bulunduysa devam edilmesine gerek yoktur.
                    }
                }
            }
            else//bütün ağaç dolaşılıp bulunamadı
            {
                Console.WriteLine("bütün ağaç dolaşılıp bulunamadı");
                break;
            }
        }
    }
}
```

1.b Ağaç derinliği ve ağaçtaki bilgilerin ekrana listelenmesi

1.b.1 Kaynak Kod

```
static void For1B(BinarySearchTree ağaç)

    {
        Console.WriteLine("Ağacın derinliği : "+ağaç.DerinlikBul(ağaç.root));
```

```
            ağaç.AğaçYaz(ağaç.root);
        }
```

Class ağaç methodları:

```
        public int DerinlikBul(Node tempNode, int sayı = 0)
        {

            if (tempNode!=null)//temp node null olana kadar öncelikle en sol çocuğa ulaşıçak ve null olmadan önceki seviyeyi döndürücek.
                               //en soldaki çocuğun derinliğini sağındaki kardeşinin en derin çoğunun derinliği ile karşılaşıtırıp sırası
ile yukarı çıkıcaktır.
            {
                int ikisideEşitGirsin = sayı+1;
                int sol = DerinlikBul(tempNode.leftChild, ikisideEşitGirsin);
                int sağ = DerinlikBul(tempNode.rightChild, ikisideEşitGirsin);
                if (sağ > sol)
                {
                    return sağ;
                }
                return sol;

            }

            return sayı-1;//nulldan önceki derinliğin döndürülmesi

        }

        public void AğaçYaz(Node tempNode)
        {
            if (tempNode!= null)
            {
                AğaçYaz(tempNode.leftChild);
                AğaçYaz(tempNode.rightChild);
                Console.WriteLine(tempNode.durak.ToString());
                foreach (Müşteri müş in tempNode.durak.müşteriler)
                {
                    Console.WriteLine(müş.ToString());
                }
                Console.WriteLine("---------");


            }
        }
```

1.b.2 Ekran görüntüleri

```
Ağacın derinliği : 4
Bornova Hastahane 5 3 15
16 4:3
19 23:19
12 3:0
11 17:58
9 19:11
12 19:39
17 13:38
19 10:35
---------
Basmane 55 2 12
7 3:20
15 19:34
---------
Çankaya 13 0 2
10 7:44
13 2:13
15 13:50
15 10:28
8 14:16
16 4:0
16 21:26
---------
Halkapınar 23 4 0
16 7:27
17 11:39
14 3:12
18 18:42
14 7:58
18 14:0
---------
Hilal 6 4 11
2 6:42
6 23:55
16 19:46
19 17:14
12 22:1
5 4:14
7 9:55
16 8:4
17 19:41
---------
Doğal Yaşam Parkı 17 1 6
10 16:7
10 20:31
12 22:9
8 1:39
5 2:1
```

1.c Verilen müşteri ID'si için bilgi listeleme

1.c.1 Kaynak Kod

```
static void For1C(BinarySearchTree ağaç)
        {
            Console.WriteLine("Lütfen aranacak Müşteri ID'sini giriniz : ");
            int id = Convert.ToInt32(Console.ReadLine());
            ağaç.SearchFoID(id,ağaç.root);
        }
```

Class ağaç kodu:

```
public void SearchFoID(int id,Node tempNode)//post order treversal
        {

            if (tempNode!= null)
            {
                SearchFoID(id,tempNode.leftChild);
                SearchFoID(id,tempNode.rightChild);

                foreach (Müşteri müş in tempNode.durak.müşteriler)
                {
                    if (müş.müşteriID==id)
                    {
                        Console.WriteLine(tempNode.durak.durakAdı + " saat : " + müş.getZaman());
                    }

                }
            }
        }
```

1.c.2 Ekran görüntüleri

```
Lütfen aranacak Müşteri ID'sini giriniz :
2
Halkapınar saat : 22:4
Halkapınar saat : 3:42
Hilal saat : 18:29
Stadyum saat : 20:49
```

1.d Kiralama İşlemi

```
ağaç.Kiralama("İnciraltı",20);
```

Class ağaç kodu:

```
public void Kiralama(String durak,int id)
        {
            Node temp = this.root;
            while (true)
            {
                if (temp!=null) {
                    if (temp.durak != null & durak != null)
                    {

                        int sonuç = String.Compare(temp.durak.durakAdı, durak, true, new CultureInfo("tr-TR"));
                        if (sonuç == -1) {
                            temp = temp.leftChild;
                        }
                        else if (sonuç == 1)
                        {
                            temp = temp.rightChild;
                        }
                        else//doğru durak bulunmuştur
                        {
                            Müşteri müş = new Müşteri(id);
                            temp.durak.müşteriler.Add(müş);
                            temp.durak.boşPark++;
                            temp.durak.normalBisiklet--;
                            break;//bulunduysa devam edilmesine gerek yoktur.
                        }
                    }
                }
                else//bütün ağaç dolaşılıp bulunamadı
                {
                    Console.WriteLine("bütün ağaç dolaşılıp bulunamadı");
                    break;
                }
            }
        }
    }
```

2.a Hash Tablosuna Ekleme

```
List<Durak>[]  hashedDuraklar = For2A(durakListesi);
        static int hasingFuntion(String ad, int numberOfElements)//harflerin ascii koduna göre toplanması ve  bu toplamın eleman sayısına
göre modunun alınmasıyla anahtar oluşturulması
        {
            int toplam = 0;
            byte[] asciiBytes = Encoding.ASCII.GetBytes(ad);
            for (int i = 0; i < asciiBytes.Length; i++)
            {
                toplam += asciiBytes[i];
            }
            return toplam -(toplam/numberOfElements)*numberOfElements;//kalanın bulunması
        }

        static List<Durak>[]  For2A(List<Durak> durakListesi)
                                                                    // kelimelerin ascii kod değerlerinin eleman sayısına göre modunun alınıp
index yaratılıyor.
                                                                       // aynı index birden fazla kelime için üretilebilir bu nedenle hata
oluşumunu engelllemek için
                                                                       //ayrı zincirleme yöntemini tercih ettim.
        {
            int durakSayısı = durakListesi.Count;
            List<Durak>[] duraklar = new List<Durak>[durakSayısı];//durak sayısına göre , eleamanları durakları içeren liste olan bir
listenin oluşturulması.

            foreach(Durak durak in durakListesi)
            {
                int index =hasingFuntion(durak.durakAdı, durakSayısı);
                if (duraklar[index]== null) //key e göre hash table a ekleme
                {
                    List<Durak> eklenecekDurak = new List<Durak>();
                    eklenecekDurak.Add(durak);
                    duraklar[index] = eklenecekDurak;
                }
                else
                {
                    duraklar[index].Add(durak);
                }
            }
            return duraklar;

        }
```

2.b Hash Tablosu Güncelleme

```
static void For2B(List<Durak>[] durakListesi)
        {
            for (int i = 0; i < durakListesi.Length; i++)
            {
                if (durakListesi[i]!=null)//ayrı zincirleme hash tableda bazı indexler doldurlmamış olabilir
                {
                    foreach (Durak durak in durakListesi[i]) {
                        if (durak.boşPark>5) {
                            durak.boşPark -= 5;
                            durak.normalBisiklet += 5;
                        }
                    }
                }
            }

        }
```

3.a Heap Veri Yapısı Tasarlama

3.a.1 Ön Çalışma

```
Class MaxHeap{
Durak[] duraklar
Size=0;
maxSize;
MaxHeap(max){ //construction
 maxSize=max
 duraklar=new Durak[maxSize];
}
Add(durak){
if(size<maxsize){ // yer varsa eklenebilir
index=size;//size son eklenmiş nesreden bir sonraki indexi gösterir.
duraklar[index]=durak; // nesnenin dizige eklenmesi
size++; //item sayısı eklenmeden dolayı artt.
while(true){
 if(duraklar[index].normalbüklet>duraklar.[parent].normalbüklet){
 duraklar[index],duraklar[parent] yer değiştir.
 index=parent;
 }else{
   break; // değişim yapılamadıysa, artık doğru sıralamadadır.
   }
}
}

Durak getMax(){ //en üstteki itemi geri döndürecek
duraklar[0] ile duraklar[size-1](en sondaki item) yer değiştir.
durak dizbir=durak[size-1];
 durak[size-1]=null;
while(true){
 if(solçocuk boş ise){ // sol çocuk boş ise sağda boştur.(soldan sağa doldurduğu için)
   break; //düzenlemeye ihtiyac yok.
 }else{
   if(sağçocuk boş ise){ //sadece sol çocukla kıyaslanacak
   if(parent>solcocuk){
     break;// baba daha büyük olduğu için değişime ne düzenlemeye gerekiyok.
   }// devamı sayfa2 de
```

```
//Devam
else { //değişim lazım
 duraklar[parent], duraklar[solçocuk] değiştir.
  index = solçocuk;
}
} else { // sol ve sağ çocuk var ise
  if (parent > solcocuk && parent > sağcocuk) { // değişime gerek yok
    break;
  } else { // cocuklar büyük
    duraklar[parent] duraklar[büyük çocuk] değiştir.
    index = büyük çocuk;
  }
} //while kapandı.
Return döndür;
}
}
```

maxheap

Dizide, nesnelerin ağaç gibi bir hiyerarşi ile tutulup dizi elemanı erişim hızının kombine edilmesini sağlar. Log iki tabanında ekleme ve çıkartma süresi ve indexi bulma darama gerçek zamanda ulaşılmasını sağlar.

Ekleme metodu: dizinin ensonuna direk ekler ve eklenen nesne sadece parentları ile kıyaslanarak doğru yeri arar. En kötü durumda ($\log_2(n)$) de yeri bulunur.

Çıkarma metodu; dizinin her zaman 0. indexli nesnesi en büyük / en küçük item'i içerir. Bu itemin geri döndürülmesi çok basittir. Kalanlık sağlanması adına listenin en sonundaki nesnesi ile yerdeğiştirilir ve listeden çıkartılır. en başa alınan nesnenin yeri en kötü $\log_2(n)$ değişimde yerine ulaşır.

3.a.2 Kaynak Kod

```
        static MaxHeap For3B(List<Durak> durakListesi)
        {
            MaxHeap heap = new MaxHeap(durakListesi.Count);
            for (int i = 0; i < durakListesi.Count; i++)
            {
                heap.Add(durakListesi[i]);
            }
            return heap;
        }
```

Heap classı:

```
class MaxHeap
    {
        private Durak[] duraklar;
        private int size=0;
        private int maxSize=1;

        public MaxHeap(int mxs = 1) {
            this.maxSize = mxs;
            duraklar = new Durak[maxSize];
        }

        public int Parent(int index) { return index / 2; }
        public int LeftChild(int index) { return index * 2+1 ; }
        public int RightChild(int index) { return index * 2 +2; }

        public Durak getMax() {//en fazla bisiklete sahip nesneyi heapten çıkartıp geri döndürür.
            Durak döndür = duraklar[0];//ilk itemin döndürülmesi

            if (size > 0) {//heap boş olabilir.
                duraklar[0] = duraklar[size-1];//son itemin başa getirilmesi
                duraklar[size-1] = null;//son kullanılmış indexin boşaltılması
                size--;
                int index = 0;//heap gezilirken yardımcı olucak
                while (true)
                {
                    int indexOfLeftChild = LeftChild(index);
                    int indexOfRightChild = RightChild(index);
                    if (indexOfLeftChild < maxSize || indexOfRightChild < maxSize)
                    {
                        if (duraklar[LeftChild(index)] != null)//sol çocuk boş mu
                        {
                            if (duraklar[RightChild(index)] != null)//eğer sağ çocukta boş değilse ikisinden büyük olan ile parent
değiştirilmelidir.
                            {
                                int büyükIndex;
                                if (duraklar[LeftChild(index)].normalBisiklet >= duraklar[RightChild(index)].normalBisiklet) { büyükIndex
= LeftChild(index); }//sol çocuk büyük veya eşit
                                else { büyükIndex = RightChild(index); }//sağ çocuk büyük
                                if (duraklar[index].normalBisiklet < duraklar[büyükIndex].normalBisiklet)
                                {
                                    Durak temp = duraklar[index];
                                    duraklar[index] = duraklar[büyükIndex];
                                    duraklar[büyükIndex] = temp;
                                    index = büyükIndex;//değişim olduğu için döngü en az bir kere daha devam etmelidir.
                                }
                                else//parent daha büyük olduğu için değişim yaplılamaz döngüden çıkılmalı
                                {
                                    break;
                                }

                            }

                            else//sağ çocuk boşsa sadece sol çocuk ile kontrol edilmeli
                            {
                                int büyükIndex = LeftChild(index);
                                if (duraklar[index].normalBisiklet < duraklar[büyükIndex].normalBisiklet)
                                {
                                    Durak temp = duraklar[index];
                                    duraklar[index] = duraklar[büyükIndex];
                                    duraklar[büyükIndex] = temp;
                                    index = büyükIndex;//değişim olduğu için döngü en az bir kere daha devam etmelidir.
                                }
                                else//parent daha büyük olduğu için değişim yaplılamaz döngüden çıkılmalı
                                {
                                    break;
                                }

                            }
                        }
                        else//sol çocuk boşsa sağ çocukta boştur. heapte artık düzenleme yapılamsına gerek yoktur.
                        {

                            break;
                        }
                    }
                    else { break; }
                }
            }
            return döndür;
        }
        public void Add(Durak durak) {
            if (size<maxSize) {//heapde yer varsa ekleme yapılabilir
                int index = size;
                duraklar[index] = durak;//son indexe nesnenin eklenmesi
```

```
            size++;
            if (index != 0) {//ilk item ise düzenleme yapılmasına gerek yoktur.
                while (true)
                {
                    if (duraklar[index].normalBisiklet > duraklar[Parent(index)].normalBisiklet)
                    {
                        int parentIndex = Parent(index);
                        Durak temp = duraklar[parentIndex];
                        duraklar[parentIndex] = duraklar[index];
                        duraklar[index] = temp;
                        index = parentIndex;
                    }
                    else
                    {
                        break;//büyük değilse düzenleme ihtiyacı bitmiştir.
                    }
                }
            }

        }
    }
    public void PrintMaxHeap()
    {
        for (int i = 0;i<size;i++) {
            Console.WriteLine(duraklar[i].ToString());
        }
        Console.WriteLine("--------------");

    }
}
```

3.b Max Heap düğüm yerleştirme

//3-b maddesi için yazmış olduğunuz kodları ve açıklamaları buraya ekleyiniz

```
    public void Add(Durak durak) {
        if (size<maxSize) {//heapde yer varsa ekleme yapılabilir
            int index = size;
            duraklar[index] = durak;//son indexe nesnenin eklenmesi
            size++;
            if (index != 0) {//ilk item ise düzenleme yapılmasına gerek yoktur.
                while (true)
                {
                    if (duraklar[index].normalBisiklet > duraklar[Parent(index)].normalBisiklet)
                    {
                        int parentIndex = Parent(index);
                        Durak temp = duraklar[parentIndex];
                        duraklar[parentIndex] = duraklar[index];
                        duraklar[index] = temp;
                        index = parentIndex;
                    }
                    else
                    {
                        break;//büyük değilse düzenleme ihtiyacı bitmiştir.
                    }
                }
            }

        }
    }
```

3.c Heap bilgi çekme

3.c.1 Kaynak Kod

```
    static void For3C(MaxHeap heap)
    {
        for (int i =0;i<3;i++)
        {
            Console.WriteLine(heap.getMax().ToString());
        }
    }
```

Class:

```
public Durak getMax() {//en fazla bisiklete sahip nesneyi heapten çıkartıp geri döndürür.
        Durak döndür = duraklar[0];//ilk itemin döndürülmesi

        if (size > 0) {//heap boş olabilir.
            duraklar[0] = duraklar[size-1];//son itemin başa getirilmesi
            duraklar[size-1] = null;//son kullanılmış indexin boşaltılması
            size--;
            int index = 0;//heap gezilirken yardımcı olucak
            while (true)
            {
                int indexOfLeftChild = LeftChild(index);
                int indexOfRightChild = RightChild(index);
                if (indexOfLeftChild < maxSize || indexOfRightChild < maxSize)
                {
                    if (duraklar[LeftChild(index)] != null)//sol çocuk boş mu
                    {
```

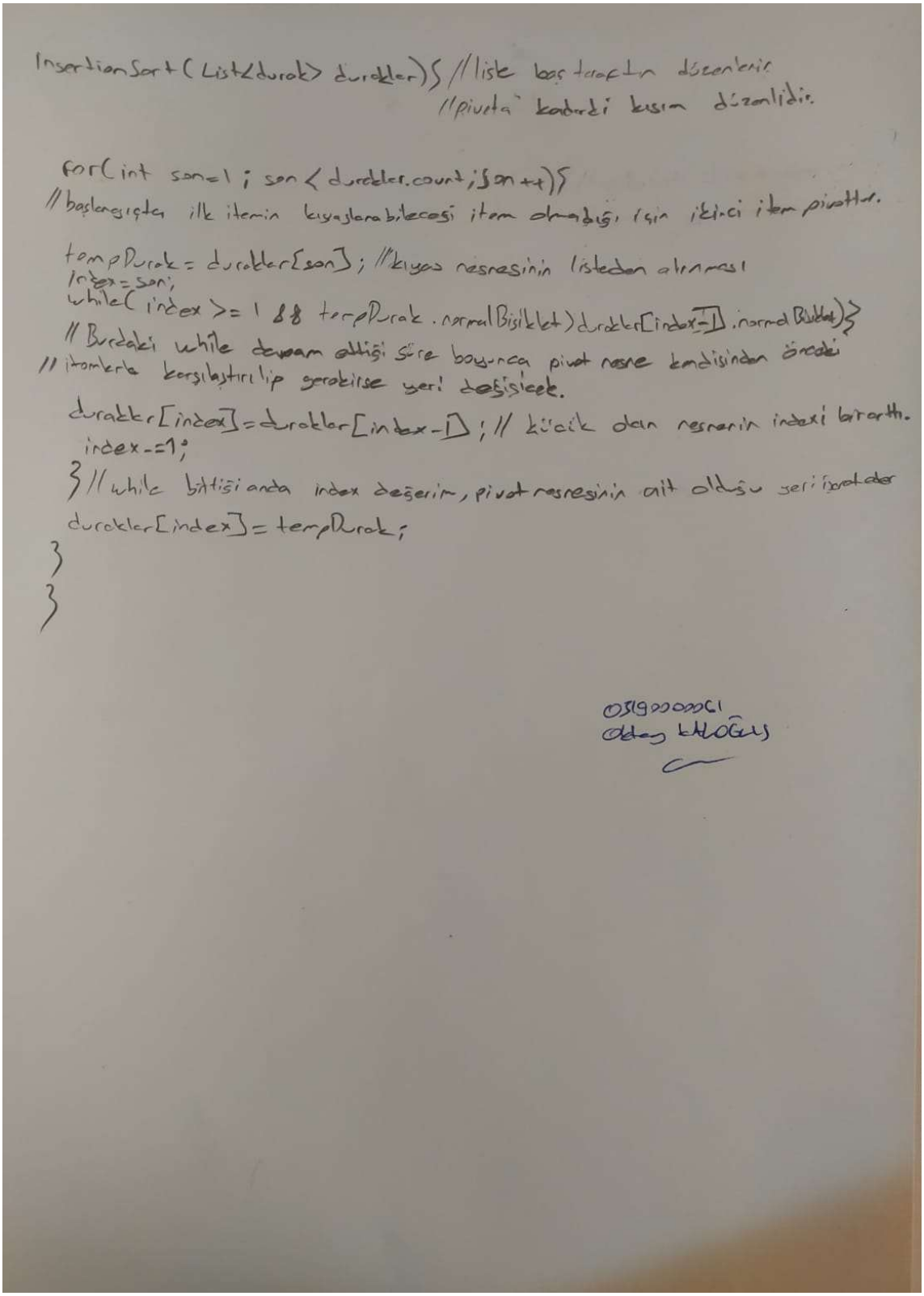
```
                        if (duraklar[RightChild(index)] != null)//eğer sağ çocukta boş değilse ikisinden büyük olan ile parent
değiştirilmelidir.
                        {
                            int büyükIndex;
                            if (duraklar[LeftChild(index)].normalBisiklet >= duraklar[RightChild(index)].normalBisiklet) { büyükIndex
= LeftChild(index); }//sol çocuk büyük veya eşit
                            else { büyükIndex = RightChild(index); }//sağ çocuk büyük
                            if (duraklar[index].normalBisiklet < duraklar[büyükIndex].normalBisiklet)
                            {
                                Durak temp = duraklar[index];
                                duraklar[index] = duraklar[büyükIndex];
                                duraklar[büyükIndex] = temp;
                                index = büyükIndex;//değişim olduğu için döngü en az bir kere daha devam etmelidir.
                            }
                            else//parent daha büyük olduğu için değişim yaplılamaz döngüden çıkılmalı
                            {
                                break;
                            }

                        }

                        else//sağ çocuk boşsa sadece sol çocuk ile kontrol edilmeli
                        {
                            int büyükIndex = LeftChild(index);
                            if (duraklar[index].normalBisiklet < duraklar[büyükIndex].normalBisiklet)
                            {
                                Durak temp = duraklar[index];
                                duraklar[index] = duraklar[büyükIndex];
                                duraklar[büyükIndex] = temp;
                                index = büyükIndex;//değişim olduğu için döngü en az bir kere daha devam etmelidir.
                            }
                            else//parent daha büyük olduğu için değişim yaplılamaz döngüden çıkılmalı
                            {
                                break;
                            }

                        }
                    }
                    else//sol çocuk boşsa sağ çocukta boştur. heapte artık düzenleme yapılamsına gerek yoktur.
                    {

                        break;
                    }
                }
                else { break; }
            }
        }
        return döndür;
    }
```

3.c.2 Ekran görüntüleri

//3-c maddesi için üretilen konsol/ekran görüntüsünü buraya ekleyiniz

```
Basmane 50 2 17
Sahilevleri 3 1 16
Hilal 1 4 16
```

4.a Simple sorting algoritması




```
        static void For4A(List<Durak> duraklistesi)//insertion sort. normal bisiklet sayıları için sıralama yapılmıştır.büyükten küçüğe sıralıyor.
        {
            Durak temp = null;
```

```
                //başlangıçta ilk itemin sıralanmasına gerek yoktur.
            for (int sortedLast=1 ; sortedLast<duraklistesi.Count;sortedLast++)//listedeki bütün itemların dönülmesi için
            {
                temp = duraklistesi[sortedLast];//bellekte gecici olarak işlem yapılan nesnenin tutulması
                int index = sortedLast;//değişim işlemlerine başlanılıcak index
                while (index >= 1 && temp.normalBisiklet > duraklistesi[index-1].normalBisiklet)//başlangıç noktasından e
                {
                    duraklistesi[index ] = duraklistesi[index-1];
                    index--;
                }
                duraklistesi[index] = temp;
            }
            foreach (Durak dur in duraklistesi) { Console.WriteLine(dur.ToString()); }
        }
```

4.b Advanced sorting algoritması

```
   static void For4B(List<Durak> duraklistesi)//quick sort. Normal bisiklet sayıları için sıralama yapılmıştır.büyükten küçüğe doğru
sıralıyor.
        {
            //diğer fonksiyonlar listenin girildiği fonksiyonun scopu içerisinde oldukları için listenin diğer fonksiyonlara parametre
olarak girilmesi gerek kalmıyor
            quickSort(0,duraklistesi.Count-1);//programın çalıştırılması
            foreach (Durak dur in duraklistesi) { Console.WriteLine(dur.ToString()); } // sıralama sonrasında listenin yazdırılma
kontrolü


            void quickSort(int baş, int son)
            {
                if (baş < son)
                {

                    int pivot = partition(baş, son);//pivot olarak kullanılan nesne listede olması gereken yerde

                    quickSort(baş, pivot - 1);  // pivottan önceki gurupun sıralanması
                    quickSort(pivot + 1, son); // pivottan sonraki gurupun sıralanması
                }
            }

            int partition(int baş, int son)
            {

                Durak temp=null;//bellekte gecici olarak işlem yapılan nesnenin tutulması için


                int pivot = duraklistesi[son].normalBisiklet;// pivot değernin atanması

                int i = (baş - 1);  // en baştaki itemin indexi

                for (int j = baş; j <= son - 1; j++)
                {

                    if (duraklistesi[j].normalBisiklet > pivot)//şuanki nesnenin değeri pivottan büyükse değişim yapılmalı
                    {
                        i++;    // en küçük itemin indexinin arttırılması

                        //nesnelerin indexlerinin değiştirilmesi.
                        temp = duraklistesi[i];
                        duraklistesi[i] = duraklistesi[j];
                        duraklistesi[j] = temp;

                    }
                }
                //sonuncu nesnenin pivotun yerine getirilmesideğiştirilmesi.
                temp = duraklistesi[i+1];
                duraklistesi[i+1] = duraklistesi[son];
                duraklistesi[son] = temp;

                return (i + 1);//pivotun döndürülmesi
            }
```

4.c Sıralama algoritmalarının karşılaştırılması

İnsertion sort için time complexity :

En iyi durumda sıralı durumudur. Bu durumda hiç değişim yapılmadan sadece eleman sayısı kadar kontrol edilicek yani O(N).

En kötü durumda ters sıralı olmasıdır. Bu durumda sırası ile her item için sırası ile kendisinden bir önceki bütün itemlerle kıyaslanması gerekicektir. Kontrol ve değişim sayısı sırsası ile 2. Itemden başlayarak 1,2,3,4,5,6,7…….n-1 dir. Toplam işlem sayısı = ((n-1+1)/2)*(n-1-1+1)=n*(n-1)/2. O(N^2).

Quick sort için time complexity:

En kötü durumu sıralı durumda olmasıdır. Bu şeçilen pivotlar her durumda listelerin sonunda ki itemler olarak kalıcaklar ve listeleri iyi bir şekilde ikiye bölemeyeceği için her n item için n tane karşılaştırılma yapılması gerekicek.O(N^2)

En iyi durumda seçilen pivotlar her zaman listeyi tam olarak 2 ye bölüceklerdir. $Log_2(N)$ de alt listeler 1 e inicektir. Her adımda n tane iteme bakılacaktır bu yüzden O(N*log(N))dur.

Bu iki methot arasından liste sıralı ise en verimli çalışıcak methot insertion sorttur. Düşük eleman sayılı listelerin düzenlenmesinde de gene insertion sort daha verimlidir çünkü quick sortun kurulması pivot seçimi olana kadar zaten insertion sort düzenlemeye ve değişme başlamış olur. Eğer eleman sayısı az ve yarı sıralı ise kesinlikle insertion sort daha verimli çalışıcaktır. Eleman sayısı çok fazla olan listelerde ise quick sort daha verimli çalışmaya başlayacaktır.

4.d Görselleştirme araçları

Videodaki gibi önceden hesaplanmış, belirlenmiş çubukların yerdeğiştirmelerini izlemek ilk bakışta yeterince öğretici olmamaktadır. Algoritmik olarak metotların ne yaptığının adım adım görsel olarak gösterilmesi ve algortitmanın ne durumda olduğunun gösterilmesi açıklayıcılık ve öğreticilik konusunda daha başarılıdır.

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	5	Yapıldı
3 a) Heap Veri Yapısı Tasarlama	5	5	Yapıldı
3 b) Max Heap düğüm yerleştirme	5	5	Yapıldı
3 c) Heap bilgi çekme	5	5	Yapıldı
4 a) Simple sorting algoritması	4	4	Yapıldı
4 b) Advanced sorting algoritması	4	4	Yapıldı
4 c) Sıralama algoritmalarının karşılaştırılması	4	4	Yapıldı
4 d) Görselleştirme araçları	8	8	Yapıldı
5) Özdeğerlendirme Tablosu	10	10	Yapıldı
Toplam	100	100	

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.