

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

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

**2021–2022 FALL SEMESTER**

**PROJECT-2 REPORT**

**(List, Stack, Queue, PQ – Priority Queue Data Structures)**

**DELIVERY DATE**

04/01/2022

**PREPARED BY**

05200000096, Uğur Alp Balta

İçindekiler

[1.a Bileşik Veri Yapısı için Ön Çalışma 2](#_Toc91168778)

[1.b Bileşik Veri Yapısı Kodlama ve Çalıştırma 2](#_Toc91168779)

[1.b.1 Kaynak Kod 2](#_Toc91168780)

[1.b.2 Ekran görüntüleri 2](#_Toc91168781)

[1.b.3 Veri Yapıları ve Açıklama 2](#_Toc91168782)

[1.c Bileşik Veri Yapısı Bilgi Çıkarma 2](#_Toc91168783)

[1.c.1 Kaynak Kod 2](#_Toc91168784)

[1.c.2 Ekran görüntüleri 2](#_Toc91168785)

[2.a Yığıt 2](#_Toc91168786)

[2.a.1 Kaynak Kod 2](#_Toc91168787)

[2.a.2 Ekran görüntüleri 2](#_Toc91168788)

[2.b Kuyruk 2](#_Toc91168789)

[2.b.1 Kaynak Kod 2](#_Toc91168790)

[2.b.2 Ekran görüntüleri 2](#_Toc91168791)

[3.a Öncelikli Kuyruk Oluşturma 2](#_Toc91168792)

[3.a.1 Kaynak Kod 2](#_Toc91168793)

[3.a.2 Ekran görüntüleri 2](#_Toc91168794)

[3.b List ve Dizi altyapılarının karşılaştırılması 2](#_Toc91168795)

[4.a Öncelikli Kuyruk Güncelleme 3](#_Toc91168796)

[4.b Ortalama İşlem Tamamlama Süresi 3](#_Toc91168797)

[4.b.1 Kaynak Kod 3](#_Toc91168798)

[4.b.2 Ekran görüntüleri 3](#_Toc91168799)

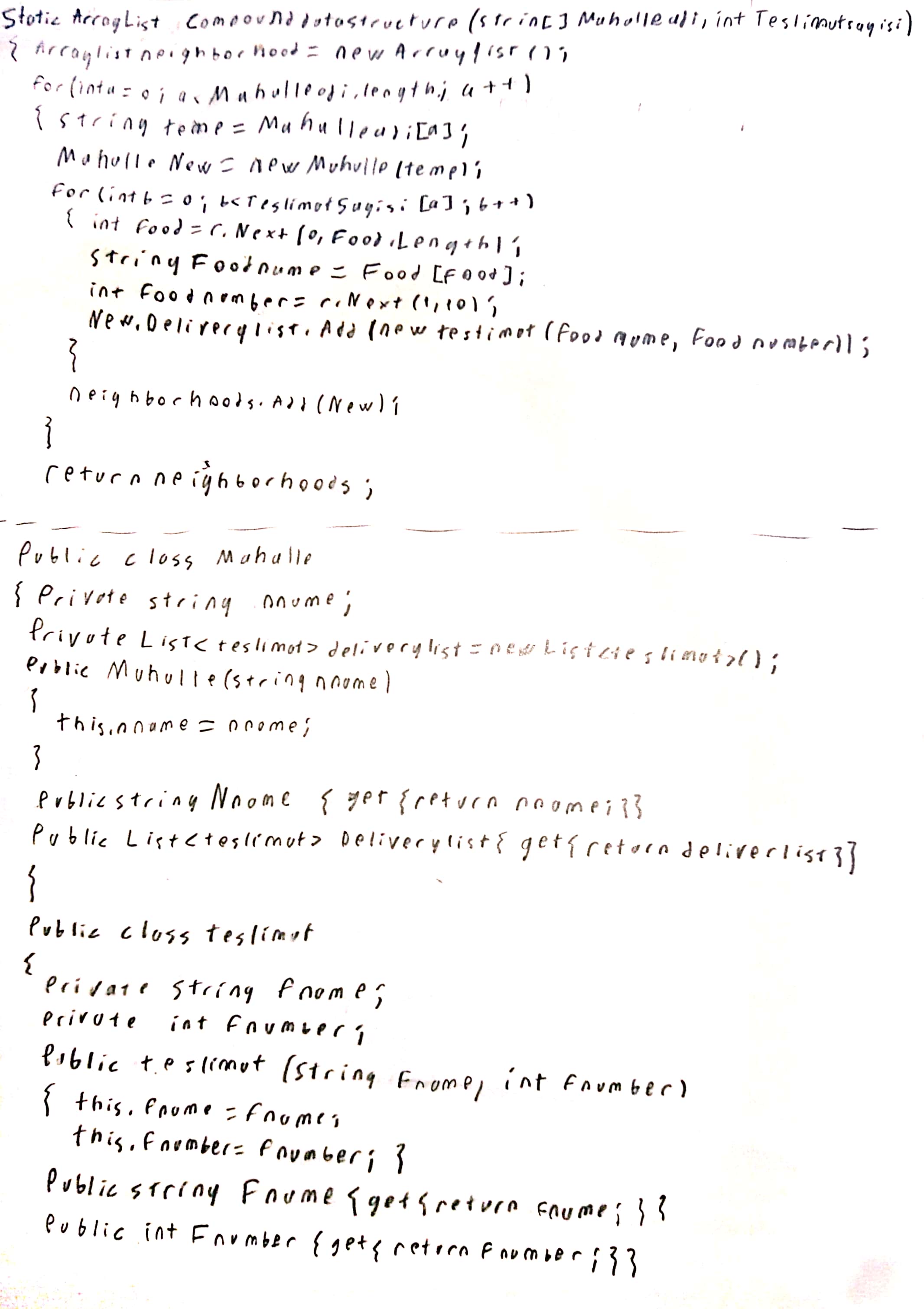
[4.c Öncelikli Kuyruk Tartışma 3](#_Toc91168800)

[5. Özdeğerlendirme Tablosu 3](#_Toc91168801)

LİSTE, YIĞIT, KUYRUK ve ÖNCELİKLİ KUYRUK VERİ YAPILARI

Visual Stusio 2022, C#

## 1.a Bileşik Veri Yapısı için Ön Çalışma



## 1.b Bileşik Veri Yapısı Kodlama ve Çalıştırma

### 1.b.1 Kaynak Kod

static Random r = new Random(); //Defines a random value.

static string[] Food = {"Pizza","Hamburger","Cheeseburger","Köfte","Tavuk şiş", "Kuzu şiş", "Pirzola", "Kanat Kova", "Kebap", "Lahmacun"}; // A random list of foods that could be used later. (Could be increased without tmepering with anything else)

static void Main(string[] args)

{

// Names of neighborhoods and orders of said neighborhoods they could be increased withoud tempering with anything else however they need to be the same size at all times.

string[] MahalleAdi { "Özkanlar", "Evka 3", "Atatürk", "Erzene", "Kazımdirik", "Mevlana", "Doğanlar", "Ergene"};

int[] TeslimatSayisi = { 5, 2, 7, 2, 7, 3, 0, 1 };

ArrayList neighborhoods = Compounddatastructure(MahalleAdi, TeslimatSayisi); // Arraylist gets defined

-------------------------------------------------------------------------------------------------------------------------

static ArrayList Compounddatastructure(string[] MahalleAdi, int[] TeslimatSayisi)

{

ArrayList neigborhoods = new ArrayList();

for (int a = 0; a < MahalleAdi.Length; a++)

{

string temp = MahalleAdi[a]; // Temporary string that holds current neighborhood name

Mahalle New = new Mahalle(temp); // Neighborhood gets created

for (int b = 0; b < TeslimatSayisi[a]; b++) // Inserts random food names and numbers in Neigborhood's Generic list.

{

int food = r.Next(0, Food.Length);

string foodname = Food[food];

int foodnumber = r.Next(1, 10);

New.Deliverylist.Add(new teslimat(foodname, foodnumber));

}

neigborhoods.Add(New); // New neighborhood gets added to the arraylist

}

return neigborhoods;

}

-------------------------------------------------------------------------------------------------------------------------

public class Mahalle

{

private string nname; // Name of the neighborhood

private List<teslimat> deliverylist = new List<teslimat>(); // List that holds orders of the said neighborhood

public Mahalle(string nname) //Constructor

{

this.nname = nname;

}

//Getters

public string Nname

{

get { return nname; }

}

public List<teslimat> Deliverylist

{

get { return deliverylist; }

}

}

public class teslimat

{

private string fname; // Name of the food

private int fnumber; // Number of said food

public teslimat(string fname, int fnumber) //Constructor

{

this.fname = fname;

this.fnumber = fnumber;

}

//Getters

public string Fname

{

get { return fname; }

}

public int Fnumber

{

get { return fnumber; }

}

}

}

### 1.b.2 Ekran görüntüleri

### metin içeren bir resim Açıklama otomatik olarak oluşturuldu

### 1.b.3 Veri Yapıları ve Açıklama

Random r: Bütün main için rasgele sayı üretmeye yardımcı olur.

String[] Food: Yemek türleri içeren dizi.

String[]: MahhalleAdi: Mahalle isimlerini içeren dizi.

Int TeslimaSayisi: Her bir mahalle için teslimat sayısını içeren dizi.

Not!: Programın doğru çalışabilmesi için TeslimatSayisi dizisi ve MahlleAdi dizisi aynı büyüklükte olmalıdır.

Arraylist neighborhoods: Mahalleleri alır.

String temp: a. indexteki mahalle adıdır.

Mahalle New = Mahalle tipinde yeni bir obje yaratılır.

Int food = Food dizisinden çekilmek için için o dizenin uzunluğunda rastgele değer üretilir.

String foodname = Food dizisindeki rastgele bir yemek adı.

Int foodnumber = 1 ila 10 arası rastgele bir sipariş adedi üretilir.

String name: Mahallenin adı

List<teslimat>: Bu mahalleye teslim edilecek teslimatları tutar.

String fname: Yemeğin adı

Int fnumber: Yemeğin adedi

Int orderSum: Bütün siparişlerin sayısını tutar.

### 1.c Bileşik Veri Yapısı Bilgi Çıkarma

### 1.c.1 Kaynak Kod

int ordersSum = 0; // Counter for number of total orders

//Prints the compound data structure

foreach (Mahalle a in neighborhoods)

{

Console.Write(a.Nname + ": ");

foreach (teslimat b in a.Deliverylist)

{

Console.Write("|" + b.Fname + ", " + b.Fnumber + "|" );

ordersSum++;

}

Console.WriteLine(" ");

}

// Prints the requirements of 1.c

Console.WriteLine("Number of neighborhoods: " + neighborhoods.Count);

### Console.WriteLine("Total amount of orders: " + ordersSum);

### 1.c.2 Ekran görüntüleri

1-b maddesi için eklenen kısım bu maddenin de gereksinimlerini içermektedir.

## 2.a Yığıt

### 2.a.1 Kaynak Kod

class Stack //Stack from the book changed to the Mahalle type

{

private int maxSize;

private Mahalle[] stackArray;

private int top;

public Stack(int max) // Constructor

{

maxSize = max;

stackArray = new Mahalle[maxSize];

top = -1;

}

public int Max { get { return maxSize; } } // For finding the number of neighborhoods easily

public void push(Mahalle j) //Adds a new element and increases the top

{ stackArray[++top] = j; }

public Mahalle pop() // Gets the element and decreases the top

{ return stackArray[top--]; }

public bool isEmpty() // true, if it is empty

{ return top == -1; }

### }

### 2.a.2 Ekran görüntüleri

## metin, iç mekan, ekran görüntüsü, kapat içeren bir resim Açıklama otomatik olarak oluşturuldu

## 2.b Kuyruk

### 2.b.1 Kaynak Kod

class Queue

{

private int size;

private Mahalle[] QueueArray;

private int head, tail;

private int NumberOfElements;

public Queue(int s) // Constructor

{

size = s;

QueueArray = new Mahalle[size];

head = 0; tail = -1; NumberOfElements = 0;

}

public void enque(Mahalle j) // Adds an element to the tail

{

if (tail == size - 1) // Possibility of a full circle

tail = -1;

QueueArray[++tail] = j; // Increases the tail and adds to the index that tail points

NumberOfElements++;

}

public Mahalle deque() // Removes an element from the head

{

Mahalle temp = QueueArray[head++]; // Gets the element and increases head

if (head == size) // Possibility of a full circle

head = 0;

NumberOfElements--;

return temp;

}

public bool isEmpty() // true, if it is empty

{

return (NumberOfElements == 0);

}

public int Size { get { return size; } } //For finding the number of neighborhoods easily

}

### 2.b.2 Ekran görüntüleri

metin, iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 3.a Öncelikli Kuyruk Oluşturma

### 3.a.1 Kaynak Kod

class PriorityQueue

{

private List<Mahalle> QueueList; // List that holds neighborhoods

public PriorityQueue() // Constructor

{

QueueList = new List<Mahalle>();

}

public void enque(Mahalle j) //Just adds the new neighborhood to the list.

{

QueueList.Add(j);

}

public Mahalle deque() // After finding the neighborhood which highest number of orders came from by comparing them to a temprary value it removes that neighborhood and removes it drom the list.

{

Mahalle temp = null;

int min = -1; // Since there could be 0 orders...

foreach (Mahalle m in QueueList)

{

if (m.Deliverylist.Count >= min)

{

min = m.Deliverylist.Count;

temp = m;

}

}

QueueList.Remove(temp);

return temp;

}

public bool isEmpty() // true, if it is empty

{

return (QueueList.Count == 0);

}

public int Elemnum() { return QueueList.Count; } //For finding the number of neighborhoods easily

### }3.a.2 Ekran görüntüleri

## metin içeren bir resim Açıklama otomatik olarak oluşturuldu3.b List ve Dizi altyapılarının karşılaştırılması

List yapısı sabit boyutlu olmadığından head,tail vb. gibi değişkenlere gerek kalmamaktadır. Bu sayede de özellikle çıkarma işlemi yapılırken head, tail değişkenlerinin güncellenmesine gerek olunmamaktadır. Aynı nedenden ötürü başa dönme durumu da gerçekleşmez. Ayrıca listenin kendi metodları sayesinde büyüklük bulma gibi işlemler için de fazladan değişkenlere gerek kalmaz.

## 4.a Öncelikli Kuyruk Güncelleme

public int deque() // After finding the neighborhood which highest number of orders came from by comparing them to a temprary value it removes that neighborhood and removes it drom the list.

{

int temp = 0;

int max = int.MaxValue;

foreach (int m in QueueList)

{

if (m <= max)

{

max = m;

temp = m;

}

}

QueueList.Remove(temp);

return temp;

## }

## 4.b Ortalama İşlem Tamamlama Süresi

### 4.b.1 Kaynak Kod

int processtime = 3; // Amount of time that cashier spends for scanning each product

int SumofTimes = 0; //Total time that pases in this particular checkout

int SumOfEachcustomertime = 0; //Sum of times each customer spends at checkout

int Numberofcustomers = CustomerTransactions.Size;

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

{

int customerCart = CustomerTransactions.deque(); //Number of items this customer have in their cart

int thisprocesstime = (customerCart \* processtime); //Seconds cashier spends scanning this customer's products

int timethiscustomerspends = thisprocesstime + SumofTimes; // Seconds this customer spends in checkout

SumofTimes+=thisprocesstime;

SumOfEachcustomertime += timethiscustomerspends;

Console.WriteLine((i + 1) + ". customer stayed: " + timethiscustomerspends + " seconds at chekout.");

}

Console.WriteLine("Average time a customer stays at checkout is: " + (Convert.ToDouble(SumOfEachcustomertime)/Convert.ToDouble(Numberofcustomers)) + (" seconds."));

}

### 4.b.2 Ekran görüntüleri

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu  
metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 4.c Öncelikli Kuyruk Tartışma

Priority Queue ödev sorusunda da belirtildiği üzere müşterilerin bekleme süresi azaltmaktadır ayrıca çıkarma işlemi yapılırken listenin bütün elemanlarını birbiriyle kıyasladığı için sabit bir noktadan çıkarım yapan Queue’dan zaman bakımından daha az etkin olmaktadır. Sıralama gerekli olmayan durumlarda (müşterilerin aynı sayıda ürün alması.) sıralama/karşılaştırma yaptığı için etkin olmaz.

# 5. Özdeğerlendirme Tablosu

**Özdeğerlendirme Tablosu**

|  |  |  |  |
| --- | --- | --- | --- |
| **Proje 2 Maddeleri** | **Puan** | **Tahmini Not** | **Açıklama** |
| **1 a) A4 Ön çalışma** | **20** | **18** | **Yapıldı.**  **İstenilen metod ve sınıflar a4 kağıdına yazılarak çalışmaya başlanıldı. Öğrencinin yazısı yeteri kadar okunaklı değil.** |
| **1 b) Kaynak kod, ekran görüntüsü, veri yapısının elemanlarının listelenmesi** | **20** | **20** | **Yapıldı.**  **İstenilen a4 kağıdının c# koduna dökülmüş versiyonu rapora taşındı. Veri yapısının elemanları çıktı olarak yazdırıldı, koddaki veri yapıları listekenip açıklandı.** |
| **1 c) Kaynak kodlar, Liste sayısı, veri yapısındaki toplam teslimat sayısı** | **5** | **5** | **Yapıldı. Kod rapora taşındı. 1-b için eklenen kısımda veri yapısından çekilmiş istenilen değerler görünmektedir.** |
| **2 a) Yığıt kaynak kod ve ekran görüntüleri** | **5** | **5** | **Yapıldı. Kodlar ve ekran görüntüleri rapora taşındı. Kodlar yazılırken slattlardan faydalanıldı.** |
| **2 b) Kuyruk kaynak kod ve ekran görüntüleri** | **5** | **5** | **Yapıldı. Kodlar ve ekran görüntüleri rapora taşındı. Kodlar yazılırken slaytlardan faydalanıldı** |
| **3 a) Öncelikli Kuyruk kod ve ekran görüntüleri** | **10** | **10** | **Yapıldı. Kodlar ve ekran görüntüleri rapora taşındı. Kod 2b’deki Queuenun içindeki dizinin List veri yapısı ile değiştirilmesi ve head,tail,s,ze gibi list yöntemi sayesinde gereksiz kılınan değişkenlerin silinmesiyle; ve dequeue metodunun List içindeki en büyük değerin bütün değerlerin birbirleri ile kıyaslanması sonucu bulunmasıyla gerçekleştirildi.** |
| **3 b) List ve Dizi altyapılarının karşılaştırılması** | **5** | **5** | **Yapıldı. Öncelikle aynı işemin Array ile yapıldığı düşünülerek karşılaşılacak potansiyal zorluklar gözden geçirildi. Rapor templateinde belirtilenin aksine proje kağıdında belirtilen şekilde generic List kullanıldığından karşılaştırma bu iki yapısında olacak şekilde başlık düzenlendi.** |
| **4) Kod, sonuçlar tablosu, ekran görüntüleri, karşılaştırma ve soruların cevapları.** | **20** | **19** | **Yapıldı. Gerekli veri yapıları daha önceki kısımlardan alınıp düzenlendi. Sorular anlaşıldığı üzere cevaplandı.** |
| **5) Özdeğerlendirme Tablosu** | **10** | **9** | **Yapıldı. Bazı formatlama hataları düzeltildi. İstanilen değerlerin çoğunun nasıl yapıldığı net olabilindikçe net bir şekilde açıklandı.** |
| **Toplam** | **100** | **96** |  |

**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.**