

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

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

**2021–2022 FALL SEMESTER**

**PROJECT-3 REPORT**

**(Search Tree, Heap, Hash Table, Food Delivery Company)**

**DELIVERY DATE**

15/01/2022

**PREPARED BY**

05200000045, Şevval GÖNÜL

İçindekiler

[1.a YemekSınıfı nesnesinin ve Yemek Sipariş Ağacının oluşturulması 3](#_Toc92533878),4,5

[1.b Ağaç derinliği ve ağaçtaki bilgilerin ekrana listelenmesi 5](#_Toc92533879)

[1.b.1 Kaynak Kod 5,6](#_Toc92533880)

[1.b.2 Ekran görüntüleri 6](#_Toc92533881)

[1.c 150 TL üstündeki siparişlerin bilgilerini listeleme 7](#_Toc92533882)

[1.c.1 Kaynak Kod 7](#_Toc92533883)

[1.c.2 Ekran görüntüleri 7](#_Toc92533884)

[1.d Bir yiyecek/içeceğin tüm ağaçta kaç adet sipariş verildiğini döndürme ve fiyat güncelleme 8](#_Toc92533885)

[1.d.1 Kaynak Kod 8](#_Toc92533886)

[1.d.2 Ekran görüntüleri 9](#_Toc92533887)

[2.a Hash Tablosuna Ekleme 9](#_Toc92533888)

[2.b Hash Tablosu Güncelleme 10](#_Toc92533889)

[3.a Heap Veri Yapısı Tasarlama 11,12,13](#_Toc92533890)

[3.b Max Heap düğüm yerleştirme 13,14](#_Toc92533891)

[3.c Heap bilgi çekme 14](#_Toc92533892)

[3.c.1 Kaynak Kod 14](#_Toc92533893)

[3.c.2 Ekran görüntüleri 14](#_Toc92533894)

[4.a Simple sorting veya Advanced sorting algoritması 14,15](#_Toc92533895)

[4.b Sıralama algoritması ile karşılaştırılması 15](#_Toc92533896)

[4.c Görselleştirme araçları 15](#_Toc92533897)

[Özdeğerlendirme Tablosu 16](#_Toc92533898)

ARAMA AĞAÇLARI, YIĞINLAR VE HASH TABLOSU: YEMEK DAĞITIM ŞİRKETİ

Visual Studio, 2019, C# used

## 1.a YemekSınıfı nesnesinin ve Yemek Sipariş Ağacının oluşturulması

class YemekSinifi

{

public string ad;

public int adet;

public double fiyat;

public YemekSinifi(string ad, int adet, int fiyat)

{

this.ad = ad;

this.adet = adet;

this.fiyat = fiyat;

}

public override string ToString()

{

return String.Format("[Yemek Adı: {0}, Adet : {1}, Fiyat: {2}]", ad, adet, fiyat);

}

}

class Mahalle

{

public string mahalleAdi;

public List<YemekSinifi[]> siparisListesi; // siparisListesinin herbir elemanı o mahalleye ait siparişleri tutan bir dizi, dizinin her elemanı da siparişi oluşturan yiyecek bilgilerini tutan YemekSinifi nesnesi

public Mahalle(string mahalleAdi)

{

this.mahalleAdi = mahalleAdi;

siparisListesi = new List<YemekSinifi[]>();

}

}

class TreeNode // Herbir düğüm mahalle nesnelerinden oluşur

{

public Mahalle mahalle; // data

public TreeNode leftChild;

public TreeNode rightChild;

public TreeNode(Mahalle mahalle)

{

this.mahalle = mahalle;

}

public void displayNode()

{

Console.WriteLine("Mahalle Adı: " + mahalle.mahalleAdi);

int sayac = 1;

foreach(YemekSinifi[] dizi in mahalle.siparisListesi)

{

Console.Write(sayac + ".Sipariş: ");

foreach(YemekSinifi yemek in dizi)

{

Console.Write(yemek.ToString() + " ");

}

Console.WriteLine();

sayac++;

}

}

}

class Tree

{

private TreeNode root;

int maxDepth;

public Tree()

{

root = null;

maxDepth = 0;

}

public TreeNode getRoot()

{

return root;

}

public void insert(Mahalle mahalle) // Önce geliyorsa leftChild, sonra geliyorsa rightChild

{

TreeNode newNode = new TreeNode(mahalle);

if (root == null)

root = newNode;

else

{

TreeNode current = root;

TreeNode parent;

while(true)

{

parent = current;

if(mahalle.mahalleAdi.CompareTo(current.mahalle.mahalleAdi) < 0) // Eklenen mahallenin adı şu an üzerinde bulunan düğümün mahalle adından önce geliyorsa:

{

current = current.leftChild;

if (current == null)

{

parent.leftChild = newNode;

return;

}

}

else // Sonra geliyorsa:

{

current = current.rightChild;

if (current == null)

{

parent.rightChild = newNode;

return;

}

}

}

}

}

… main

static Random random = new Random();

static string[] mahalleAdlari = { "Evka3", "Özkanlar", "Atatürk", "Erzene", "Kazımdirik" };

static string[] yiyecekListesi = { "Ayran", "Kola", "Simit", "İçli Köfte", "Hamburger", "Mantı", "Döner", "Börek", "Pilav", "Türlü", "Pizza", "Kızartma" };

static void Main(string[] args)

{

Tree agac = new Tree();

for(int i = 0; i < mahalleAdlari.Length; i++)

{

Mahalle mahalle = new Mahalle(mahalleAdlari[i]);

int siparisSay = random.Next(5, 10);

for(int j = 0; j < siparisSay; j++) // 5-10 arasında rastgele sayıda sipariş içeren sipariş listesi oluşturmak için:

{

int turSay = random.Next(3, 5); // Her bir siparişte de 3-5 arasında rastgele yiyecek/içecek türü bilgisi olsun

YemekSinifi[] siparisBilgileri = new YemekSinifi[turSay];

for(int k = 0; k < turSay; k++)

{

int rnd = random.Next(0, yiyecekListesi.Length);

siparisBilgileri[k] = new YemekSinifi(yiyecekListesi[rnd], random.Next(1, 8), 3 \* (rnd + 1));

}

mahalle.siparisListesi.Add(siparisBilgileri);

}

agac.insert(mahalle);

}

NOT: Yiyecek fiyatlarını belirlemek için 3\*(rnd + 1) gibi bir formülizasyon kullandım.

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

### 1.b.1 Kaynak Kod

// Agacın inOrder Dolasılması

public void inOrder(TreeNode localRoot, int depth)

{

if (localRoot != null)

{

depth++;

if (depth > maxDepth) // Maximum derinliği(ağacın derinliğini) bulma

maxDepth = depth;

inOrder(localRoot.leftChild,depth);

localRoot.displayNode();

inOrder(localRoot.rightChild,depth);

}

}

public void traverseAndFindDepth() // Ağacı inorder olarak dolaşır ve derinliği ekrana yazdırır

{

inOrder(root, -1);

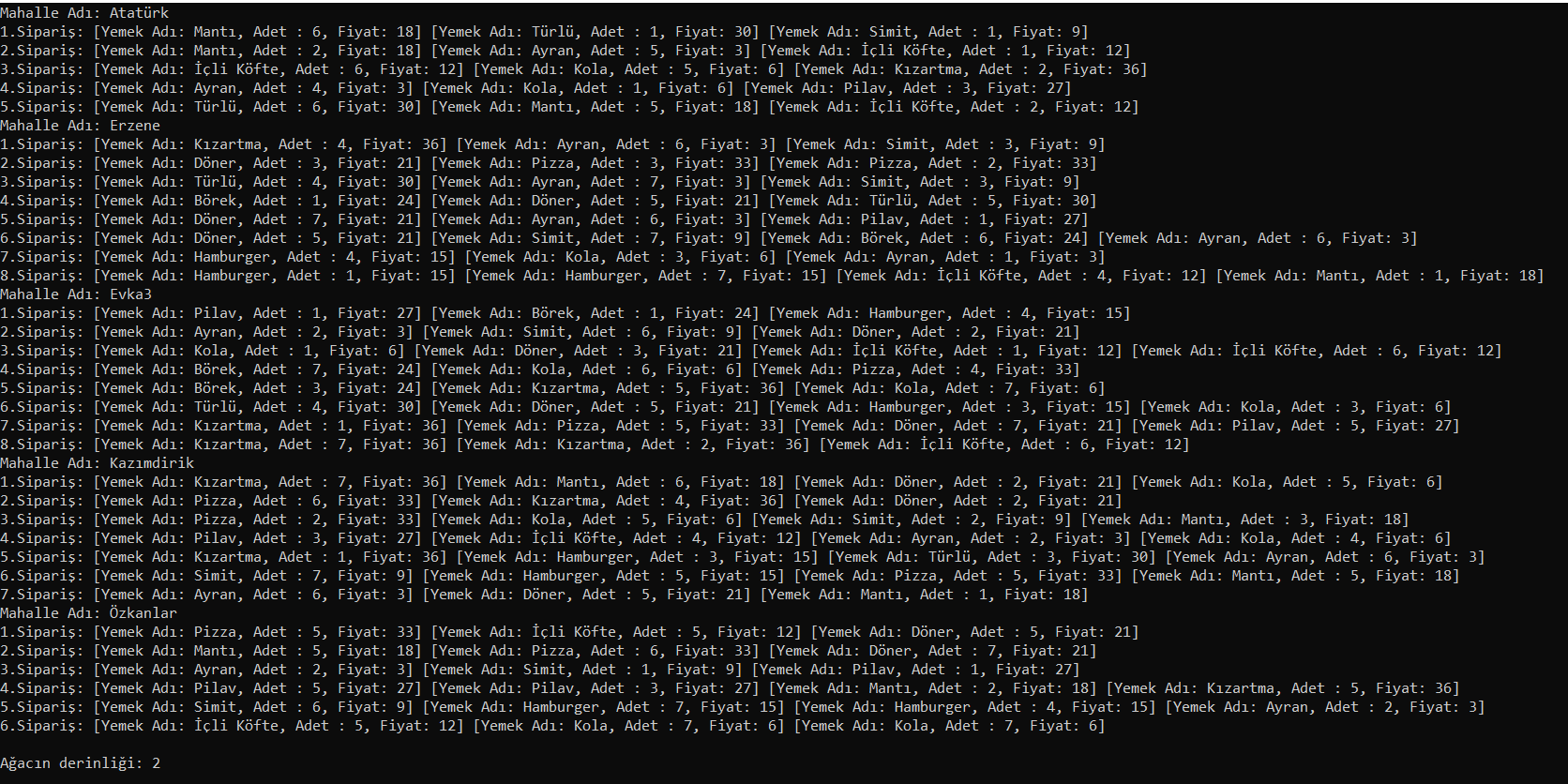
Console.WriteLine("\nAğacın derinliği: " + maxDepth + "\n");

}

… main

agac.traverseAndFindDepth();

### 1.b.2 Ekran görüntüleri



## 1.c 150 TL üstündeki siparişlerin bilgilerini listeleme

### 1.c.1 Kaynak Kod

public TreeNode find(string mahalleAdi) // Adı verilen mahalleyi ağaçta bulur

{

TreeNode current = root;

while(current != null)

{

if (mahalleAdi.CompareTo(current.mahalle.mahalleAdi) < 0) // Aranan mahalle current'taki mahalleden önce geliyorsa

current = current.leftChild;

else if (mahalleAdi.CompareTo(current.mahalle.mahalleAdi) > 0) // Sonra geliyorsa

current = current.rightChild;

else // mahalleAdi current'taki mahalle adına eşitse aranan node bulunmuş demektir

return current;

}

return null; // Aranan mahalle bulunamadı

}

public void listOrders(string mahalleAdi) // Adı verilen mahalledeki 150 TL üstündeki siparişlerin bilgilerini listeleyen metod

{

TreeNode node = find(mahalleAdi);

Console.WriteLine(mahalleAdi + " mahallesindeki 150 TL üstündeki siparişler: ");

foreach(YemekSinifi[] siparis in node.mahalle.siparisListesi)

{

double siparisToplam = 0;

foreach (YemekSinifi yemek in siparis)

{

siparisToplam += yemek.adet \* yemek.fiyat;

}

if(siparisToplam > 150)

{

foreach (YemekSinifi yemek in siparis)

Console.Write(yemek.ToString() + " ");

Console.WriteLine();

}

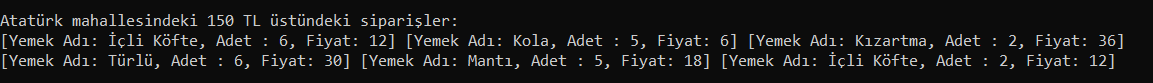
}

}

… main

agac.listOrders("Atatürk");

### 1.c.2 Ekran görüntüleri



## 1.d Bir yiyecek/içeceğin tüm ağaçta kaç adet sipariş verildiğini döndürme ve fiyat güncelleme

### 1.d.1 Kaynak Kod

public int countFood(TreeNode localRoot, string yiyecekAdi) // Ağacı kuyruk yapısı sayesinde iterative olarak dolaşır ve adı verilen bir yiyecek/içeceğin tüm ağaçta kaç adet sipariş verildiğini döndürür

{

int yiyecekSay = 0;

Queue<TreeNode> kuyruk = new Queue<TreeNode>();

kuyruk.Enqueue(localRoot);

while (kuyruk.Count > 0)

{

TreeNode current = kuyruk.Dequeue();

foreach (YemekSinifi[] siparis in current.mahalle.siparisListesi)

{

foreach (YemekSinifi yemek in siparis)

{

if (yemek.ad.Equals(yiyecekAdi))

{

yiyecekSay += yemek.adet;

yemek.fiyat = yemek.fiyat \* 0.9; // Yemeğin birim fiyatına %10 indirim uygulandı

}

}

}

if (current.leftChild != null)

kuyruk.Enqueue(current.leftChild);

if (current.rightChild != null)

kuyruk.Enqueue(current.rightChild);

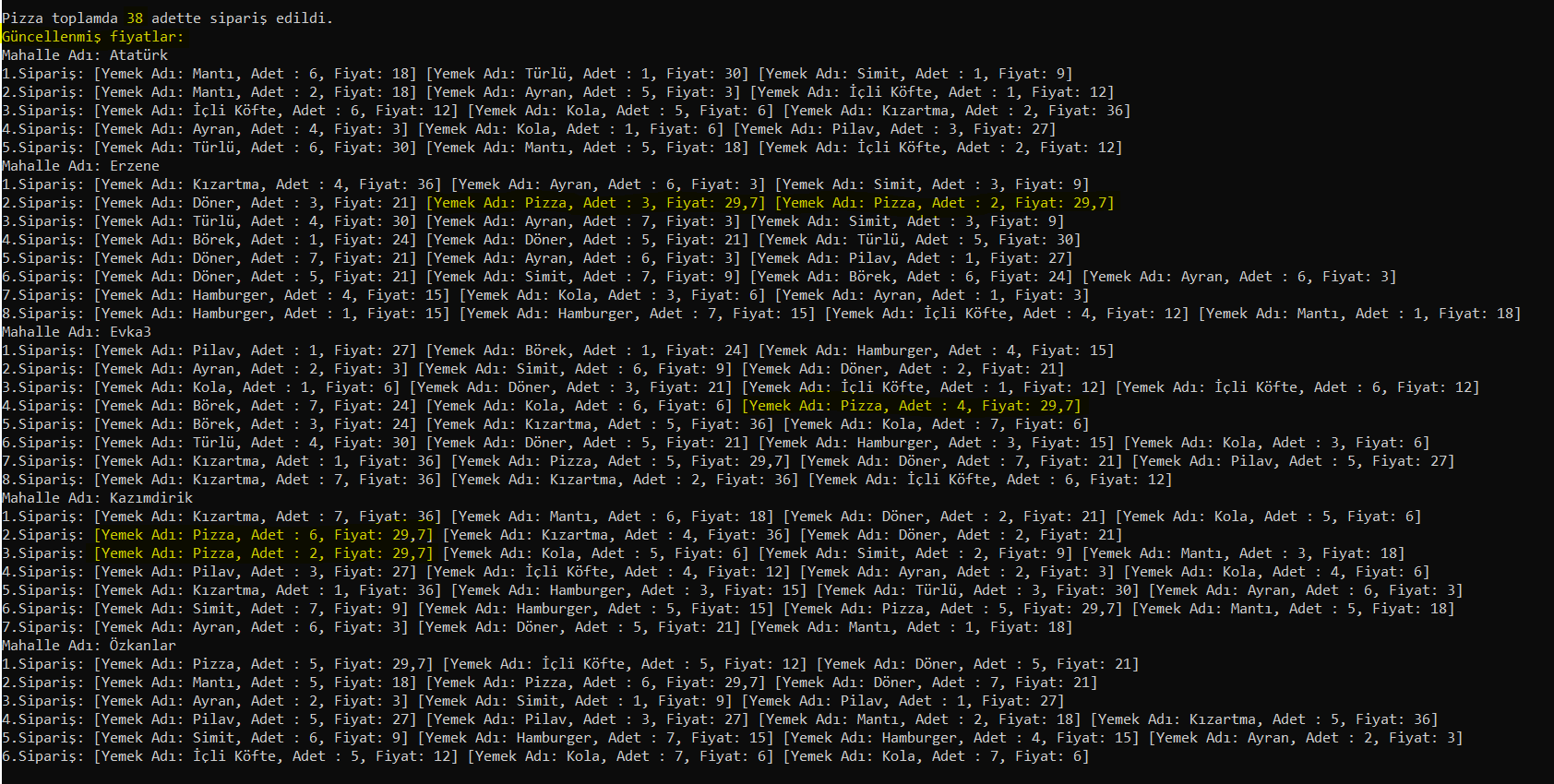
}

return yiyecekSay;

}

}

### 1.d.2 Ekran görüntüleri



## 2.a Hash Tablosuna Ekleme

static Hashtable bornova = new Hashtable(); // Yeni bir hashtable nesnesi oluşturma

static void Main(string[] args)

{

bornova.Add("Kızılay", 15795);

bornova.Add("Erzene", 35135);

bornova.Add("Kazımdirik", 33934);

bornova.Add("Yeşilova", 31008);

bornova.Add("Atatürk", 28912);

bornova.Add("İnönü", 25778);

bornova.Add("Evka3", 20445);

bornova.Add("Evka4", 14630);

bornova.Add("Mevlana", 25492);

bornova.Add("Doğanlar", 21461);

hashTableYazdir();

nufusGuncelle('K');

Console.WriteLine("\nK harfiyle başlayan mahallelerin nüfusları güncellendikten sonra: ");

hashTableYazdir();

}

## 2.b Hash Tablosu Güncelleme

static void nufusGuncelle(char basHarf) // Baş harfi verilen mahallelerin toplam nüfusuna 1 ekleyerek Hash Tablosunda güncelleyen metod

{

Hashtable temp = (Hashtable)bornova.Clone();

foreach (object mahalleAdi in temp.Keys)

{

if(Convert.ToString(mahalleAdi)[0].Equals(basHarf))

{

bornova[mahalleAdi] = Convert.ToInt32(bornova[mahalleAdi]) + 1;

}

}

}

static void hashTableYazdir()

{

foreach (object mahalle in bornova.Keys)

Console.WriteLine(mahalle + " - " + bornova[mahalle]);

}

## 3.a Heap Veri Yapısı Tasarlama

## public class Node {

## private int iData; // data item (key) -- nüfus

## private String mahalleAdi;

## 

## // -------------------------------------------------------------

## public Node(String mAdi,int key) // constructor

## { iData = key;

## mahalleAdi = mAdi;

## }

## // -------------------------------------------------------------

## public int getKey()

## { return iData; }

## // -------------------------------------------------------------

## public void setKey(int id)

## { iData = id; }

## 

## public String toString() {

## return "Mahalle Adı: " + mahalleAdi + ", Nüfus: " + iData;

## }

## }

## public class Heap {

## private Node[] heapArray;

## private int maxSize; // size of array

## private int currentSize; // number of nodes in array

## // -------------------------------------------------------------

## public Heap(int mx) // constructor

## {

## maxSize = mx;

## currentSize = 0;

## heapArray = new Node[maxSize]; // create array

## }

## // -------------------------------------------------------------

## public boolean isEmpty()

## { return currentSize==0; }

## // -------------------------------------------------------------

## public boolean insert(String mAdi, int key)

## {

## if(currentSize==maxSize)

## return false;

## Node newNode = new Node(mAdi, key);

## heapArray[currentSize] = newNode;

## trickleUp(currentSize++);

## return true;

## } // end insert()

## // -------------------------------------------------------------

## public void trickleUp(int index)

## {

## int parent = (index-1) / 2;

## Node bottom = heapArray[index];

## while( index > 0 &&

## heapArray[parent].getKey() < bottom.getKey() )

## {

## heapArray[index] = heapArray[parent]; // move it down

## index = parent;

## parent = (parent-1) / 2;

## } // end while

## heapArray[index] = bottom;

## } // end trickleUp()

## // -------------------------------------------------------------

## public Node remove() // delete item with max key

## { // (assumes non-empty list)

## Node root = heapArray[0];

## heapArray[0] = heapArray[--currentSize];

## trickleDown(0);

## return root;

## } // end remove()

## // -------------------------------------------------------------

## public void trickleDown(int index)

## {

## int largerChild;

## Node top = heapArray[index]; // save root

## while(index < currentSize/2) // while node has at

## { // least one child,

## int leftChild = 2\*index+1;

## int rightChild = leftChild+1;

## // find larger child

## if(rightChild < currentSize && // (rightChild exists?)

## heapArray[leftChild].getKey() <

## heapArray[rightChild].getKey())

## largerChild = rightChild;

## else

## largerChild = leftChild;

## // top >= largerChild?

## if( top.getKey() >= heapArray[largerChild].getKey() )

## break;

## // shift child up

## heapArray[index] = heapArray[largerChild];

## index = largerChild; // go down

## } // end while

## heapArray[index] = top; // root to index

## } // end trickleDown()

## }

## 3.b Max Heap düğüm yerleştirme

public class Test {

public static void main(String[] args) {

Heap maxHeap = new Heap(50);

// Bornovadaki 10 mahalleyi max heap'e yerleştirme:

maxHeap.insert("Kızılay", 15795);

maxHeap.insert("Erzene", 35135);

maxHeap.insert("Kazımdirik", 33934);

maxHeap.insert("Yeşilova", 31008);

maxHeap.insert("Atatürk", 28912);

maxHeap.insert("İnönü", 25778);

maxHeap.insert("Evka3", 20445);

maxHeap.insert("Evka4", 14630);

maxHeap.insert("Mevlana", 25492);

maxHeap.insert("Doğanlar", 21461);

## 3.c Heap bilgi çekme

### 3.c.1 Kaynak Kod

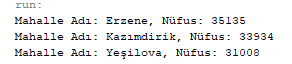
// Nüfusu en fazla olan 3 mahalleyi sıra ile Heap’ten çekme:

System.out.println(maxHeap.remove()); // toString otomatik olarak çağrılır

System.out.println(maxHeap.remove());

System.out.println(maxHeap.remove());

### 3.c.2 Ekran görüntüleri



## 4.a Simple sorting veya Advanced sorting algoritması

class InsertionSort

{

public void sort(int[] arr) // Insertion sortu kullanarak diziyi sıralar

{

int n = arr.Length;

for (int i = 1; i < n; ++i) // i, 1'den başlar ve her adımda bir key değeri belirlenir.

{

int key = arr[i];

int j = i - 1;

// Dizide 0'dan i-1'e kadar olan indisli elemanlar(key değerinin solunda kalanlar) eğer key değerinden büyükse bir sağa kaydırılır:

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j]; // Elemanları kaydırma

j = j - 1;

}

arr[j + 1] = key; // key değerini diziye yerleştirme

}

}

}

class Program

{

static void printArray(int[] arr)

{

int n = arr.Length;

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

Console.Write(arr[i] + " ");

Console.Write("\n");

}

public static void Main()

{

int[] arr = { 12, 11, 13, 5, 6, -1, 2, 3, 17 };

Console.Write("İnitial Array: ");

printArray(arr);

InsertionSort ob = new InsertionSort();

ob.sort(arr);

Console.Write("Sorted Array: ");

printArray(arr);

}

}

## 4.b Sıralama algoritması ile karşılaştırılması

Insertion Sort algoritmasının worst case ve average case’teki zaman karmaşıklığı O(n2)’dir. Best Case’te ise O(n)’dir.

Insertion Sort ile Selection Sort’u karşılaştıracağım: Selection Sort, Insertion Sort'tan daha az atama işlemi yaparken daha fazla karşılaştırma işlemi yapar. Bu nedenle Selection Sort büyük kayıtlardan oluşan az elemanlı veri grupları için (atamaların süresi çok fazla olmaz) ve karşılaştırmaların daha az yük getireceği basit anahtarlı durumlarda uygundur. Tam tersi için, insertion sort uygundur.

## 4.c Görselleştirme araçları

İlk etapta anlayamadığım algoritmalarda görselleştirmenin büyük yararını görüyorum. Görselleştirme araçları ile adım adım ilerlerken kodun üzerinden geçerek anlayabiliyorum. Veri Yapılarını öğrenme açısından Etkileşimli Görselleştirme Araçlarını videolardan daha yararlı buluyorum. Örnek olarak bu proje için sıralama algoritmalarında video çok hızlı bir şekilde geçerken görselleştirme aracından hem adım adım ilerletebildiğim için hem de değerlerin değişimini görüp sıralama hızını ayarlayabildiğim için daha çok verim aldım.

# Özdeğerlendirme Tablosu

**Özdeğerlendirme Tablosu**

|  |  |  |  |
| --- | --- | --- | --- |
| **Proje 3 Maddeleri** | **Not** | **Tahmini Not** | **Açıklama** |
| **1 a) Ağaç (Yemek Sipariş Ağacı Oluşturma)** | **10** | **10** | **Yapıldı** |
| **1 b) Derinlik Bulma, Ağacı Listeleme** | **10** | **10** | **Yapıldı** |
| **1 c) Arama ve Listeleme** | **10** | **10** | **Yapıldı** |
| **1 d) Ağaçta bir yemek türüne indirim yapma** | **10** | **10** | **Yapıldı** |
| **2) Hash Tablosu** | **15** | **15** | **Yapıldı** |
| **3) Yığın Ağacı (Heap)** | **15** | **15** | **Yapıldı** |
| **4) Sıralama Algoritmaları** | **20** | **20** | **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.**