RAPOR BAŞLIĞI

HAZIRLAYAN

ADI SOYADI : RABİA ŞENLİK

ÖĞRENCİ NUMARASI : 190303027

TESLİM TARİHİ : 20/11/2022

DERS ADI : ALGORİTMA TASARIMI

DERS YÜRÜTÜCÜSÜ : IŞIL KARABEY AKSAKALLI

Rapor/Ödev Başlığı :
Hazırlayanın Adı Soyadı :

BÖLÜM I

1. AYNI İŞİ YAPAN FARKLI ALGORİTMALARIN KARŞILAŞTIRILMASI

a) Bu bölümde insertion sort ve Merge sort algoritmalarını kullanarak klavyeden dizi boyutu alınan ve bu boyuta göre elemanları kullanıcı tarafından girilebilen bir diziyi küçükten büyüğe doğru sıralayan java kodunu yazacağız.

İlk olarak aşağıdaki gibi insertion sort classımızı oluşturacağız.

```
1   package main;
2   public class InsertionSort {
3
4       /*sort array  kullanarak sıralama işlevi*/
5       void sort(int arr[]) {
6           int n = arr.length;
7           for (int i = 1; i < n; ++i) {
8               int key = arr[i];
9               int j = i - 1;
10
11              while (j >= 0 && arr[j] > key) {
12                  arr[j + 1] = arr[j];
13                  j = j - 1;
14              }
15              arr[j + 1] = key;
16          }
17      }
18
19      //n boyutlu dizimizi sıralama işlemini yapıyoruz.
20      static void printArray(int arr[]) {
21          int n = arr.length;
22          for (int i = 0; i < n; ++i) {
23              System.out.print(arr[i] + " ");
24          }
25
26          System.out.println();
27      }
28  }
29
```

Daha sonra aşağıdaki gibi merge sort sınıfımızı oluşturacağız.

```java
package main;

public class MergeSort {

    void merge(int arr[], int l, int m, int r) {
        // Birleştirilecek iki alt dizinin boyutunu bulun
        int n1 = m - l + 1;
        int n2 = r - m;

      //Geçici diziler oluşturuyorum
        int L[] = new int[n1];
        int R[] = new int[n2];

        //sağ(n1) ve sol(n2) olarak iki tane geçici arraye verileri atıyorum
        for (int i = 0; i < n1; ++i) {
            L[i] = arr[l + i];
        }
        for (int j = 0; j < n2; ++j) {
            R[j] = arr[m + 1 + j];
        }

      //Geçici dizileri birleştirme işlemi

        int i = 0;//ilk subarrayin indexi
        int j = 0;// ikinci subarrayin indexi

        // Birleştirilmiş alt dizi dizisinin ilk dizini
        int k = l;//merge edilmiş subarrayın indexi
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
            } else {
                arr[k] = R[j];
                j++;
            }
            k++;
        }
        // L[] öğesinin  öğelerini kopyalama
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }

        //R[]'nin öğelerini kopyalama
        while (j < n2) {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    void sort(int arr[], int l, int r) {
        if (l < r) {
            // orta noktayı bul
            int m = l + (r - l) / 2;

            // Birinci ve ikinci yarıyı sırala
            sort(arr, l, m);
            sort(arr, m + 1, r);

            // Sıralanan yarımları birleştir
            merge(arr, l, m, r);
        }
    }

    // n boyutunda bir diziyi yazdırma işlemi
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}
```

Daha sonra da bu merge sort ve insertion sort ile işlemlerimizi yapabilmek için main sınıfımızda her iki sınıftan da birer obje tanımlayıp , bu objede tanımladığımız değişken ismini kullanarak sıralama işlemimizi yapıyoruz.

```java
package main;

import java.util.Scanner;
import static main.MergeSort.printArray;

public class Main {

    public static void main(String[] args) {
        //Bu bölümde kullanıcıdan dizimizin boyutunu ve bu boyuttaki diziyi oluşturacak dizi elamanlarımızı alıyoruz.
        Scanner sn = new Scanner(System.in);
        System.out.print("sıralamak istediğiniz dizinin boyutunu giriniz:");
        int arraySize = sn.nextInt();
         //Bu elamanları arr dizimizin içine atıyoruz
        int arr[] = new int[arraySize];
        for (int i = 0; i < arraySize; i++) {
            arr[i] = sn.nextInt();
        }
        //ve bu diziyi yazdırma işlemi yapıyoruz.
        System.out.println("kullanıcıdan aldığımız dizimiz:");
        printArray(arr);

        //merge sort ile sıralama işlemi
        System.out.println("merge sort ile sıralanmış hali");
        MergeSort ms = new MergeSort();//mergesort clasımızdan obje oluşturduk
        ms.sort(arr, 0, arr.length - 1);
        printArray(arr);//dizimizin merge sort ile sıralanmış halini bastırıyoruz

        //insertion sort ile sıralama işlemi
        System.out.println("************************");
        System.out.println("insertion sort ile sıralanmış hali:");
        InsertionSort ist = new InsertionSort();//insertionsort clasımızdan obje oluşturduk
        ist.sort(arr);//dizimizin insertion sort ile sıralanmış halini bastırıyoruz
        printArray(arr);

        //hazır olarak verilen 10 elemanlı dizi üzerinde yapılacak işlemler
        System.out.println("verilen 10 elemanlı dizinin insertion sort ve merge sort ile sıralanmış halleri");
        System.out.println("************************");
        int arr1[] = {15, 20, 3, 56, 76, 34, 2, 13, 65, 89, 12, 126};
        int arr2[] = {15, 20, 3, 56, 76, 34, 2, 13, 65, 89, 12, 126};
        System.out.println("merge sort ile sıralanmış hali");
        MergeSort ms1 = new MergeSort();
        ms1.sort(arr1, 0, arr1.length - 1);
        printArray(arr1);

        System.out.println("insertion sort ile sıralanmış hali:");

        InsertionSort ist1 = new InsertionSort();
        ist1.sort(arr2);

        printArray(arr2);
    }
}
```

Ve son olarak da kodumuzu çalıştırdığımızda aşağıdaki gibi dizi boyutumuzu ve dizi elemanlarımızı alıp hangi algoritma ile nasıl sıralandığını gösteren çıktımızı veriyor.

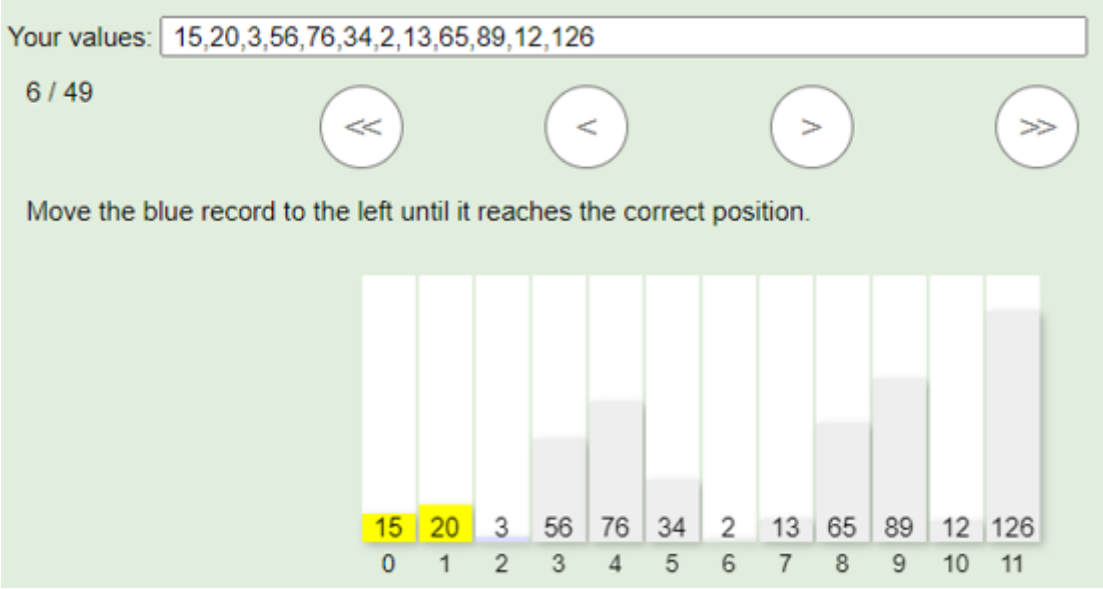
```
Output - Main (run)
run:
sıralamak istediğiniz dizinin boyutunu giriniz:4
98
87
23
0
kullanıcıdan aldığımız dizimiz:
98 87 23 0
merge sort ile sıralanmış hali
0 23 87 98
************************
insertion sort ile sıralanmış hali:
0 23 87 98
verilen 10 elemanlı dizinin insertion sort ve merge sort ile sıralanmış halleri
**********************
merge sort ile sıralanmış hali
2 3 12 13 15 20 34 56 65 76 89 126
insertion sort ile sıralanmış hali:
2 3 12 13 15 20 34 56 65 76 89 126
BUILD SUCCESSFUL (total time: 13 seconds)
```

b) Bu bölümde de a) maddesinde yazdığımız kodun main class içinde 10 elemanlı bir diziyi insertion sort ve merge sort algoritması ile sıraladık.

```
35          //hazır olarak verilen 10 elemanlı dizi üzerinde yapılacak işlemler
36          System.out.println("verilen 10 elemanlı dizinin insertion sort ve merge sort ile sıralanmış halleri");
37          System.out.println("**********************");
38          int arr1[] = {15, 20, 3, 56, 76, 34, 2, 13, 65, 89, 12, 126};
39          int arr2[] = {15, 20, 3, 56, 76, 34, 2, 13, 65, 89, 12, 126};
40          System.out.println("merge sort ile sıralanmış hali");
41          MergeSort ms1 = new MergeSort();
42          ms1.sort(arr1, 0, arr1.length - 1);
43          printArray(arr1);
44
45          System.out.println("insertion sort ile sıralanmış hali:");
46
47          InsertionSort ist1 = new InsertionSort();
48          ist1.sort(arr2);
49
50          printArray(arr2);
51      }
52  }
53
```

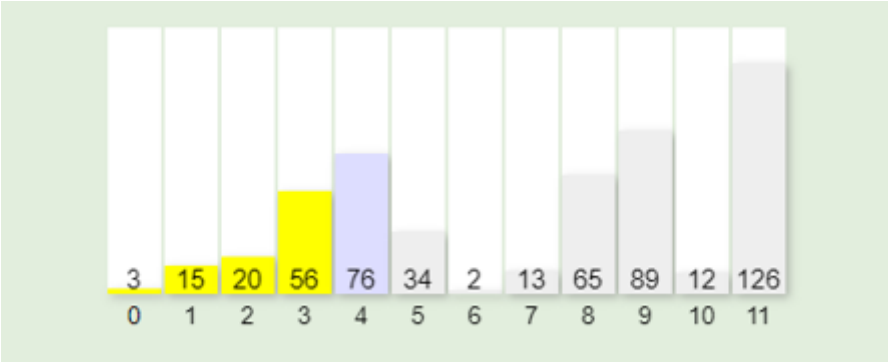
Yukarıdaki gibi dizimizi main içinde bu şekilde tanımladık ve sıraladık.

Şimdi de bu dizinin sıralanma olayının nasıl yapıldığını daha iyi anlamamız için insertion sort ve merge sort için simulation ile adım adım tüm basamakları göstereceğim.İlk olarak insertion sortta sıralama olayımız nasıl gerçekleşiyormuş ona bakalım.



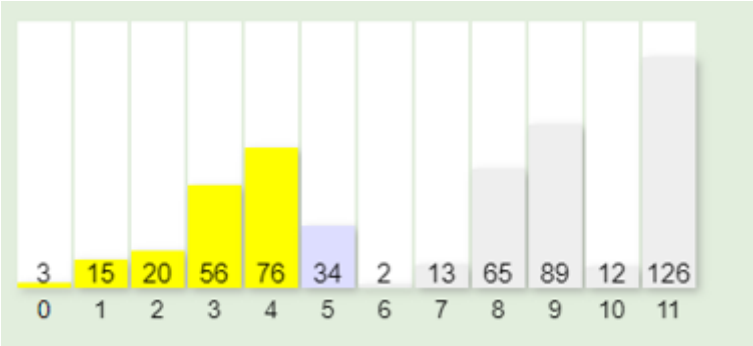
(1)

Bu şekilde dizimizi yazdık ve şimdi sırasıyla kontrol edeceğiz bir sonraki elemandan büyük mü küçük mü diye.15 <20 o halde bir sorun yok.20 ile 3 ‘ü karşılaştıracağız.20>3 o halde 3 20 ile yer değişecek. Ama 3 <15 o halde 3 ,15 ile de yer değişip en başa gelecek.



(2)

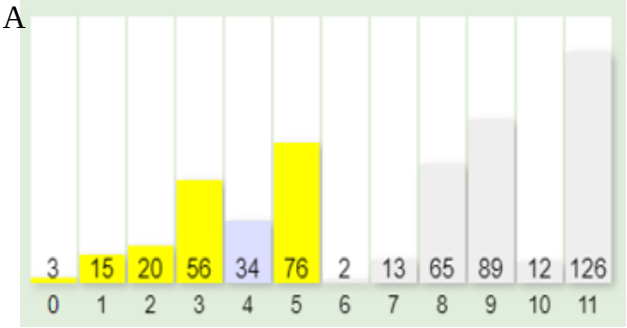
Şimdi de 56 ya bakacağız. 56<76 o halde sorun yok.ilerleyeceğiz.



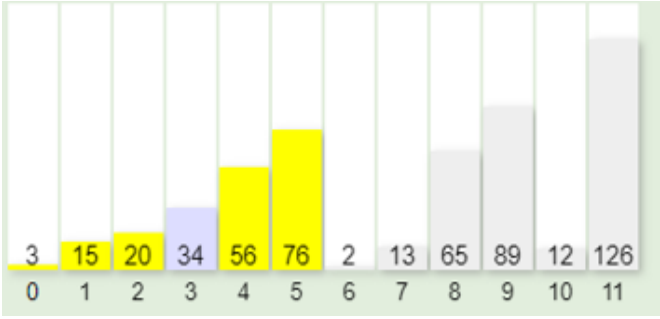
(3)

Şimdi de 76 ile 34 ü kıyaslayacağız.76>34 o halde yer değişmeleri gerek .şekil (4). Ancak 56>34 olduğu için 34 ‘ün 56 ile de yer değişmesi gerekiyor.şekil(5).

A

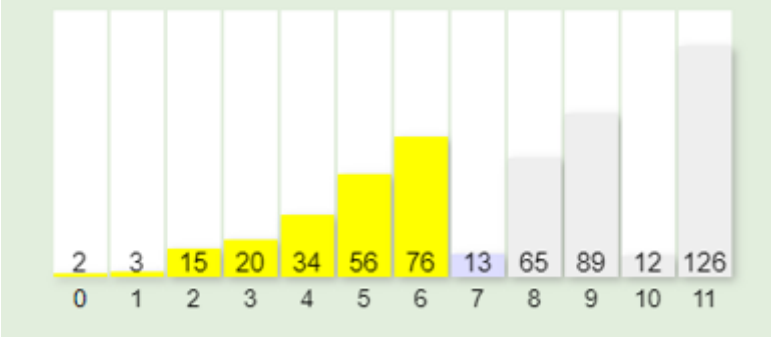


(4)



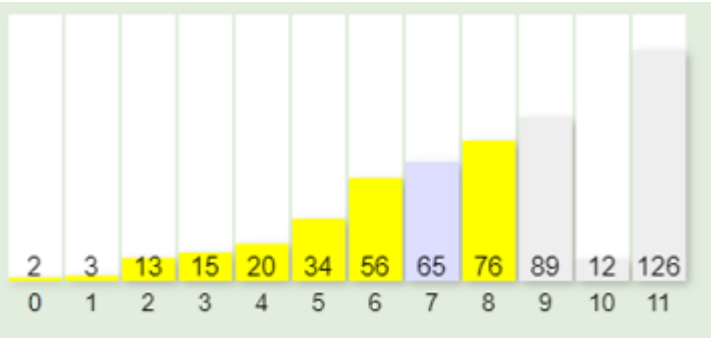
(5)

Daha sonra 76 ile 2’yi karşılaştırıyoruz. 2<76<56<34<20<15<3 olduğu için dizinin en başına geliyoruz.



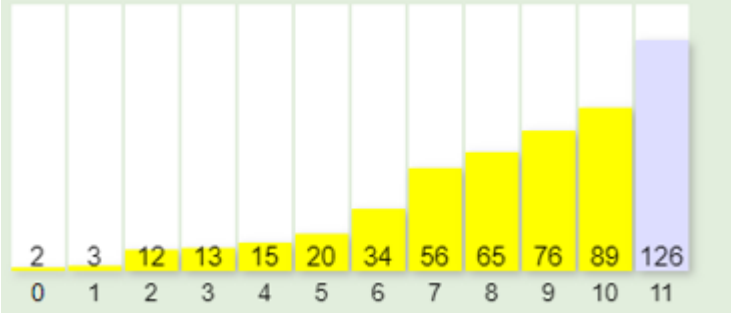
(6)

Daha sonra 76 ile 13'ü kıyaslıyoruz.13<76<56<34<20<15 olduğu için 3 ile 15 arasına geliyor.



(7)

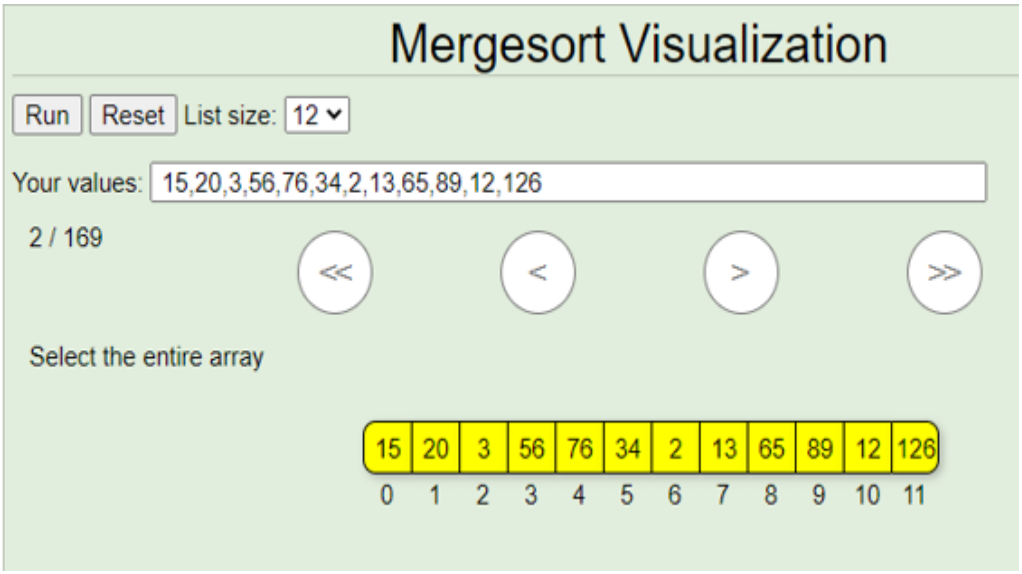
Sonra 76 ile 89 'u kıyaslıyoruz.76<89 o halde yer değişmemize gerek yok. 89 ile 12'yi kıyaslıyoruz.12<89<76<65<56<34<20<15<13 olduğu için 3 ile 13 arasına yerleştiriyoruz 12'yi.



(8)

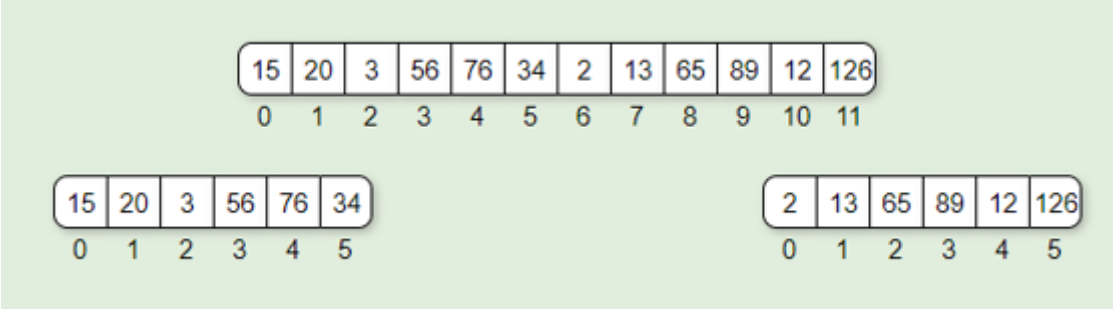
Ve en son da 89 ile 126'yı kıyaslıyoruz.89<126.O halde değişmemize gerek yok. Öyleyse dizimizin son hali şekil 8 deki gibi olur.

Şimdi de bu dizimizin merge sort ile nasıl sıralandığına bakalım. Dizimiz şekildeki gibi karışık dizilmiştir. Biz bunu rastgele ikiye ayırıyoruz.



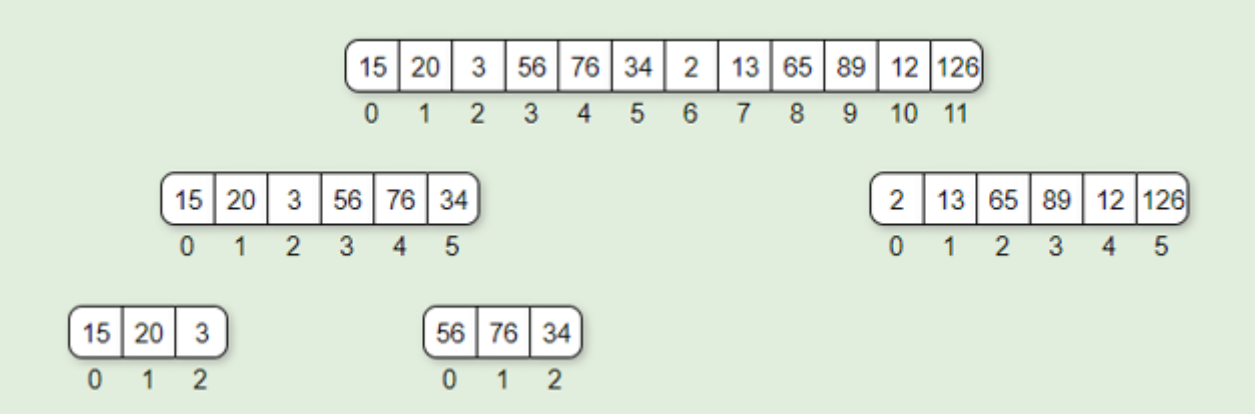
(1)

Dizimizi şekil2 deki gibi sağ ve sol olmak üzere ikiye ayırdık.



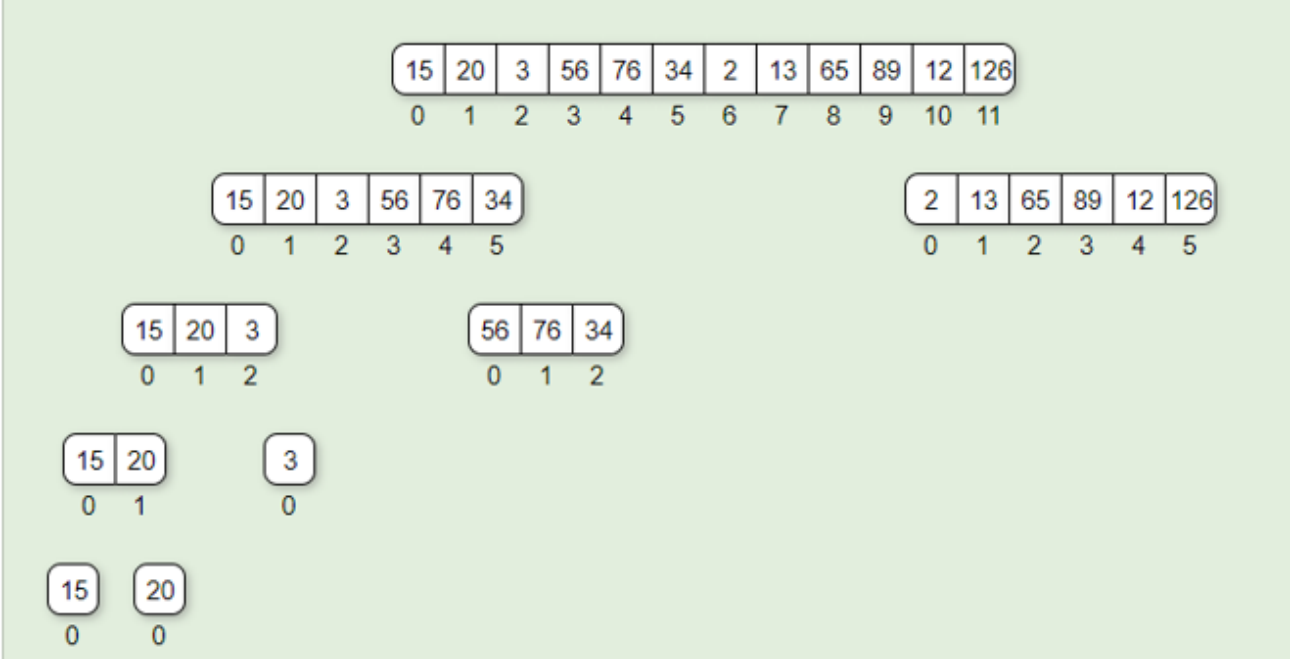
(2)

Sonra da sol kısımdaki dizimizi ikiye ayırdık şekil3 deki gibi.



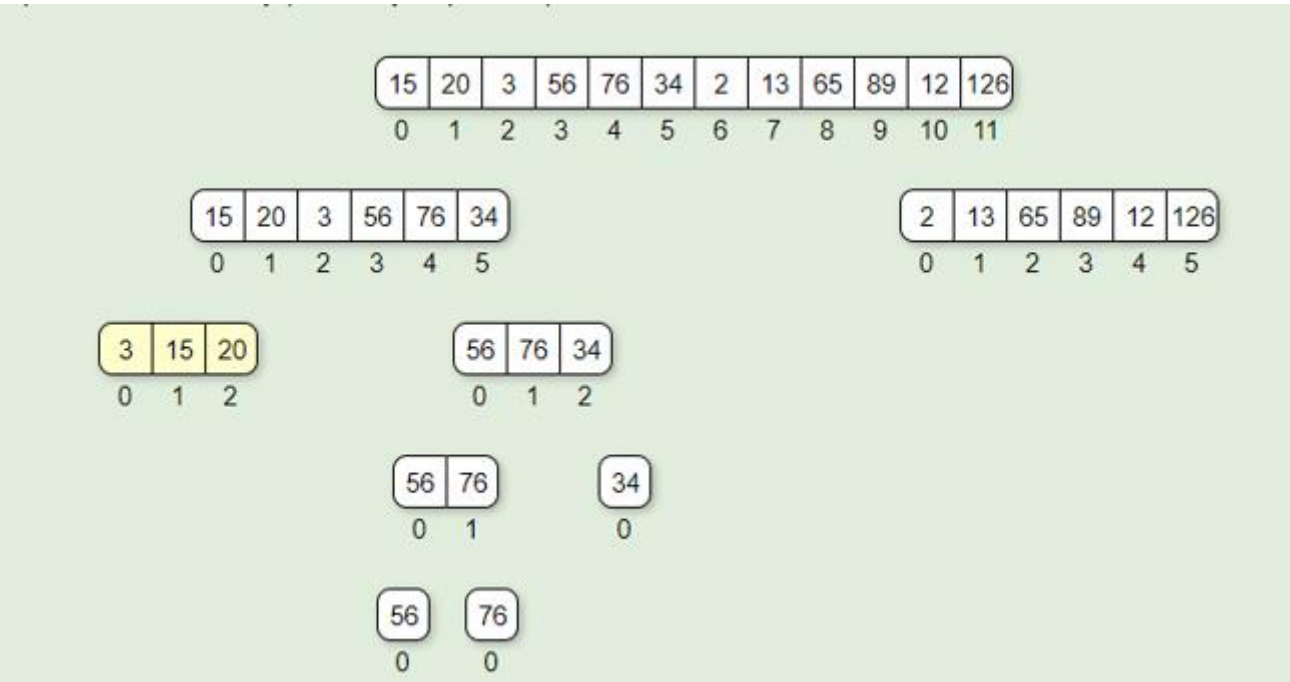
(3)

Sonrada ikiye ayrılan bu iki parçayı her bir eleman tek tek gözükene kadar bölüyoruz.



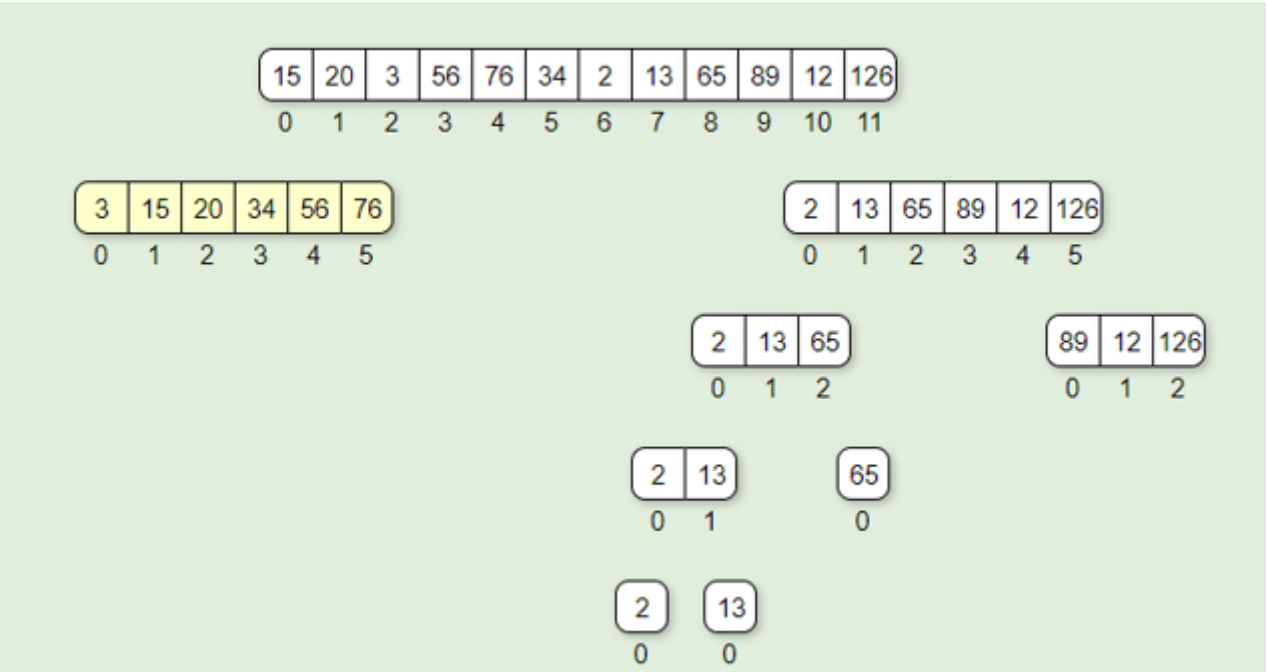
(4)

Daha sonrada da sağ taraftaki 56-76-34 dizimizi tek eleman kalana kadar ayırma işlemi yapıyoruz.

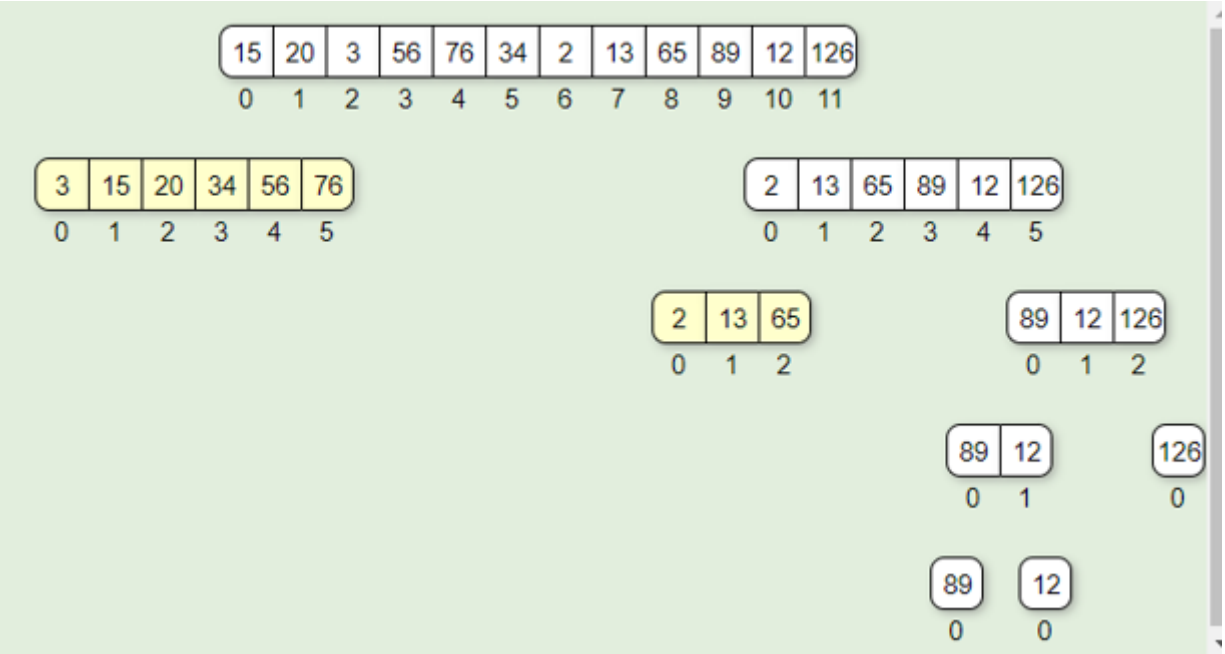


(5)

Daha sonra en sağdaki 2-13-65-89-12-126 dizimizi soldaki dizimiz gibi en altta birer eleman kalana kadar parçalıyoruz.

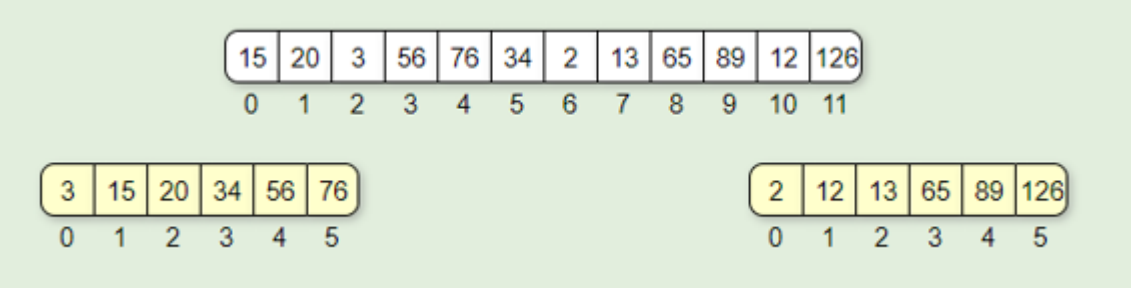


(6)



(7)

Son olarak da parçalanan bu dizimiz en alttan küçükten büyüğe doğru sıralayarak birleştirme işlemi yapıyoruz .Ve dizimizin sıralanmış hali şekil8 deki gibi oluyor.



(8)

Şimdi de algoritmalarımızın s**özde** kodu **üzerinden** en k**ötü,** en iyi ve ortalama durum analizlerini yapalım.

İlk olarak merge sortun kaba kodu üzerinden yorumlama yapalım.

```
MERGE_SORT(arr, beg, end)

if beg < end
set mid = (beg + end)/2
MERGE_SORT(arr, beg, mid)
MERGE_SORT(arr, mid + 1, end)
MERGE (arr, beg, mid, end)
end of if

END MERGE_SORT
```

Yukarıdaki gibi kaba kodu verilen merge sortun;

1-En İyi Durum Karmaşıklığı: Sıralama gerekli olmadığında, yani dizi zaten sıralandığında ortaya çıkar. Birleştirme sıralamasının en iyi durum zaman karmaşıklığı **O(n*logn)** şeklindedir.

2-Ortalama Vaka Karmaşıklığı: Dizi öğeleri düzgün bir şekilde artan ve düzgün bir şekilde azalan karışık sırada olduğunda ortaya çıkar. Birleştirme sıralamasının ortalama vaka süresi karmaşıklığı **O(n*logn)** şeklindedir.

3-En Kötü Durum Karmaşıklığı: Dizi öğelerinin ters sırada sıralanması gerektiğinde oluşur. Bu, dizi öğelerini artan düzende sıralamanız gerektiğini, ancak öğelerinin azalan düzende olduğunu varsayalım. Birleştirme sıralamasının en kötü zaman karmaşıklığı **O(n*logn)** şeklindedir.

Şimdi de insertion sortun kaba kodu üzerinden bu yorumları yapalım.

```
INSERTION-SORT(A)

for i=2 to A.length
    key = A[i]
    j = i - 1
    while j > 0 and A[j] > key
        A[j+1] = A[j]
        j = j - 1
    A[j + 1] = key
```

Yukarıdaki kaba kodu verilen insertion sortun karmaşıklık yorumlaını yapalım:

1-En İyi Durum Karmaşıklığı: Sıralama gerekli olmadığında, yani dizi zaten sıralandığında ortaya çıkar. Eklemeli sıralamanın en iyi durum zaman karmaşıklığı **O(n)** şeklindedir.

2-Ortalama Vaka Karmaşıklığı: Dizi öğeleri düzgün bir şekilde artan ve düzgün bir şekilde azalan karışık sırada olduğunda ortaya çıkar. Ekleme sıralamasının ortalama durum süresi karmaşıklığı **$O(n^2)$** şeklindedir .

3-En Kötü Durum Karmaşıklığı: Dizi öğelerinin ters sırada sıralanması gerektiğinde oluşur. Bu, dizi öğelerini artan düzende sıralamanız gerektiğini, ancak öğelerinin azalan düzende olduğunu varsayalım. Ekleme sıralamasının en kötü durum zaman karmaşıklığı **$O(n^2)$** şeklindedir .

BÖLÜM 2

2. ÇALIŞMA ZAMANI ANALİZİ

a) Bu bölümümüzde de aşağıda verilen algoritmayı adım adım analiz ederek toplam maliyetimizi c ve n türünden belirteceğiz.Son olarak da Big-O notasyonu cinsinden karmaşıklığını belirteceğiz.

```
                          Unit Cost      Times
j:=n                        c1
while j >=1 do              c2
begin
  i:=j                      c3
  while i>=1 do             c4
  begin
    x:=x + 1                c5
    i:= floor (i/2)         c6
  end
j:= floor (j/2)             c7
end
```

Bu kodumuzun toplam maliyetini şu şekilde hesaplarız:

Toplam maliyet=(c1*1)+(c2*logn)+(c3*logn)+(c4*lognlogn)+(c5*logn)+(c6*logn) olur. Şimdi bu kodun zaman karmaşıklığı ise buradaki en büyük katsayı değeri olur.Yani bizim zaman karmaşıklığımız O(log^2n).Bunun nedeni şudur:

Bizim için maliyet oluşturan şeyler döngülerdir.Sıradan bir satırın maliyeti O(1) dir.

```
1    x = n
2    while ( x > 0 ) {
3        x = x / 2
4    }
```

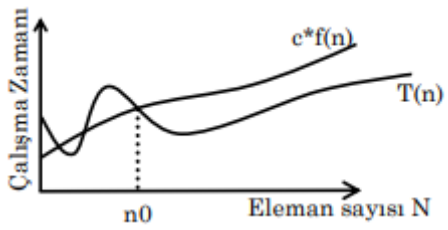
.

Şöyle bir kod parçasının karmaşıklığı ise O(logn)dir. Dikkat edecek olursak bu kod bizim kaba kodumuza çok benzemektedir .İç içe 2 tane bu koddan var. Bu da (lognlogn) demektir.

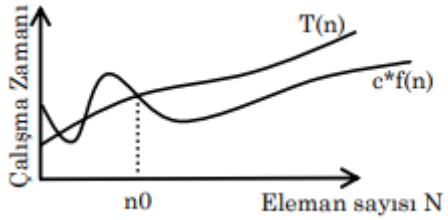
b) Bu bölümde de Big-O tanımını kullanarak aşağıdaki ifadeleri açıklayıp ,bulduğumuz c ve n sabitlerini açıklayacağız.

Bu örnekleri açıklamadan önce konuya bir değinelim.Bizim üç tane asimptotik notasyonumuz vardır.

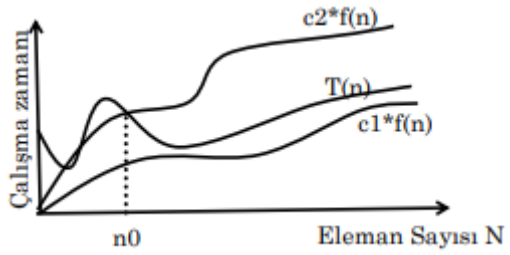
1-Big-O notasyonu: O notasyonu, en kötü durumda asimptotik üst sınırı (*asymptotic upper bound*) göstermek için kullanılır. c ve n0 şeklinde pozitif sabitlerimiz olduğunu düşünelim. n >= n0 ifadesini sağlayan tüm değerler için T(n) <= c*f(n) dir.



2-Ω notasyonu: En iyi durumda asimptotik alt sınırı (*asymptotic lower bound*) göstermek için kullanılır. c ve n0 şeklinde pozitif sabitlerimiz olduğunu düşünelim. n >= n0 ifadesini sağlayan tüm değerler için T(n) >= c*f(n) dir.

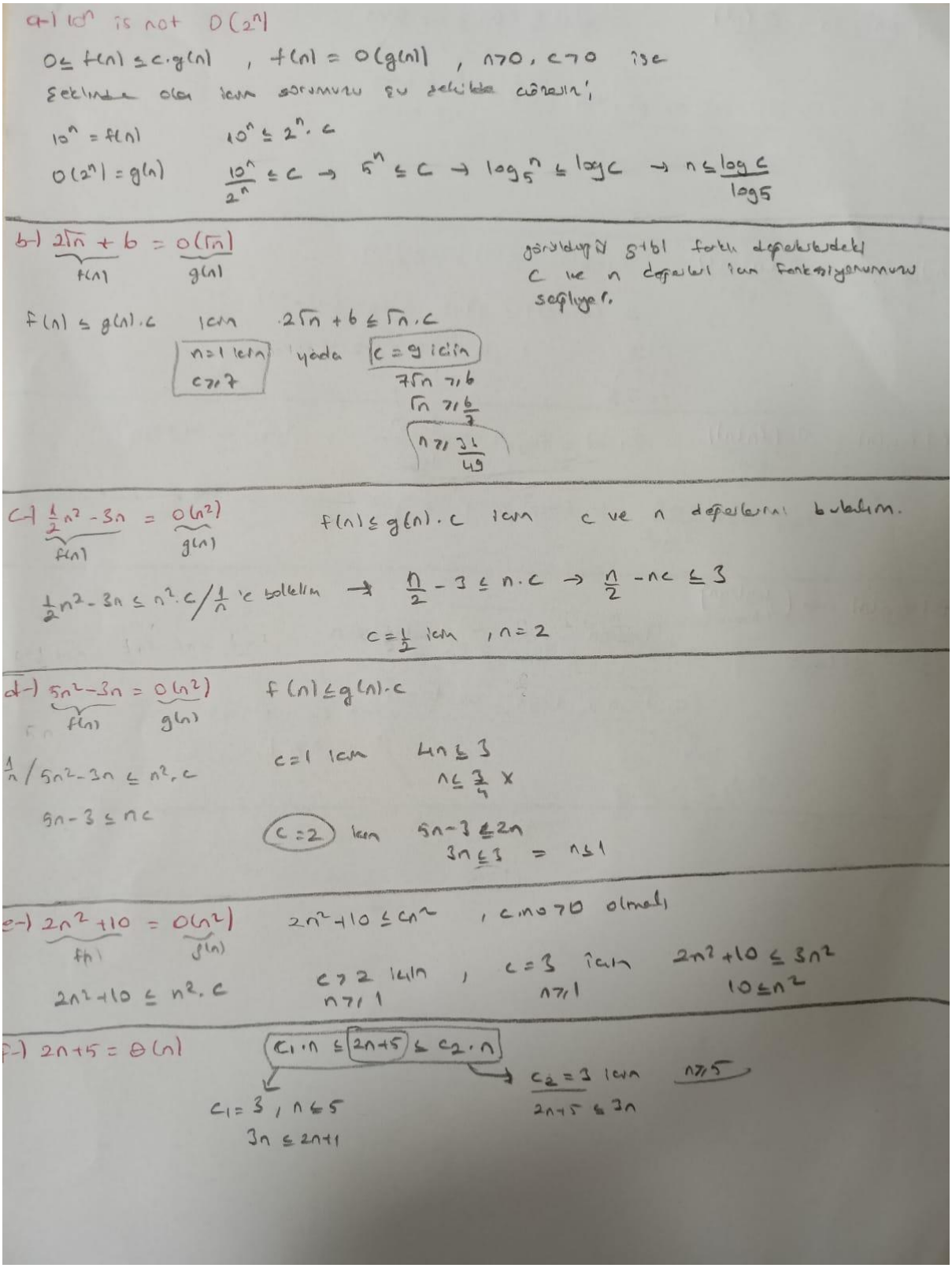


3-θ notasyonu: θ notasyonu programın (algoritmanın) çalışması için gerekli ortalama süreyi göstermek için kullanılır. c1,c2 ve n0 şeklinde pozitif sabitlerimiz olduğunu düşünelim n >= n0 ifadesini sağlayan tüm değerler için c1*f(n) <= T(n) <= c2*f(n) dir.

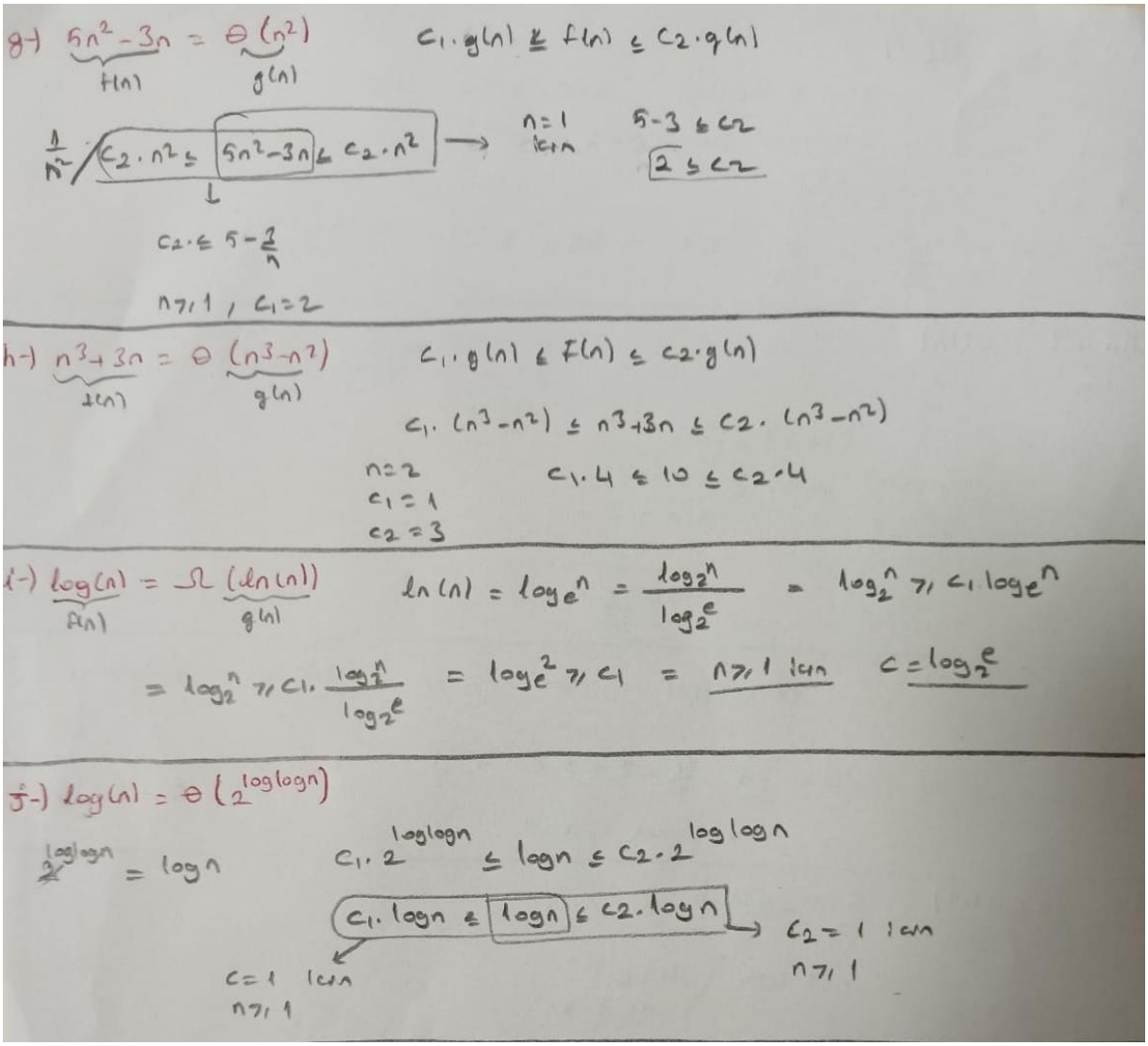


Konuyu da kısaca özet geçtikten sonra aşağıdaki soruların çözümlerine bakalım.

Rapor/Ödev Başlığı :
Hazırlayanın Adı Soyadı :



Rapor/Ödev Başlığı :
Hazırlayanın Adı Soyadı :



KAYNAKÇA

Rapor hazırlanırken ve ödevde yer alan algoritmaların uygulaması gerçekleştirilirken yararlandığım kaynaklar aşağıdaki gibidir:

1- https://www.javatpoint.com/merge-sort
2- https://www.baeldung.com/java-insertion-sort
3- https://www.mobilhanem.com/algoritma-dersleri-insertion-sort/
4- https://www.quora.com/How-can-we-check-for-the-complexity-log-n-and-n-log-n-for-an-algorithm/answer/Rajesh-Durgapal
5- https://www.youtube.com/watch?v=3bhBo9YCTpo&list=PLh9ECzBB8tJPTWIUbZjHZMMGuZcpHUv5h
6- https://medium.com/algorithms-data-structures/algoritma-karma%C5%9F%C4%B1kl%C4%B1%C4%9F%C4%B1-big-o-5f14316890a4
7- https://medium.com/yaz%C4%B1l%C4%B1m-ve-bili%C5%9Fim-kul%C3%BCb%C3%BC/big-o-notation-notasyonu-nedi%CC%87r-490f41de6f76#:~:text=Big%2DO%20notasyonu%20bir%20algoritmay%C4%B1,bir%20g%C3%B6sterim%20oldu%C4%9Fu%20%C5%9Feklinde%20tan%C4%B1mlanm%C4%B1%C5%9Ft%C4%B1r.

8- http://bilgioloji.com/pages/yazilim/kod/program/algoritma/analiz/algoritmalarin-asimptotik-analizinde-hangi-notasyonlar-kullanilir/
9- https://birhankarahasan.com/algoritma-analizi-nedir-zaman-karmasikligi-big-o-gosterimi
10- https://web.ogu.edu.tr/Storage/egulbandilar/Uploads/AlgoritmaAnalizi.pdf
11- https://opendsa-server.cs.vt.edu/embed/mergesortAV
12- https://opendsa-server.cs.vt.edu/OpenDSA/AV/Sorting/insertionsortAV.html
13- https://math.stackexchange.com/questions/512772/prove-that-3n-is-not-o2n