Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

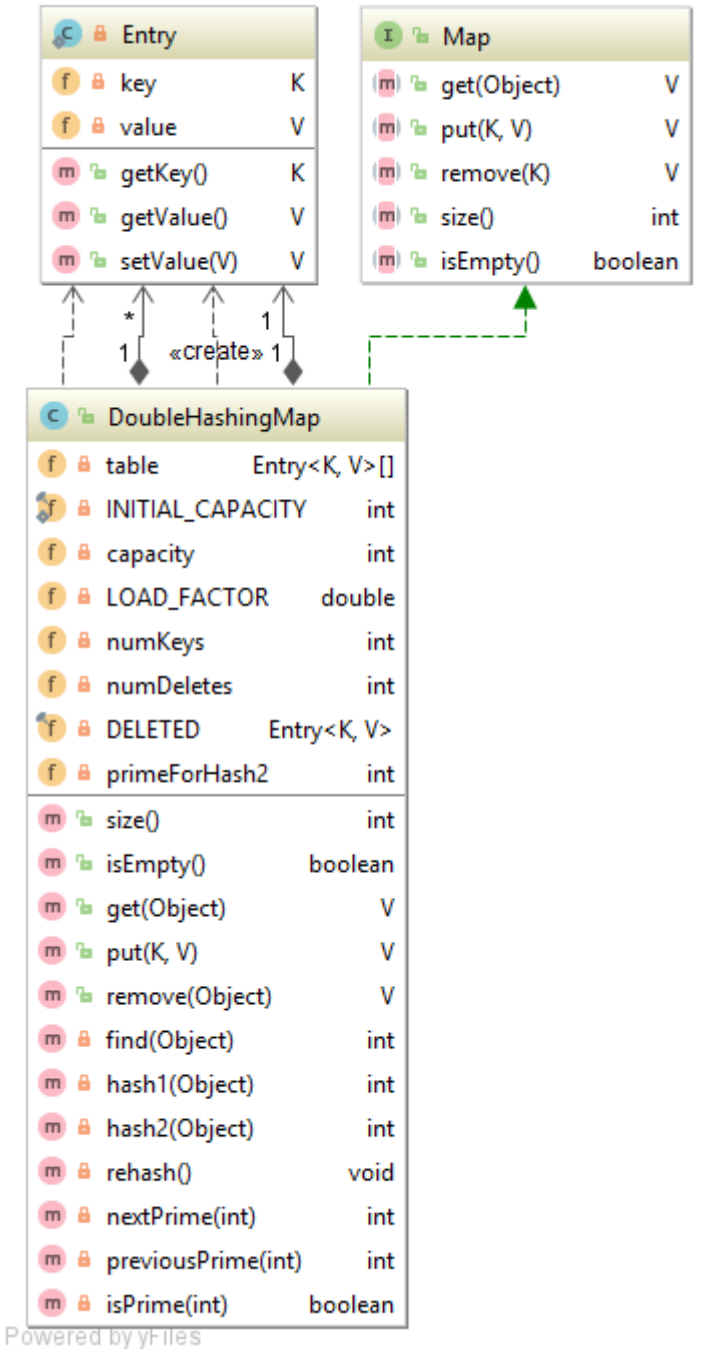
SEVGİ BAYANSALDUZ
151044076

Course Assistant: FATMANUR ESİRCİ

1 Double Hashing Map

This part about Question1 in HW5

1.1 Pseudocode and Explanation



DoubleHashingMap sınıfı Map arayüzünü implement eder.

Bu sınıf içerisinde bulunan Entry sınıfı ile key,value çiftleri tutulur.

DoubleHashingMap sınıfı hashtable mantığı ile çalışır,içerisinde bulunan table veri alanı(data field) Entry tipinde bir dizidir. Bu diziye Entry tipindeki objeler hash yöntemi ile eklenir ve dizi üzerindeki işlemler hash mantığı ile gerçeklenir.

- **SÖZDE KODLAR(PSEUDOCODE CODES)**

```
private int hash1(Object key) { return key.toString().hashCode()%capacity; }
```

Yukarıdaki resimde bulunan metod birinci hash metodudur.Gelen key parametresinin hash kodunun modunu tablonun kapasitesine göre alır ve çıkan sonucu döndürür.

```
private int hash2(Object key) { return primeForHash2-(key.toString().hashCode()%primeForHash2); }
```

Yukarıdaki resimde bulunan metod ikinci hash metodudur.Gelen key parametresinin hash kodunun modunu primeForHas2 veri alanına(data field) göre alır ve çıkan sonucu bu very alanından çıkararak döndürür.Bu very alanı tablo kapasitesinden bir önceki asal sayıdır.

- **Find metodu:**

1. İndekse hash1(key) `i ata
2. Eğer indeks negatif ise
   →indekse tablonun kapasite-sini ekle.
3. Table[index]!=null ve Deleted ve key table[index].key'e eşit oldukça
   →indekse (hash1(key)+ i*hash2(key))%capacity ata.
4. indeksi döndür

```
/**
 * Finds either the target key or the first empty slot in the table
 * @param key target object.
 * @return position of the target or the first empty slot
 */
private int find(Object key) {
    int index = hash1(key);//The first hash method is called
    if (index < 0)
        index += table.length; // Make it positive.
    for ( int i=0;(table[index] != null && table[index] != DELETED )
            && (!key.equals(table[index].key));++i) {
        index=(hash1(key) + i*hash2(key))%capacity;
    }
    return index;
}
/**
```

- **Put metodu:**

1. İndekse find(key) `i ata
2. Table[index] eşittir null ise
   →Yeni eleman ekle
   →numKeysi arttır
   →rehash için kontrol et
   →null döndür
3. Table[index].value`yu bir değişkene ata
4. Table[index].value değerini yeni ve-rilen value parametresi ile değiştir.
5. Değişkene atanan değeri döndür.

```
@Override
public V put(K key, V value) {
    int index = find(key);
    if (table[index] == null)
    {
        table[index] = new Entry < K, V > (key, value);
        numKeys++;
        double loadFactor = (double) (numKeys + numDeletes) / table.length;
        if (loadFactor > LOAD_FACTOR)
            rehash();
        return null;
    }
    V oldVal = table[index].value;
    table[index].value = value;
    return oldVal;
}
```

- **Remove metodu:**

1. İndekse find(key) `i ata
2. Table[index] eşittir null ise
   →null döndür
3. Table[index].value`yu bir değişkene ata
4. Table[index]è DELETED ata
5. numKeysì azalt
6. numDeltesì arttır
7. Değişkene atanan değeri döndür

```
*/
@Override
public V remove(Object key) {
    int index = find(key);
    if (table[index] == null)
        return null;
    V oldValue = table[index].value;
    table[index] = DELETED;
    numKeys--;
    numDeletes++;
    return oldValue;
}
```

1.2 Test Cases

- put

```
@Test
void put() {
    DoubleHashingMap<String,Integer> test1=new DoubleHashingMap<>();
    test1.put("First",1);
    System.out.println("Is adding elements successful?: "+!test1.isEmpty());
    System.out.println("Size of the test1: "+test1.size());
    System.out.println("The value of First is : "+test1.get("First"));
}
```

```
DoubleHashingMapTest > remove()

1 test passed - 31ms

C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
Is adding elements successful?: true
Size of the test1: 1
The value of First is : 1

Process finished with exit code 0
```

- remove

```
@Test
void remove() {
    DoubleHashingMap<String,Integer> test1=new DoubleHashingMap<>();
    test1.put("First",1);
    test1.put("Second",2);
    test1.put("Third",3);
    try {
        System.out.print("The value of Second is : ");
        System.out.println(test1.get("Second"));
    }catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(e);
    }
    test1.remove( key: "Second");
    try {
        System.out.print("After removing,The value of Second is :");
        System.out.println(test1.get("Second"));
    }catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(e);
    }
    test1.remove( key: "Second");
}
```

```
DoubleHashingMapTest

1 test pass

C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
The value of Second is : 2
After removing,The value of Second is :null
```

- isEmpty

```
@Test
void isEmpty() {
    DoubleHashingMap<String,Integer> test1=new DoubleHashingMap<>();
    System.out.println("Is the test1  empty?: "+test1.isEmpty());
    test1.put("First",1);
    test1.put("Second",2);
    test1.put("Third",3);
    System.out.println("After adding two element,Is the test1 empty?: "+test1.isEmpty());
```

```
oubleHashingMapTest
1 test passed - 16ms
C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
The value of Second is : 2
After removing,The value of Second is :null
```

- get

```
@Test
void get() {
    DoubleHashingMap<String,Integer> test1=new DoubleHashingMap<>();
    test1.put("First",1);
    test1.put("Second",2);
    test1.put("Third",3);
    System.out.println("The value of First is : "+test1.get("First")
```

```
oubleHashingMapTest
1 test p
C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
The value of First is : 1
```

- Main

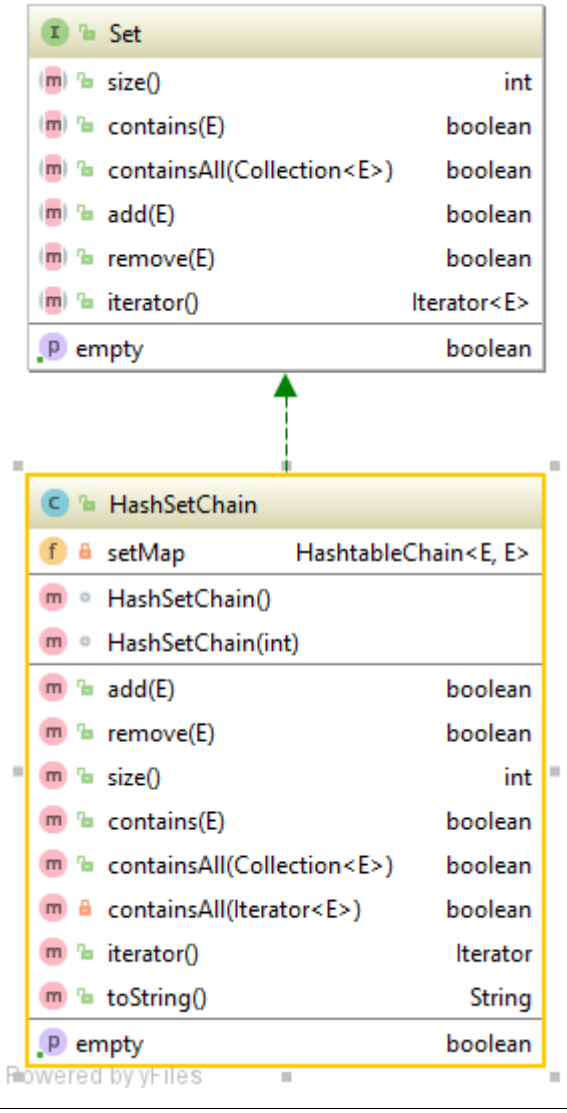
```
public static void main(String[] args) {

    System.out.println("Test 1 ");
    DoubleHashingMap<String,Integer> test1=new DoubleHashingMap<>();

    System.out.println("IS EMPTY:"+test1.isEmpty());
    test1.put("First",1);
    test1.put("Second",2);
    test1.put("Z",29);
    test1.put("F",7);
    test1.put("I",10);
    test1.put("E",6);
    test1.put("D",5);
    System.out.println("IS EMPTY:"+test1.isEmpty()+"\n SIZE: "+test1.size());
    try {
        System.out.print("The value of Z is : ");
        System.out.println(test1.get("Z"));
    }catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(e);
    }
    test1.remove( key: "Second");
    try {
        System.out.print("After removing,The value of Second is :");
        System.out.println(test1.get("Second"));
    }catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(e);
    }
    test1.remove( key: "Second");

}
```

```
C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
Test 1
IS EMPTY:true
IS EMPTY:false
 SIZE: 7
The value of Z is : 29
After removing,The value of Second is :null

Process finished with exit code 0
```

2 Recursive Hashing Set

This part about Question2 in HW5

2.1 Pseudocode and Explanation



HashSetChain sınıfı Set arayüzünü implement eder.

Bu sınıf içerisinde tuttuğu setMap veri alanı (data field) ile HashTableChain metodlarını kullanarak içerisinde barındırdığı metodların implementasyonunu gerçekleştirir.

Bu sınıf iki yapıcı metoda sahiptir.Parametre alan yapıcı işlev verilen parametreye göre tablonun maksimum kapasitesini belirler.

- **SÖZDE KODLAR(PSEUDOCODE CODES)**

  HashSetChain ,HashTableChain sınıfının metodlarını kullandığı için bu sınıfın metodlarının sözde kodları aşağıda verilmiştir.

- **Put metodu**

1. indekse key.hashCode() % tablo.length`i ata.
2. Eğer indeks negatif ise
   →indekse tablonun kapasitesini ekle.
3. Eğer table[indeks] eşittir null ise
   →Elemanı kolayca ekle,
   → numKeys` i arttır,
   →null döndür.
4. Eğer key daha önce tabloda ise
   → Table[index].value değerini yeni verilen value parametresi ile değiştir.
   →Eski value değerini döndür

(Yukarıdaki eğerlere girmiyorsa collision oluşmuştur)

5. Eğer table[index].next (bir sonraki tabloyu işaret eden değer) null ise
   →table[index].next `e yeni obje ata
   →NumKeysi arttır
   →bu obje üzerinden put metodunu çağır (bu adımı return et)
6. Diğer türlü
   → table[index].next.table`put metoduna parametre olarak gönder. (bu adımı return et)

```java
@Override
public V put(K key, V value) { return put(key,value,table);

private V put(K key, V value, Chain<K, V>[] table) {
    int index = key.hashCode() % table.length;
    if (index < 0)
        index += table.length;
    if (table[index]== null)
    {
        table[index] =new Chain<>(key,value) ;
        numKeys++;
        return null;
    }
    else if(table[index].data.equals(key))
        return table[index].data.setValue(value);
    else if (table[index].next==null)
    {
        table[index].next=new HashtableChain<>(prime);
        prime=previousPrime( num: prime-1);
        numKeys++;
        return table[index].next.put(key,value);
    }else
        return put(key,value,table[index].next.table);
}
```

(kodda yer alan prime değeri collision oluştuğunda oluşacak tablonun boyutunu belirler,Bu değer ilk collison oluştuğunda tablo boyutundan küçük ilk asal sayıdır.Örneğin tablo boyutu 10 ise ilk collisonda oluşacak tablonun boyutu 7 olur,ikici collisonda 5 olur bu şekilde 2 `ye doğru gider.)

- **Remove Metodu**

1. indekse key.hashCode() % tablo.length`i ata.
2. Eğer indeks negatif ise → indekse tablonun kapasitesini ekle.
3. Eğer tablonun bulunan indekse karşılık gelen elemanı null ise →null döndür
4. Eğer key tabloda bulunmuşsa vebulunan indeksteki elemanın next`i null ise →Kolayca sil, →numKeysi azalt, →silinen elemanın value değerini döndür.
5. Eğer key bulunmuş ve next null değilse, →table[index].data`ya table[index].next.table[?].data`yı ata →numKeys`i azalt , →Atanan data`yı next.table`dan sil, →ilk silinen değeri döndür .
6. Diğer türlü →table[index].next.table`ı remove metoduna parametre olarak gönder.

```
private V remove(K key, Chain<K, V>[] table) throws Throwable {
    int index = key.hashCode() % table.length;
    if (index < 0)
        index += table.length;
    if (table[index]== null)
        return null;
    if(table[index].data.key.equals(key) && table[index].next==null)
    {
        V returnValue=table[index].data.value;
        table[index]=null;
        numKeys--;
        return returnValue;
    }else if(table[index].data.key.equals(key) && table[index].next!=null)
    {
        V returnValue=table[index].data.value;
        table[index].data= table[index].next.table[key.hashCode()%table[index].next.table.length].data;
        numKeys--;
        table[index].next.remove(table[index].data.key);
        return returnValue;
    }else
    {
        V returnValue=remove(key,table[index].next.table);
       if(isEmpty(table[index].next.table))
          table[index].next=null;
        return returnValue;
    }
}
```

2.2 Test Cases

- **Add**

```
@Test
void add() {
 HashSetChain<Integer> test=new HashSetChain<>( i: 10);
    test.add(14);test.add(24);
    test.add(10);test.add(34);
    test.add(20);
    test.add(94);
    test.add(60);test.add(40);
    System.out.println(test.toString()+"\nsize="+test.size());
}
```

```
shSetChainTest > remove()

C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
[10,20,60,40,14,24,94,34]
size=8

Process finished with exit code 0
```

- **Remove**

```
@Test
void remove() {
    HashSetChain<Integer> test=new HashSetChain<>( i: 10);
    test.add(14);test.add(24);
    test.add(10);test.add(34);
    test.add(20);test.add(94);
    test.add(60);test.add(40);
    System.out.println(test.toString());
    try {
        test.remove( e: 94);
    } catch (Throwable throwable) {
        throwable.printStackTrace();
    }
    System.out.println(test.toString());
    try {
        test.remove( e: 20);
    } catch (Throwable throwable) {
        throwable.printStackTrace();
    }
    System.out.println(test.toString());
}
```

```
C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
[10,20,60,40,14,24,94,34]
[10,20,60,40,14,24,34]
[10,60,40,14,24,34]
```

- **Contains**

```
@Test
void contains() throws Throwable {
    HashSetChain<Integer> test=new HashSetChain<>( i: 10);
    test.add(14);test.add(24);
    test.add(10);test.add(34);
    test.add(20);test.add(94);
    test.add(60);test.add(40);
    System.out.println("contains:"+test.contains(94));
    test.remove( e: 94);
    System.out.println("contains:"+test.contains(94));
}
```

HashSetChainTest › remove()

```
C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
contains:true
contains:false
```

- **ContainsAll**

```
@Test
void containsAll() {
    HashSetChain<Integer> test=new HashSetChain<>();
    test.add(1);test.add(2);
    test.add(0);test.add(4);
    test.add(5);test.add(8);
    test.add(11);test.add(13);
    test.add(17);test.add(19);
    test.add(31);test.add(23);
    List<Integer> test2=new ArrayList<>();
    test2.add(8);
    test2.add(5);
    System.out.println(test.containsAll(test2));
    test2.add(3);
    System.out.println(test.containsAll(test2));
}
```

```
HashSetChainTest > remove()

C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
true
false

Process finished with exit code 0
```

- **Empty**

```
@Test
void isEmpty() {
    HashSetChain<Integer> test=new HashSetChain<>( i: 10);
    System.out.println("Empty:"+test.isEmpty());
    test.add(14);test.add(24);
    System.out.println("Empty:"+test.isEmpty());
}
```

```
HashSetChainTest > remove()

1 test passed - 31ms

C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
Empty:true
Empty:false
```

- **Size**

```
@Test
void size() {
    HashSetChain<Integer> test=new HashSetChain<>( i: 10);
    System.out.println("Size:"+test.size());
    test.add(14);test.add(24);
    System.out.println("Size:"+test.size());
}
```

```
shSetChainTest > remove()

C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
Size:0
Size:2
```

3 Sorting Algortihms

Sıralanan diziler random olarak Random sınıfı ile oluşur. Aşağıdaki resimlerde bulunan diziler örnek olarak eklenmiştir.

- Ortalama çalışma süresi bulmak için kullanılan diziler:

```
C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
First Array(size:10)
Before sorted:[ 9566, 941, 21224, 10228, 9075, 13533, 6645, 4930, 8742, 15746 ]
SortArray2:[ 941, 4930, 6645, 8742, 9075, 9566, 10228, 13533, 15746, 21224 ]
```

Resim avrg.1

```
**********
Second Array(size:20)
Before sorted:[ 9395, 17241, 23232, 1214, 3407, 11665, 20088, 10033, 17425, 19414, 6699, 13039, 2083, 11173, 7892, 18279, 13382, 19598, 1970, 16482 ]
SortArray2:[ 1214, 1970, 2083, 3407, 6699, 7892, 9395, 10033, 11173, 11665, 13039, 13382, 16482, 17241, 17425, 18279, 19414, 19598, 20088, 23232 ]
```

Resim avrg.2

```
**********
Third Array(size:50)
Before sorted:[ 10566, 12936, 14305, 16873, 120, 6268, 17571, 12192, 20464, 7662, 12521, 20385, 3315, 16769, 17951, 23092, 17289, 4677, 7448, 18757, 13763, 7916, 19077
SortArray2:[ 5, 120, 1301, 2136, 2385, 2773, 2889, 3315, 3933, 4677, 4857, 6268, 7448, 7662, 7829, 7916, 9223, 10016, 10566, 11136, 12192, 12247, 12521, 12936, 13188,
```

Resim avrg.3

```
**********
Fourth Array(size:100)
Before sorted:[ 5566, 2402, 6458, 8896, 16546, 10852, 250, 7377, 11501, 14613, 23230, 2596, 1336, 7239, 139, 952, 17733, 18771, 3213, 12686, 6085, 9753, 10339, 7894, 1641
SortArray2:[ 139, 250, 952, 1336, 1583, 1631, 2138, 2290, 2402, 2454, 2581, 2596, 2755, 3169, 3180, 3213, 3542, 3751, 3789, 4257, 4389, 4958, 5566, 5597, 5994, 6012, 6030
```

Resim avrg.4

```
**********
Fifth Array(size:200)
Before sorted:[ 12926, 24128, 21242, 18251, 8886, 16631, 11880, 9561, 9187, 2884, 6360, 21123, 4016, 24241, 22122, 22903, 8309, 22133, 14901, 24586, 12178, 9190, 14649, 20544
SortArray2:[ 139, 394, 735, 1216, 1231, 1326, 1364, 1412, 1473, 1492, 1659, 1680, 1714, 1789, 1992, 2455, 2554, 2690, 2884, 3248, 3251, 3448, 3503, 3705, 4016, 4119, 4210, 46
```

Resim avrg.5

```
**********
Sixth Array(size:500)
Before sorted:[ 3888, 13685, 13190, 5908, 373, 24684, 10271, 1691, 14889, 24413, 4242, 3144, 5209, 17049, 3677, 7075, 15725, 21776, 14683, 2170, 1095, 16425, 651, 9152, 12895, 1
SortArray2:[ 191, 372, 373, 421, 506, 596, 618, 651, 651, 683, 698, 782, 815, 822, 902, 958, 1004, 1093, 1095, 1156, 1168, 1180, 1283, 1325, 1412, 1484, 1534, 1585, 1691, 1701,
```

Resim avrg.6

```
**********
Seventh Array(size:1000)
Before sorted:[ 17963, 17943, 7114, 18858, 2293, 13666, 14881, 14870, 8132, 16548, 17197, 12228, 5944, 19285, 6998, 23840, 24666, 10249, 10523, 17544, 16718, 21177, 8498, 13808,
SortArray2:[ 14, 19, 34, 35, 66, 99, 108, 136, 143, 242, 253, 255, 259, 318, 321, 327, 332, 352, 360, 400, 432, 440, 459, 460, 548, 586, 594, 654, 655, 659, 674, 696, 711, 739,
```

Resim avrg.7

```
**********
Eighth Array(size:2000)
Before sorted:[ 16687, 11925, 2839, 12976, 6251, 2205, 22569, 14323, 1262, 9440, 19460, 18970, 4819, 20168, 1528, 10086, 1766, 13547, 15405, 18331, 21693, 10241, 3080, 14591, 73
SortArray2:[ 23, 26, 34, 38, 69, 81, 104, 106, 108, 111, 126, 134, 135, 145, 165, 175, 185, 190, 190, 195, 197, 204, 204, 230, 247, 248, 254, 262, 281, 287, 327, 331, 348, 349,
```

Resim avrg.8

```
**********
Ninth Array(size:5000)
Before sorted:[ 6842, 4468, 18565, 10210, 24431, 12478, 2430, 15122, 21547, 21102, 417, 13396, 3012, 5140, 3177, 21942, 16489, 21397, 1421, 4516, 8370, 23591,
SortArray2:[ 9, 11, 13, 21, 22, 29, 30, 37, 39, 40, 46, 47, 58, 70, 75, 75, 77, 87, 98, 98, 105, 108, 111, 114, 123, 126, 132, 132, 134, 137, 138, 144, 154, 155
```

Resim avrg.9

```
**********
Tenth Array(size:100000)
Before sorted:[ 19037, 14463, 6754, 18000, 6787, 19793, 23389, 7327, 12088, 12316, 19118, 4173, 2140, 6092, 19432, 14998, 24408, 11697, 23314, 23166, 23863, 16080, 24849, 23355, 2
SortArray2:[ 0, 2, 4, 4, 5, 11, 14, 16, 18, 23, 29, 32, 37, 38, 38, 39, 44, 45, 45, 48, 48, 52, 55, 56, 58, 60, 62, 66, 67, 68, 70, 71, 74, 75, 78, 79, 82, 82, 83, 85, 91, 98, 101
```

Resim avrg.10

- Worst case çalışma süresi bulmak için kullanılan diziler:

```
C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
First Array(size:100)
Before sorted:[ 23523, 22871, 22800, 22573, 22413, 22262, 22040, 22014, 21603, 21555, 21294, 20901, 20622, 20616, 20563, 20511, 20216, 20110
SortArray2:[ 45, 134, 233, 1131, 1557, 1588, 2230, 2405, 3093, 3127, 3740, 3770, 3990, 4091, 4125, 4184, 4290, 4321, 4355, 4754, 5090, 5227,
```

Resim worst.1

```
*******************************************************
Second Array(size:1000)
Before sorted:[ 24978, 24976, 24955, 24946, 24936, 24926, 24879, 24859, 24857, 24849, 24834, 24793, 24770, 24699, 24676, 24667, 24656, 24620
SortArray2:[ 145, 146, 156, 170, 199, 255, 255, 267, 271, 294, 347, 372, 404, 425, 432, 438, 504, 535, 581, 594, 599, 629, 641, 647, 822, 83
```

Resim worst.2

```
*******************************************************
Third Array(size:5000)
Before sorted:[ 24999, 24994, 24993, 24987, 24976, 24971, 24970, 24954, 24952, 24932, 24930, 24925, 24917, 24916, 24911, 24910, 24909, 24895, 24894, 24890, 24885, 248
SortArray2:[ 11, 12, 13, 19, 20, 30, 36, 37, 41, 42, 42, 45, 45, 48, 69, 75, 75, 80, 81, 81, 92, 94, 95, 95, 96, 98, 105, 119, 121, 130, 131, 134, 162, 176, 177, 177,
```

Resim worst.3

```
*******************************************************
Fourth Array(size:10000)
Before sorted:[ 24997, 24988, 24985, 24982, 24982, 24978, 24975, 24964, 24964, 24963, 24962, 24962, 24957, 24954, 24953, 24953, 24947, 24946, 24945, 249
SortArray2:[ 5, 6, 9, 18, 19, 20, 23, 38, 39, 45, 55, 56, 61, 61, 62, 70, 71, 73, 79, 81, 85, 87, 88, 96, 99, 107, 108, 108, 112, 112, 113, 117, 126, 12
```

Resim worst.4

3.1 MergeSort with DoubleLinkedList

This part about Question3 in HW5

3.1.1 Pseudocode and Explanation

- SÖZDE KODLAR(PSEUDOCODE CODES)

MergeSort

1. mergeSort(head) metodunun döndürdüğü değeri sorted objesine ata.
2. Head`e sorted.head ata.
3. Tail `e sorted.tail ata

MergeSort(left)

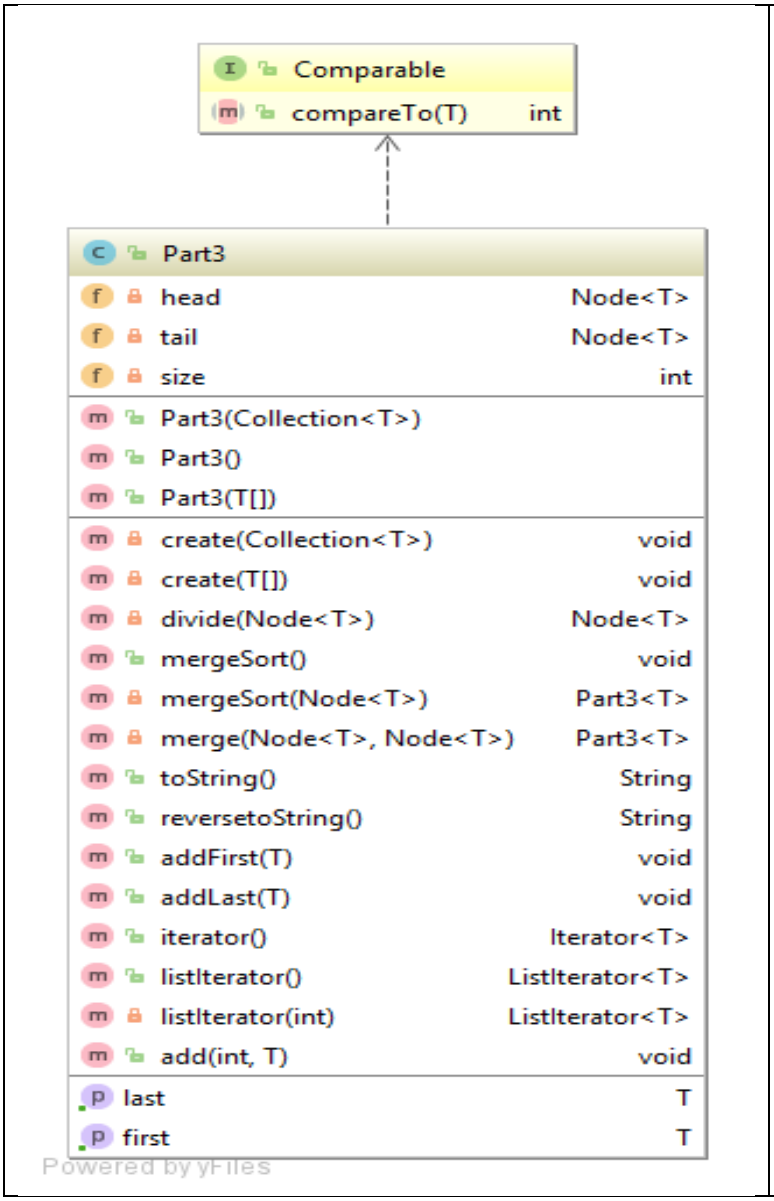
1. left null yada left.next null ise → merge(left,null) döndür.
2. right`a linkedListin sağ yarısını ata.
3. left`e mergeSort(left).head ata
4. rigth`a mergeSort(right).head ata
5. merge(left,right) döndür

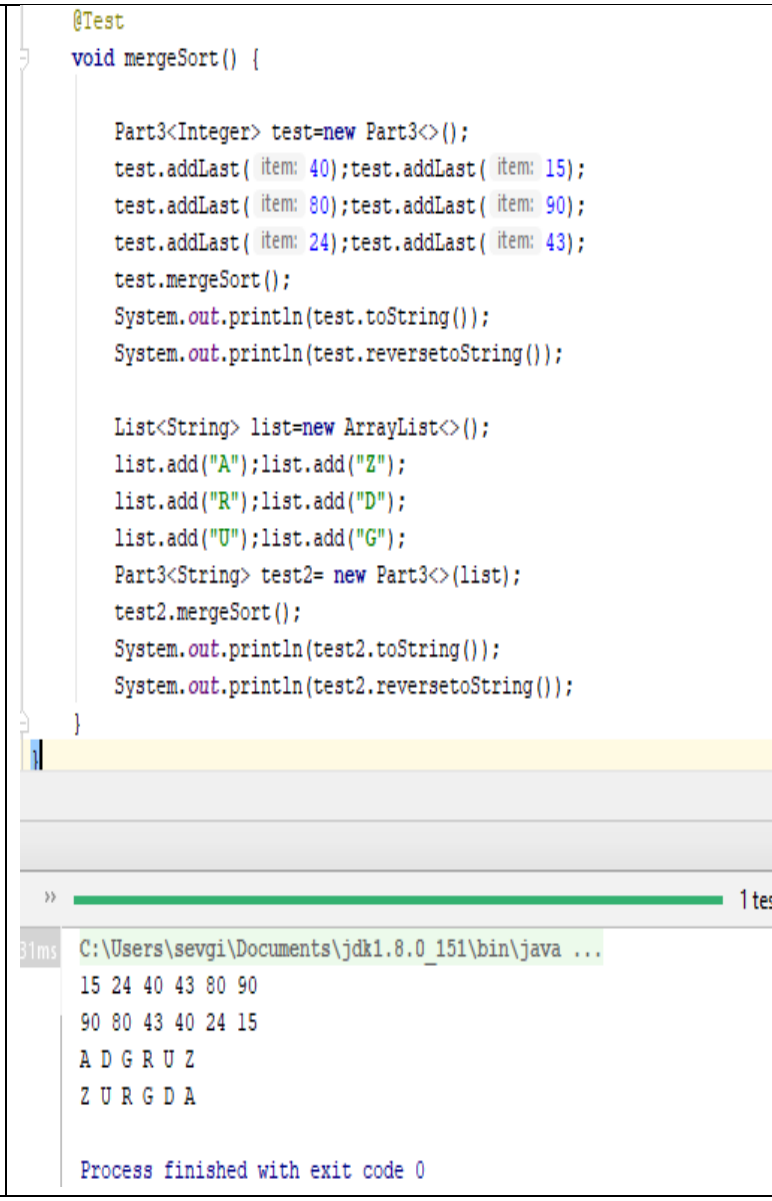
```
public void mergeSort()
{
    Part3 sorted=mergeSort(head);
     head=sorted.head;
     tail=sorted.tail;
}

/** Sort the array using the merge sort algorithm.
 pre: LinkedList contains Comparable objects.
 post: LinkedList is sorted.
 @param left The LinkedList to be sorted
 */
private Part3<T> mergeSort( Node<T> left)
{
    if(left==null || left.next==null)
        return merge(left, right: null);
    //divide the linkedList
    Node<T> right=divide(left);

    // Sort the halves.
    left=mergeSort(left).head;
    right=mergeSort(right).head;
    return merge(left,right);
}
```

Merge(left,right)

1. newArrayì oluştur(Yeni bir obje)
2. i `ye 0 ata
3. left ve right null olmadıkça
   →Eğer left.datası Righ.data`dan büyükse
   ->left.data`yı newArraye ekle
   →diğer türlü
   ->right.data`yı newArraye ekle
4. left null olmadıkça
   →left.data`yı newArraye ekle
5. rigth null olmadıkça
   →right.data`yı newArraye ekle

```
/**Merge two sequences.
 * pre: left  and right  are sorted.
 * post: return the merged result and is sorted.
 * @return the merged result and is sorted.
 */
private Part3<T> merge(Node<T> left, Node<T> right) {
    Part3<T> newArray=new Part3<>();
    int i=0;
    while (left!=null && right!=null)
    {
        // Find the smaller and
        // insert it into the newArray .
        if (left.data.compareTo(right.data)<0)
        {
            newArray.add(i++,left.data);
            left=left.next;
        }else{
            newArray.add(i++,right.data);
            right=right.next;
        }
    }while (left!=null)
    {// assert: one of the sequences has more items to add.
        // add remaining input from left  into the newArray.
        newArray.add(i++,left.data);
        left=left.next;
    }
    while (right!=null)
    {
        //// add remaining input from left  into the newArray.
        newArray.add(i++,right.data);
        right=right.next;
    }
    return newArray ;
```





```
@Test
void mergeSort() {

    Part3<Integer> test=new Part3<>();
    test.addLast( item: 40);test.addLast( item: 15);
    test.addLast( item: 80);test.addLast( item: 90);
    test.addLast( item: 24);test.addLast( item: 43);
    test.mergeSort();
    System.out.println(test.toString());
    System.out.println(test.reversetoString());

    List<String> list=new ArrayList<>();
    list.add("A");list.add("Z");
    list.add("R");list.add("D");
    list.add("U");list.add("G");
    Part3<String> test2= new Part3<>(list);
    test2.mergeSort();
    System.out.println(test2.toString());
    System.out.println(test2.reversetoString());
}
}
```

```
C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
15 24 40 43 80 90
90 80 43 40 24 15
A D G R U Z
Z U R G D A

Process finished with exit code 0
```

3.1.2 Average Run Time Analysis

- Boyutu 10 olan Resim avrg.1`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 153291 nanoseconds.
```

- Boyutu 20 olan Resim avrg.2`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 188551 nanoseconds.
```

- Boyutu 50 olan Resim avrg.3`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 273107 nanoseconds.
```

- Boyutu 100 olan Resim avrg.4`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 549670 nanoseconds.
```

- Boyutu 200 olan Resim avrg.5`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 280666 nanoseconds.
```

- Boyutu 500 olan Resim avrg.6`daki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 709549 nanoseconds.
```

- Boyutu 1000 olan Resim avrg.7`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 4205084 nanoseconds.
```

- Boyutu 2000 olan Resim avrg.8`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 1671256 nanoseconds.
```

- Boyutu 5000 olan Resim avrg.9`daki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 2081079 nanoseconds.
```

- Boyutu 10000 olan Resim avrg.10`daki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 4158972 nanoseconds.
```

Ortalama çalışma süresi aşağıdaki tabloda gösterilmiştir :

Size	Merge(PART3)
10	153291
20	188551
50	273107
100	549670
200	280666
500	709549
1000	4205084
2000	1671256
5000	2081079
10000	4158972
average	1427122,5

average hücresine karşılık gelen çalışma süresi ortalama çalışma süresidir.

3.1.3 Wort-case Performance Analysis

- Boyutu 100 olan Resim worst.1`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 916836 nanoseconds.
```

- Boyutu 1000 olan Resim worst.2`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 6894202 nanoseconds.
```

- Boyutu 5000 olan Resim worst.3`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 3389650 nanoseconds.
```

- Boyutu 10000 olan Resim worst.4`deki dizi için çalışma süresi:

```
Sorted with merge sort (from the Part3)in: 5318948 nanoseconds.
```

Worst Case çalışma süresi aşağıdaki tabloda gösterilmiştir:

Size	Merge(PART3)
100	916836
1000	6894202
5000	3389650
10000	5318948
Average	4129909

Bu sütun worst case çalışma süresini verir.

3.2 MergeSort

Yukarıda resimleri verilen farklı boyutlardaki dizilerin mergeSort ile çalışma süreleri

151044076.zip isimli zip dosyası içerisindeki ScreenShot4 ve ScreenShot5 klasörleri içerisinde bulunmaktadır. Dosya içerisindeki resimler dizinin boyutuna göre adlandırılmıştır.(Heap,Insertion ve Quick sort metodları için de aynı durum geçerlidir.)

3.2.1 Average Run Time Analysis

Ortalama çalışma süresi aşağıdaki tabloda gösterilmiştir :

Size	Merge
10	32829
20	57936
50	23757
100	35528
200	92601
500	687357
1000	1631677
2000	17416982
5000	6269696
10000	2909364
average	2915773

Bu sütun ortalama çalışma süresini verir.

3.2.2 Wort-case Performance Analysis

Worst Case çalışma süresi aşağıdaki tabloda gösterilmiştir:

Size	Merge
100	265925
1000	4129709
5000	43959556
10000	15719005
Average	16018548,75

Bu sütun worst case çalışma süresini verir.

3.3 Insertion Sort

3.3.1 Average Run Time Analysis

Ortalama çalışma süresi aşağıdaki tabloda gösterilmiştir :

Size	Insertion
10	7451
20	21382
50	24999
100	75269
200	1191347
500	335471
1000	1305007
2000	5866243
5000	45422511
10000	177686040
average	23193572

Bu sütun ortalama çalışma süresini verir.

3.3.2 Wort-case Performance Analysis

Worst Case çalışma süresi aşağıdaki tabloda gösterilmiştir:

Size	Insertion
100	179209
1000	6489130
5000	82374210
10000	339473169
Average	107128929,5

Bu sütun worst case çalışma süresini verir.

3.4 Quick Sort

3.4.1 Average Run Time Analysis

Ortalama çalışma süresi aşağıdaki tabloda gösterilmiştir :

Size	Quick
10	18898
20	205667
50	451074
100	32289
200	483148
500	173648
1000	5392058
2000	4879429
5000	1220937
10000	1772821
average	1462996,9

Bu sütun ortalama çalışma süresini verir.

3.4.2 Wort-case Performance Analysis

Worst Case çalışma süresi aşağıdaki tabloda gösterilmiştir:

Size	Quıck
100	298646
1000	9238561
5000	61079847
10000	95059556
Average	41419152,5

Bu sütun worst case çalışma süresini verir.

3.5 Heap Sort

3.5.1 Average Run Time Analysis

Ortalama çalışma süresi aşağıdaki tabloda gösterilmiştir :

Size	Heap
10	32342
20	30129
50	65463
100	204857
200	745078
500	2219144
1000	1149069
2000	2202460
5000	3569507
10000	3334359
average	1355240,8

Bu sütun ortalama çalışma süresini verir.

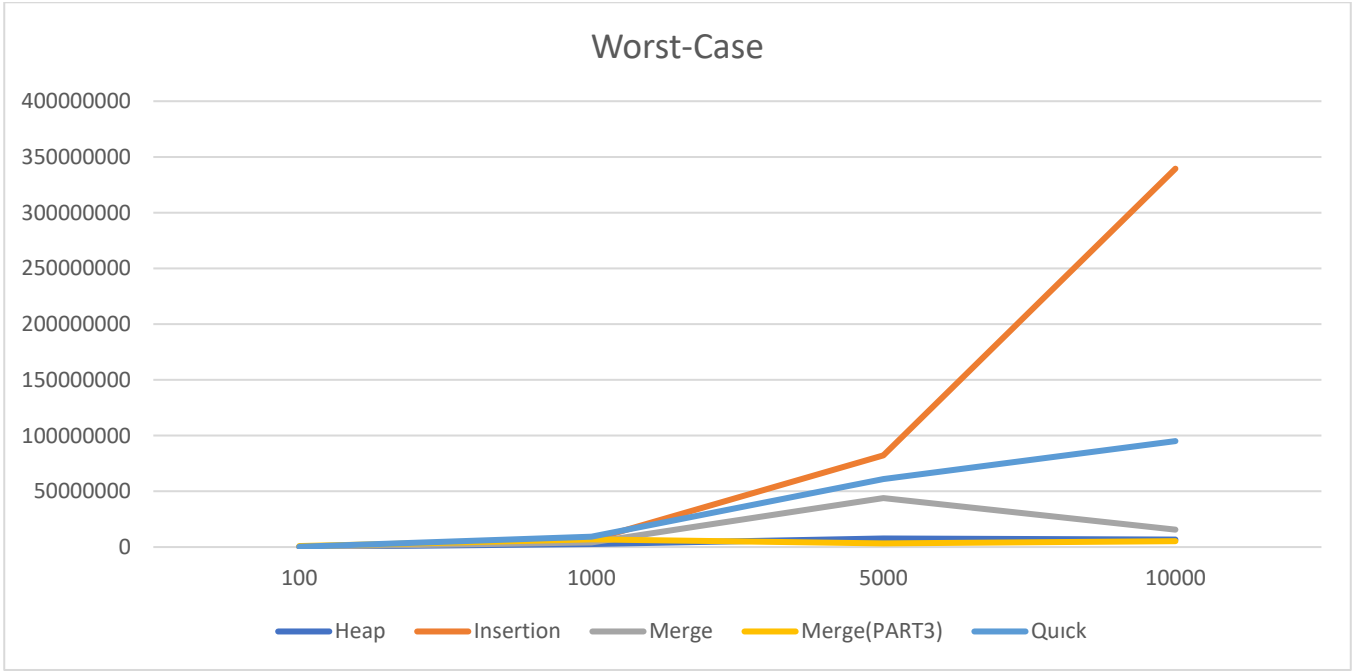
3.5.2 Wort-case Performance Analysis

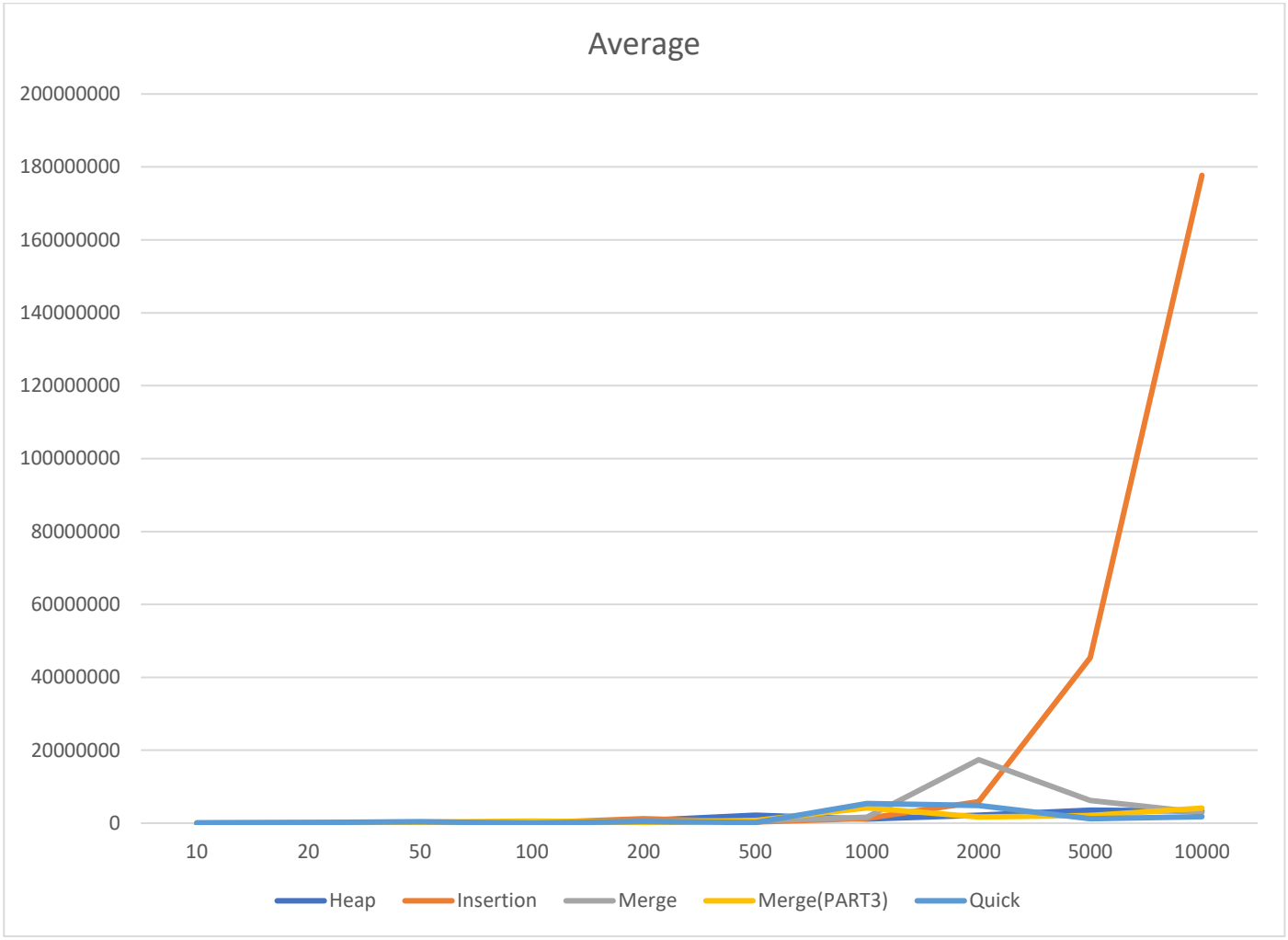
Worst Case çalışma süresi aşağıdaki tabloda gösterilmiştir:

Size	Heap
100	217384
1000	2694571
5000	7727022
10000	6668232
Average	4326802,25

Bu sütun worst case çalışma süresini verir.

4 Comparison the Analysis Results




Average
200000000
180000000
160000000
140000000
120000000
100000000
80000000
60000000
40000000
20000000
0
10
20
50
100
200
500
1000
2000
5000
10000
Heap
Insertion
Merge
Merge(PART3)
Quick