Proje Raporu 1:

```
public class Musteri
{
    10 references
    public int Musteri_No { get; set; }
    12 references
    public int Islem_Suresi { get; set; }

    6 references
    public int Islem_SuresiTers { get; set; }


}
```

Musteri sınıfında müşterilerin müşteri numarası ve işlem süresi gibi kendilerine özel bilgilerini tuttuk.

```
public interface IHesaplayabilir
{
    //aşağıdaki işlemleri iki kuyruk türü için
    //ortak metotları interfacede tutabiliriz.
    7 references
    int IslemTamamlama_suresi(int sure);
    4 references
    double Ortalama_sure();
}
```

Circular ve priority queue için bazı ortak metotlarımız var. Bu metotları IHesaplayabilir interface'inde tanımladık ve polymorphism yardımıyla queue'larda kullanabildik.

```
1 reference
public string Display()
{
    string temp = "";

    for(int i=0 ; i<20; i++)
    {
        temp += i + 1 + " Numaralı Müşterinin " + Environment.NewLine +
            "Müşteri no : " + ((Musteri)Queue[i]).Musteri_No.ToString() + Environment.NewLine +
            "İşlem süresi :" + ((Musteri)Queue[i]).Islem_Suresi.ToString()+ Environment.NewLine +
            "Toplam İşlem Süresi:" + IslemTamamlama_suresi(((Musteri)Queue[i]).Islem_Suresi) + Environment.NewLine + Environment.NewLine;
    }

    return temp;
}
public int IslemSuresi=0;
7 references
public int IslemTamamlama_suresi(int sure)
{
    IslemSuresi += sure;
    return IslemSuresi;
}
```

CircularQueue sınıfında işlem süresi dahil ne kadar kuyrukta kaldığını hesapladık ve tüm müşterilerin tüm bilgilerini yazdırmak için Display metodunu kullandık. İşlem tamamlama süresini IslemTamamlama_suresi metodu içerisinde tuttuk. Bu metotta tuttuğumuz sure değişkeni ayrı ayrı her müşteri için geçen işlem süresi dahil ne kadar süre kuyrukta kaldığını hesaplıyor. Display metodunda IslemTamamlama_suresi metoduna parametre olarak Islem_Suresini gönderiyoruz. Bu işlem süresi ise form'un içerinde her müşteriye rastgele atadığımız işlem süreleri. Böylece IslemTamamlama_suresi metodunda her müşteri için IslemSuresi hesaplanıyor. IslemSuresini daha sonra başka yerde kullanacağız kolay ulaşabilmek için public yaptık.

Priority queue ve büyükten küçüğe sıraladığımız priority queue için de aynı işlemleri yaptık.

```
4 references
public double Ortalama_sure()
{
    int IslemSuresi = 0;
    for (int i = 0; i < 20; i++)
    {
        DateTime baslangic = DateTime.Now;
        TimeSpan ts = DateTime.Now.Subtract(baslangic);
        IslemSuresi += ((Musteri)(Queue[i])).Islem_Suresi;
    }

    double Ortalama_Sure = IslemSuresi / 20;

    return Ortalama_Sure;
}
```

Ortalama_sure metodunda ortalama işlem süresini hesaplattık. Öncelikle her müşterinin işlem süresini daha önce tanımlamış olduğumuz IslemSuresi değişkeninde topladık ve Datetime – TimeSpan yardımıyla da öncelikle çalışma sürelerini hesaplattık ve her müşterinin işlem süresini birbiriyle toplayarak toplam süreye ulaştık. Daha sonra ise toplam süreyi müşteri sayısına bölerek ortalama süreyi bulduk.

Priority queue için de aynı işlemleri yaptık.

```
6 references
public void Insert(object o)
{
    if (count == size)
    {
        throw new Exception("Queue is full");
    }

    if (IsEmpty())
    {
        front++;
        Queue[front] = o;
        count++;
    }
    else
    {
        int i;
        for (i = count - 1; i >= 0; i--)
        {
            if (((Musteri)o).Islem_SuresiTers > ((Musteri)Queue[i]).Islem_SuresiTers)
                Queue[i + 1] = Queue[i];
            else
                break;
        }
        Queue[i + 1] = o;
        front++;
        count++;
    }
}
```

Kuyruğu işlem süresi en kısa olan müşteri daha önce hizmet alacak şekilde tasarlamak için priority queue'nun algoritmasında küçük bir değişiklik yapmamız yeterli oluyor. Bu nedenle Insert metodundaki küçüktür yerinde büyüktür yazıyoruz ve kuyruğumuzun öncelik sıralaması değişmiş oluyor.

```
int circularsure, prioritysure;

1 reference
public string FIFO_Bekleme()
{
    c.IslemSuresi = 0;
    string tmp = "";
    for(int i=0; i<20; i++)
    {
        circularsure = 0;
        pq.IslemSuresi = 0;
        circularsure = c.IslemTamamlama_suresi((((Musteri)c.Queue[i]).Islem_Suresi));
        for(int j=0; j<20; j++)
        {
            prioritysure = pq.IslemTamamlama_suresi((((Musteri)pq.Queue[j]).Islem_Suresi));
            if ((((Musteri)c.Queue[i]).Musteri_No) == (((Musteri)pq.Queue[j]).Musteri_No))
            {
                if (prioritysure < circularsure)
                {
                    float kazanc=(100*prioritysure)/circularsure;
                    tmp += (((Musteri)pq.Queue[j]).Musteri_No).ToString() + " numaralı muşteri daha az beklemiştir" +Environment.NewLine+
                        "Süreden Kazanç:"+(circularsure-prioritysure).ToString()+" / %"+kazanc.ToString()+Environment.NewLine+Environment.NewLine;
                }
                break;
            }
        }
    }
    return tmp;
}
```

Hangi müşterilerin FIFO kuyruğuna göre daha az beklediğini bulmak için hesaplamaları FIFO_Bekleme metodunda yaptık. Burada her iki kuyruk için de işlem sürelerine ihtiyacımız var. Bu nedenle işlem sürelerini circularsure ve prioritysure değişkenlerinde tuttuk. Bir tane circular queue bir tane de priority queue elemanları için iki tane döngü oluşturduk. En içteki döngüde priority ve circular queue'daki müşterilerin numaralarını karşılaştırdık. Eğer müşteri numaraları aynı değilse priority queue'nun o elemanı circular queue'nun karşılaştırdığımız elamanıyla aynı eleman değil demektir. Elemanı bulamadığımızdan circular queue elemanını sıradaki priority queue elemanıyla karşılaştırıyor. Eleman bulunduktan sonra içteki if koşuluna giriyor. Şu anda iki kuyruktaki elemanlar da aynı fakat kuyrukta daha az beklemiş olması için priority queue'da olan elemanın işlem süresi daha kısa olmalı. Bunu kontrol etmek için içerideki if koşuluna giriyor. Eğer priority'deki işlem süresi daha kısaysa koşuldaki işlemler gerçekleşiyor. Yani hangi müşterinin daha az beklemiş olduğunu ve zaman kazancını hesaplayıp yazdırıyoruz. Eğer priority'deki işlem süresi daha az değilse o zaman en içteki koşulu gerçekleştirmeden dış koşuldan çıkar ve sonraki priority elemanına geçer. Priority queue'nun bütün elemanlarının içerisinden circular queue'nun o elemanı bulunup süre karşılaştırılması yapıldıktan sonra circular queue'da bulunan diğer elemanları da priority queue da bulmak için içteki döngüden çıkıyor ve bir sonraki circular queue elemanını alarak tekrar priority queue döngüsünde giriyor. Bu işlem circular queue'nun bütün elemanları için yapılır.

Büyükten küçüğe sıraladığımız priority queue için de aynı işlemleri FIFO_BeklemeTers adlı metotta yaptık.

```
private void Form1_Load(object sender, EventArgs e)
{   //rastgele işlem süresi üretiyoruz
    Random r = new Random();
    for (int i = 0; i < 20; i++)
    {
        Musteri mus = new Musteri();
        mus.Musteri_No = i + 1;
        mus.Islem_SuresiTers =mus.Islem_Suresi = r.Next(60, 600);
        c.Insert(mus);
        pq.Insert(mus);
        tp.Insert(mus);
    }
}
```

Rastgele işlem süresi üretiyoruz ve her kuyruğa 20 müşteri gönderiyoruz.