

**Gebze Technical University
Computer Engineering**

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

**BURHAN ELGÜN
151044060**

Course Assistant: Fatma Nur ESİRCİ

1 Double Hashing Map

1.1 Pseudocode and Explanation

Write pseudocode and explanation about code design. Indicate what you are using that interfaces, classes, structures, etc.

MapCell adında bir class oluşturuldu.

MapCell classı her bir Map entrisini temsil etmektedir ve içerisinde key value pairlerini tutmaktadır.

MapCell constructoru oluşturuldu ve gelen key ve value değerlerini kendi data fieldlarına atar.

Map interfacesini implement eden MapWithHashTable adında bir class oluşturuldu.

MapWithHashTable classında MapCell tipinde hashTable adında bir array datafield olarak oluşturuldu. Bu array hash table dır. (MapWithHashTable classında MapCell classı kullanıldı)

MapWithHashTable classında datafield olarak capacity tutuldu.

MapWithHashTable classında datafield olarak elementSize tutuldu.

Capacity alan bir constructor yapıldı ve bu constructor capacitye göre hashTable oluşturur

Size methodu eleman sayısını return edecek şekilde implement edildi.

isEmpty methodu eleman sayısını kontrol ederek boş olup olmadığını return edecek şekilde implement edildi.

containsKey methodu get methodunu kullanır ve eğer gelen obje ile birbirine eşitse true return eder.

getPrime methodu capacity den başlayarak en yakın ilk prime sayıyı return eder.

hashCode1 methodu Object tipinde bir key alır ve o keyin hash valuesini, hashcodun capacitye göre modunu alarak return eder.

hashCode2 methodu Object tipinde bir key alır ve keyin hashcodunun capacitye göre modunu bulduktan sonra bir de capacitye en yakın prime sayıya göre modunu alır ve ardından gelen değeri capacitye en yakın prime sayıdan çıkararak return eder.

Get methodu hashCode1 ve hashCode2 methodlarını çalıştırarak iki farklı hash değeri üretir ardından has1 kullanılarak get işlemi gerçekleştirilmiyorsa hash2 kullanılır ve parametre olarak gelen key değerinin valuesi return edilir.

Put methodu hem key hem value alır ardından hash1 ve hash2 kodları ürettirilir key için. Daha sonra has1 için put işlemi gerçekleştirilmiyorsa hash2 için put işlemi gerçekleştirilir. Ve elementSize 1 arttırılır.

Remove methodu object tipinde key alır ve hash1 ve hash2 kullanılarak o key silinmeye çalışılır. Öncelikle hash1 kullanılır ve remove işlemi gerçekleştirilmezse hash2 kullanılır.

Clear methodu baştan sona tüm Map i null ile doldurur.

1.2 Test Cases

```
Q1 [-/IdeasProjects/Q1] - .../src/com/company/Main.java [Q1] - IntelliJ IDEA
Main.java x MapWithHashTable.java x MapCell.java x
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-1.306/bin/java ...
MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1 - MAP1
Size: 15 olan bir Double Hashing Map Oluşturuldu.
Double hashing map in içine 7 adet yeni key value pairi eklendi.(put methoduyla)
Hash table print edilir.
Hash Table
hasTable[0]=null
hasTable[1]=1, Adana
hasTable[2]=null
hasTable[3]=null
hasTable[4]=34, İstanbul
hasTable[5]=25, İzmir
hasTable[6]=6, Ankara
hasTable[7]=null
hasTable[8]=null
hasTable[9]=null
hasTable[10]=null
hasTable[11]=41, Kocaeli
hasTable[12]=57, Sinop
hasTable[13]=26, Eskişehir
hasTable[14]=null

Keyi 1 olan value değerini get methoduyla alırsız ve ekrana basarsız
1 = Adana
Key'i 34 olan pair listeden silinir(34,İstanbul)
Pair silindimi diye map tekrar print edilir
Hash Table
hasTable[0]=null
hasTable[1]=1, Adana
hasTable[2]=null
hasTable[3]=null
hasTable[4]=99, Deleted
hasTable[5]=25, İzmir
hasTable[6]=6, Ankara
hasTable[7]=null
hasTable[8]=null
hasTable[9]=null
hasTable[10]=null
hasTable[11]=41, Kocaeli
hasTable[12]=57, Sinop
hasTable[13]=26, Eskişehir
hasTable[14]=null

Silinen pairin(34,İstanbul) keyi mapte bulunur mu diye kontrol edilir(containsKey methoduyla)= false
Mapte bulunan bir pairin(41,Kocaeli) keyi mapte bulunur mu diye kontrol edilir(containsKey methoduyla) = true
.....
Compilation completed successfully in 1s 712ms (a minute ago)

Q1 [-/IdeasProjects/Q1] - .../src/com/company/Main.java [Q1] - IntelliJ IDEA
Main.java x MapWithHashTable.java x MapCell.java x
Run Main
hashtableyeni=map1
hasTable[9]=null
hasTable[10]=null
hasTable[11]=41, Kocaeli
hasTable[12]=57, Sinop
hasTable[13]=26, Eskişehir
hasTable[14]=null

Silinen pairin(34,İstanbul) keyi mapte bulunur mu diye kontrol edilir(containsKey methoduyla)= false
Mapte bulunan bir pairin(41,Kocaeli) keyi mapte bulunur mu diye kontrol edilir(containsKey methoduyla) = true
.....
MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2 - MAP2
Size: 10 olan bir Double Hashing Map Oluşturuldu.
Double hashing map in içine 5 adet yeni key value pairi eklendi.(put methoduyla).
Hash table print edilir.
Hash Table
hasTable[0]=null
hasTable[1]=61, Trabzon
hasTable[2]=42, Konya
hasTable[3]=null
hasTable[4]=null
hasTable[5]=null
hasTable[6]=16, Bursa
hasTable[7]=51, Niğde
hasTable[8]=null
hasTable[9]=39, Kırklareli

Keyi 42 olan value değerini get methoduyla alırsız ve ekrana basarsız
42 = Konya
Key'i 61 olan pair listeden silinir(61,Trabzon)
Pair silindimi diye map tekrar print edilir
Hash Table
hasTable[0]=null
hasTable[1]=99, Deleted
hasTable[2]=42, Konya
hasTable[3]=null
hasTable[4]=null
hasTable[5]=null
hasTable[6]=16, Bursa
hasTable[7]=51, Niğde
hasTable[8]=null
hasTable[9]=39, Kırklareli

Silinen pairin(61,Trabzon) keyi mapte bulunur mu diye kontrol edilir(containsKey methoduyla)= false
Mapte bulunan bir pairin(39,Kırklareli) keyi mapte bulunur mu diye kontrol edilir(containsKey methoduyla) = true
.....
Process finished with exit code 0
Compilation completed successfully in 1s 712ms (a minute ago)
```

2 Recursive Hashing Set

This part about Question2 in HW5

2.1 Pseudocode and Explanation

Write pseudocode and explanation about code design. Indicate what you are using that interfaces,

classes, structures, etc.

2.2 Test Cases

Try this code least 2 different hash table size and 2 different sequence of keys. Report all of situations.

3 Sorting Algorithms

Not: ScreenShotlardaki kısaltmaların anlamları:

AQS = Average Case Quick Sort

AHS = Average Case Heap Sort

AIS = Average Case Insertion Sort

AMS = Average Case Merge Sort

AmSDLL = Average Case Merge Sort With DLL

3.1 MergeSort with DoubleLinkedList

3.1.1 Pseudocode and Explanation

MergeSortWithDLL adında bir class oluşturuldu.

Bu classın içerisinde Node inner classı oluşturuldu.

Node classının içerisinde int data, Node next, Node prev tutuldu.

MergeSortWithDLL classında Node head, Node tail, int size tutuldu.

Double Linked Listeye add işlemi yapabilmek için addLast methodu implemant edildi. Bu method tail inden bir sonraki yer için ekleme yapmaktadır.(3 parametrelili Node constructoruyla)

Split methodu yazıldı , burda amaç merge yapabilmek için önce listeyi ortdan ikiye bölmektir. Bu amaçla split methodu listeyi iki parçaya böler.

MergeSort methodu gelen Node ve nexti null değilken(base case) recursive olarak splitteden gelen iki farklı listeyide mergeSort methoduna yollar(recursive)

Ardından bunları merge eder.

Merge methodu kendisine gelen iki farklı sıralı listeyi birleştirir. Hangisinin ilk elemanı daha küçükse o ilk liste olur diğeri onun sonuna eklenir.

3.1.2 Average Run Time Analysis

$O(n \log n)$ = Teta($n \log n$) zaman alır.

Size = 20 => 20.562 micro saniye (10 tane testin ortalaması)

Size = 40 => 65.915 micro saniye (10 tane testin ortalaması)

Size = 80 => 22.100 micro saniye (10 tane testin ortalaması)

Size = 160 => 33.141 micro saniye (10 tane testin ortalaması)

Size = 320 => 40.687 micro saniye (10 tane testin ortalaması)

Size = 640 => 51.596 micro saniye (10 tane testin ortalaması)

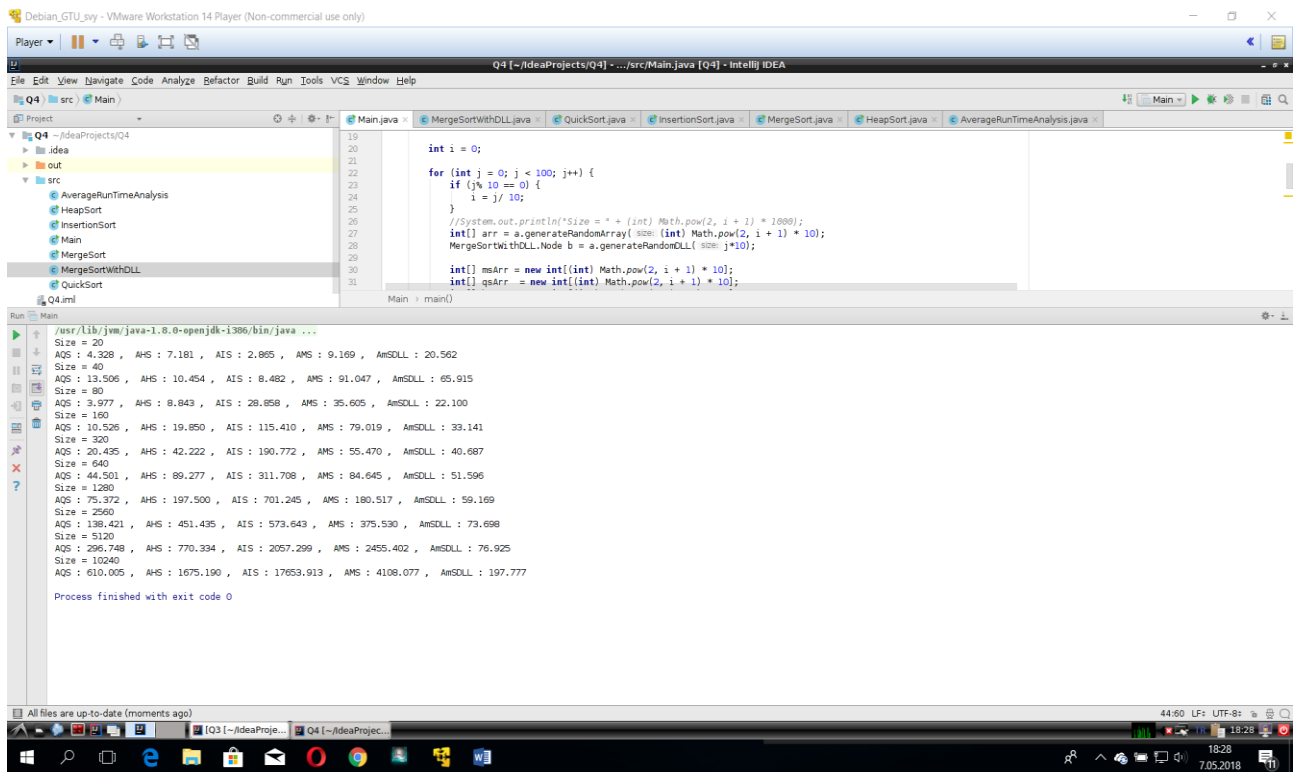
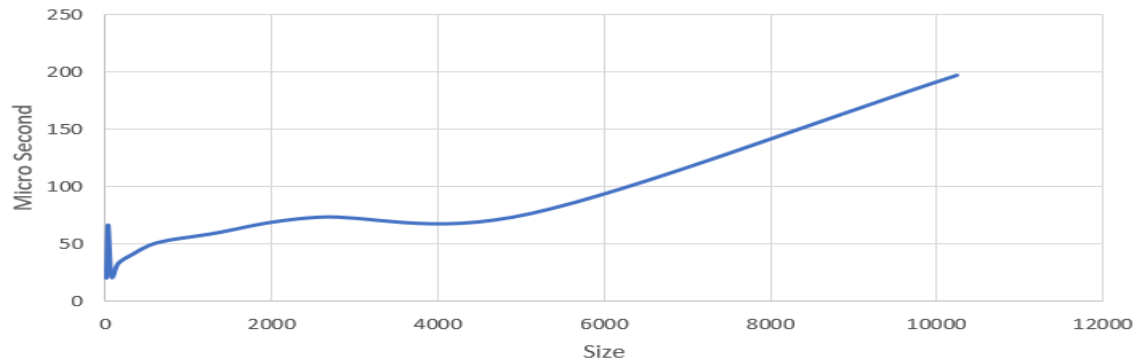
Size = 1280 => 59.169 micro saniye (10 tane testin ortalaması)

Size = 2560 => 73.698 micro saniye (10 tane testin ortalaması)

Size = 5120 => 76.925 micro saniye (10 tane testin ortalaması)

Size = 10240 => 197.777 micro saniye (10 tane testin ortalaması)

Merge Sort With DLL



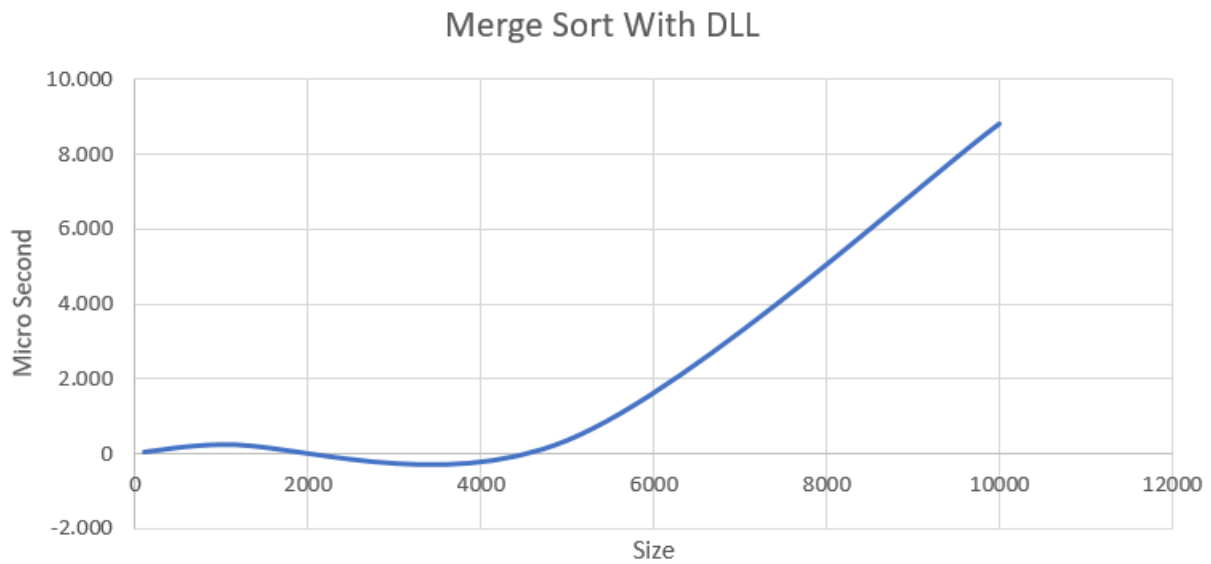
3.1.3 Worst-case Performance Analysis

Size = 100 => 51,239 micro saniye

Size = 1000 => 252,704 micro saniye

Size = 5000 => 365,777 micro saniye

Size = 10000 => 8818,719 micro saniye

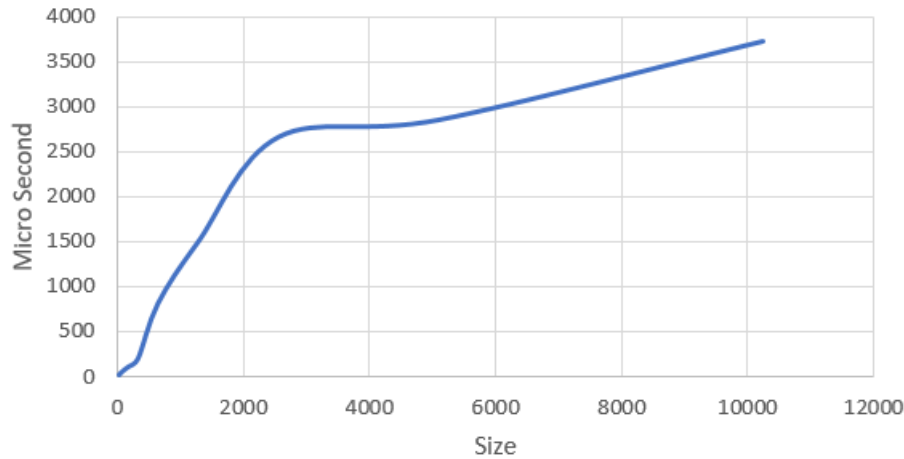


3.2 MergeSort

3.2.1 Average Run Time Analysis

Size = 20 => 21.377 micro saniye (10 tane testin ortalaması)
Size = 40 => 36.357 micro saniye (10 tane testin ortalaması)
Size = 80 => 64.279 micro saniye (10 tane testin ortalaması)
Size = 160 => 108.922 micro saniye (10 tane testin ortalaması)
Size = 320 => 199.799 micro saniye (10 tane testin ortalaması)
Size = 640 => 816.748 micro saniye (10 tane testin ortalaması)
Size = 1280 => 1502.075 micro saniye (10 tane testin ortalaması)
Size = 2560 => 2665.389 micro saniye (10 tane testin ortalaması)
Size = 5120 => 2856.318 micro saniye (10 tane testin ortalaması)
Size = 10240 => 3725.569 micro saniye (10 tane testin ortalaması)

Merge Sort



```

Debian_GTU_Lvy - VMware Workstation 14 Player (Non-commercial use only)
Player
Q4 [-IdeaProjects/Q4] - .../src/Main.java [Q4] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Main.java MergeSortWithDLL.java QuickSort.java InsertionSort.java MergeSort.java HeapSort.java AverageRunTimeAnalysis.java
1 import com.sun.xml.internal.ws.api.model.MEP;
2
3 public class Main {
4     public static void main(String[] args) {
5         AverageRunTimeAnalysis a = new AverageRunTimeAnalysis();
6         MergeSort ms = new MergeSort();
7         QuickSort qs = new QuickSort();
8         HeapSort hs = new HeapSort();
9         InsertionSort is = new InsertionSort();
10
11         double[] MergeSortTimes = new double[100];
12         double[] MergeSortDLLTimes = new double[100];
13
14         Main : main()
15
16 Run Main
17 /usr/lib/jvm/java-1.8.0-openjdk-1.8.0/bin/java ...
18 Size = 20
19 AQS : 17.075 , AHS : 22.728 , AIS : 10.462 , AMS : 21.377 , AmSDLL : 23.102
20 Size = 40
21 AQS : 21.423 , AHS : 23.764 , AIS : 19.873 , AMS : 36.357 , AmSDLL : 23.508
22 Size = 80
23 AQS : 29.633 , AHS : 45.458 , AIS : 37.679 , AMS : 64.279 , AmSDLL : 30.721
24 Size = 160
25 AQS : 930.884 , AHS : 88.630 , AIS : 74.503 , AMS : 108.922 , AmSDLL : 38.591
26 Size = 320
27 AQS : 61.342 , AHS : 753.792 , AIS : 730.792 , AMS : 199.799 , AmSDLL : 54.723
28 Size = 640
29 AQS : 424.540 , AHS : 428.123 , AIS : 8405.236 , AMS : 816.748 , AmSDLL : 369.538
30 Size = 1280
31 AQS : 1934.604 , AHS : 1319.035 , AIS : 18271.603 , AMS : 1502.075 , AmSDLL : 610.747
32 Size = 2560
33 AQS : 295.045 , AHS : 2668.684 , AIS : 6208.936 , AMS : 2665.389 , AmSDLL : 79.316
34 Size = 5120
35 AQS : 599.286 , AHS : 4147.945 , AIS : 22172.741 , AMS : 2856.318 , AmSDLL : 92.868
36 Size = 10240
37 AQS : 1252.772 , AHS : 2502.522 , AIS : 82569.822 , AMS : 3725.569 , AmSDLL : 108.851
38
39 Process finished with exit code 0
  
```

3.2.2 Worst-case Performance Analysis

SIZE : 100

Worst Case Merge Sort : 255.916

SIZE : 1000

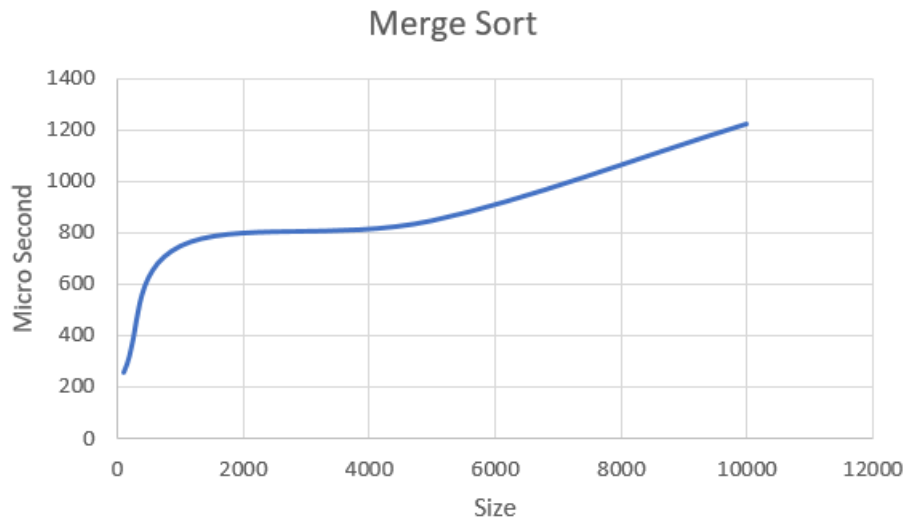
Worst Case Merge Sort : 748.375

SIZE : 5000

Worst Case Merge Sort : 847.684

SIZE : 10000

Worst Case Merge Sort : 1224.95



3.3 Insertion Sort

3.3.1 Average Run Time Analysis

Size = 20 => 14.208 micro saniye (10 tane testin ortalaması)
Size = 40 => 20.689 micro saniye(10 tane testin ortalaması)
Size = 80 => 21.972 micro saniye (10 tane testin ortalaması)
Size = 160 => 59.645 micro saniye (10 tane testin ortalaması)
Size = 320 => 1440.621 micro saniye (10 tane testin ortalaması)
Size = 640 => 5708.293 micro saniye (10 tane testin ortalaması)
Size = 1280 => 914.401 micro saniye (10 tane testin ortalaması)
Size = 2560 => 7460.527 micro saniye (10 tane testin ortalaması)
Size = 5120 => 23944.285 micro saniye (10 tane testin ortalaması)
Size = 10240 => 61323.097 micro saniye (10 tane testin ortalaması)




```

40 // Split table into halves
41 int halfSize = table.length / 2;
42 T[] leftTable = (T[])new Comparable[halfSize];
43 T[] rightTable = (T[])new Comparable[table.length - halfSize];
44 System.arraycopy(table, 0, leftTable, 0, halfSize);
45 System.arraycopy(table, halfSize, rightTable, 0,
46                 table.length - halfSize);
47
48 // Sort the halves.
49 sort(leftTable);
50 sort(rightTable);
51 // Merge the halves.
52
53

```

Run Main

```

/usr/lib/jvm/java-1.8.0-openjdk-1386/bin/java ...
Size = 20
AQS : 13.864 , AHS : 18.152 , AIS : 14.208 , AMS : 18.666 , AmSOLL : 21.384
Size = 40
AQS : 61.517 , AHS : 23.608 , AIS : 20.689 , AMS : 32.801 , AmSOLL : 211.023
Size = 80
AQS : 13.711 , AHS : 31.554 , AIS : 21.972 , AMS : 23.449 , AmSOLL : 611.331
Size = 160
AQS : 17.775 , AHS : 74.635 , AIS : 59.646 , AMS : 31.603 , AmSOLL : 29.008
Size = 320
AQS : 400.387 , AHS : 369.652 , AIS : 1440.621 , AMS : 67.387 , AmSOLL : 39.241
Size = 640
AQS : 329.268 , AHS : 395.440 , AIS : 5708.293 , AMS : 431.466 , AmSOLL : 41.923
Size = 1280
AQS : 226.689 , AHS : 341.237 , AIS : 914.401 , AMS : 279.073 , AmSOLL : 53.942
Size = 2560
AQS : 247.341 , AHS : 1423.040 , AIS : 7460.527 , AMS : 1478.950 , AmSOLL : 67.074
Size = 5120
AQS : 623.735 , AHS : 2289.097 , AIS : 23944.285 , AMS : 4436.883 , AmSOLL : 73.574
Size = 10240
AQS : 977.584 , AHS : 1523.901 , AIS : 61323.097 , AMS : 2009.781 , AmSOLL : 79.315
Process finished with exit code 0

```

3.3.2 Worst-case Performance Analysis

SIZE : 100

Worst Case Insertion Sort : 915.651

SIZE : 1000

Worst Case Insertion Sort : 46272.134

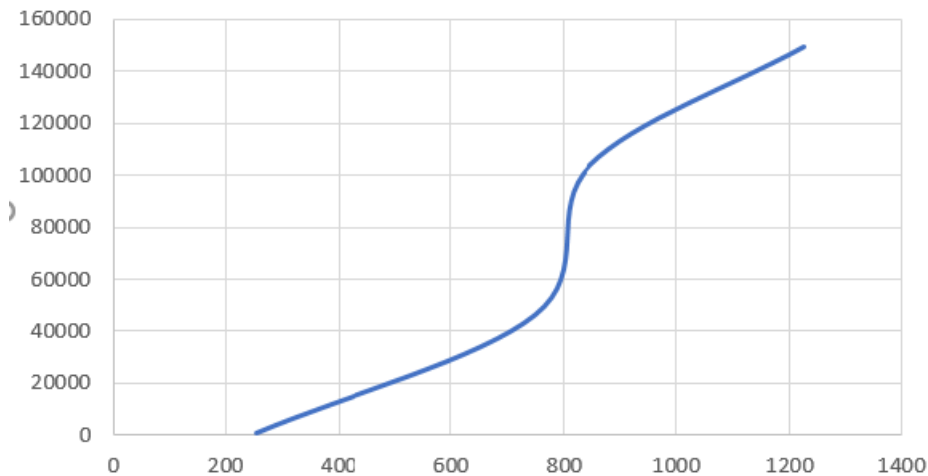
SIZE : 5000

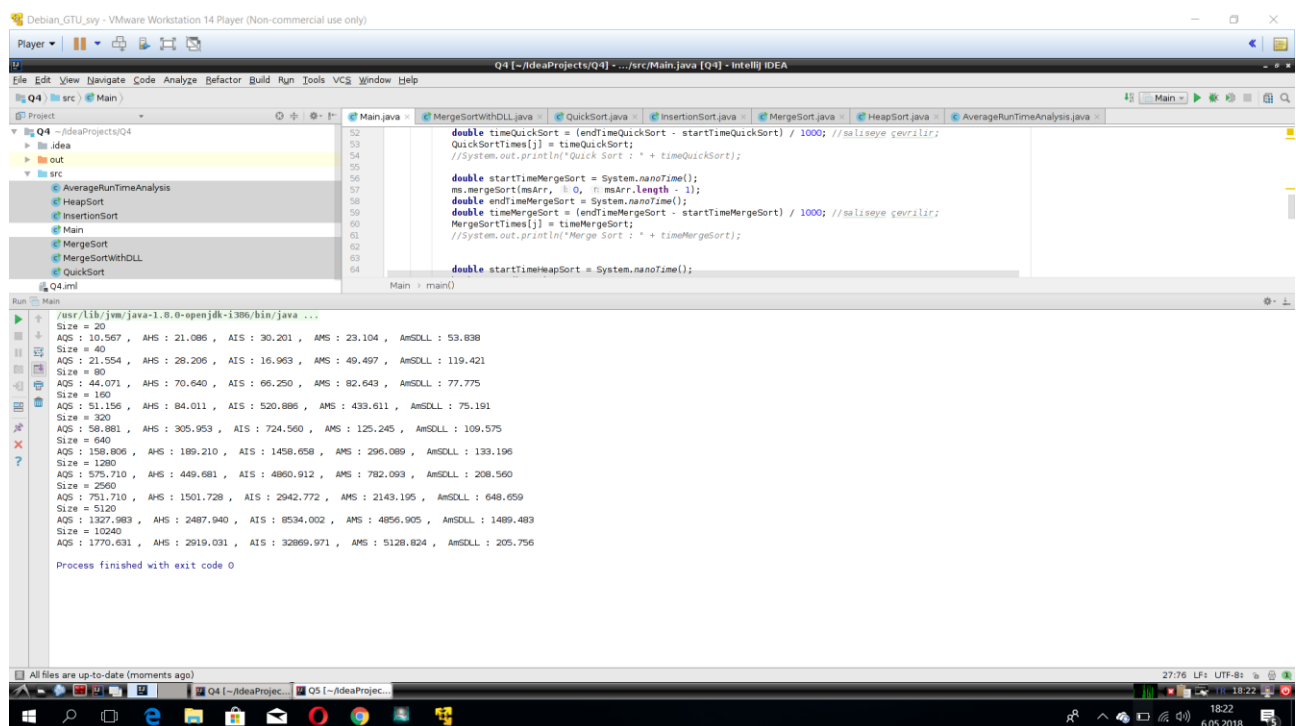
Worst Case Insertion Sort : 103901.391

SIZE : 10000

Worst Case Insertion Sort : 149353.707

Grafik Başlığı





3.4.2 Worst-case Performance Analysis

SIZE : 100

Worst Case Quick Sort : 433.821

SIZE : 1000

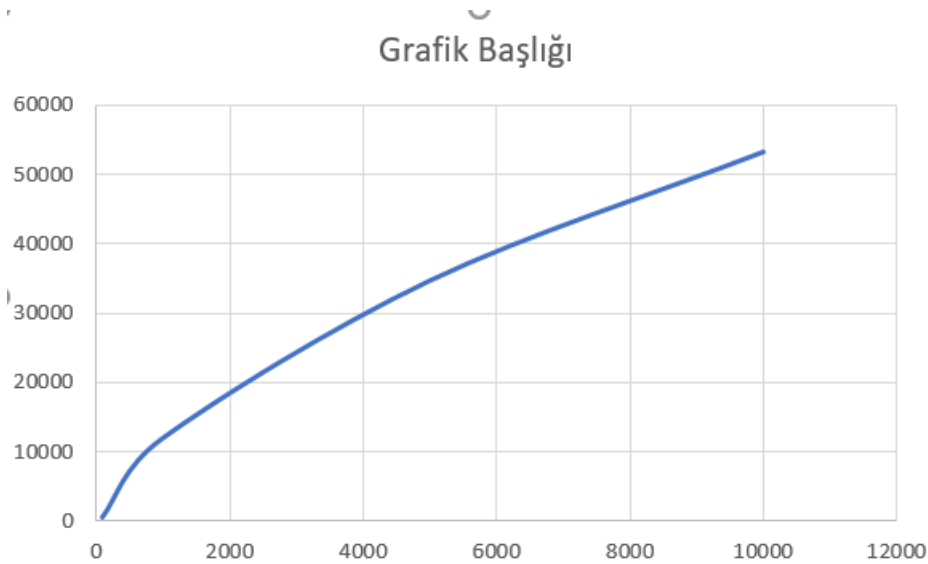
Worst Case Quick Sort : 11832.456

SIZE : 5000

Worst Case Quick Sort : 34615.998

SIZE : 10000

Worst Case Quick Sort : 53254.557



3.5 Heap Sort

3.5.1 Average Run Time Analysis

Size = 20 => 21.086 micro saniye (10 tane testin ortalaması)

Size = 40 => 28.206 micro saniye (10 tane testin ortalaması)

Size = 80 => 70.640 micro saniye (10 tane testin ortalaması)

Size = 160 => 84.011 micro saniye (10 tane testin ortalaması)

Size = 320 => 305.953 micro saniye (10 tane testin ortalaması)

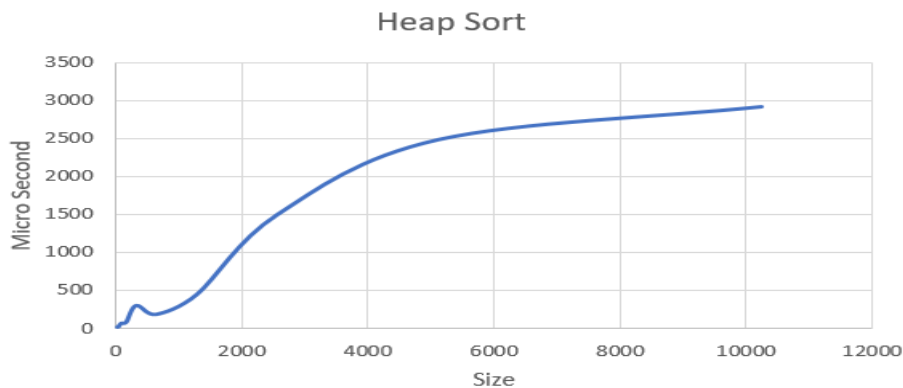
Size = 640 => 189.210 micro saniye (10 tane testin ortalaması)

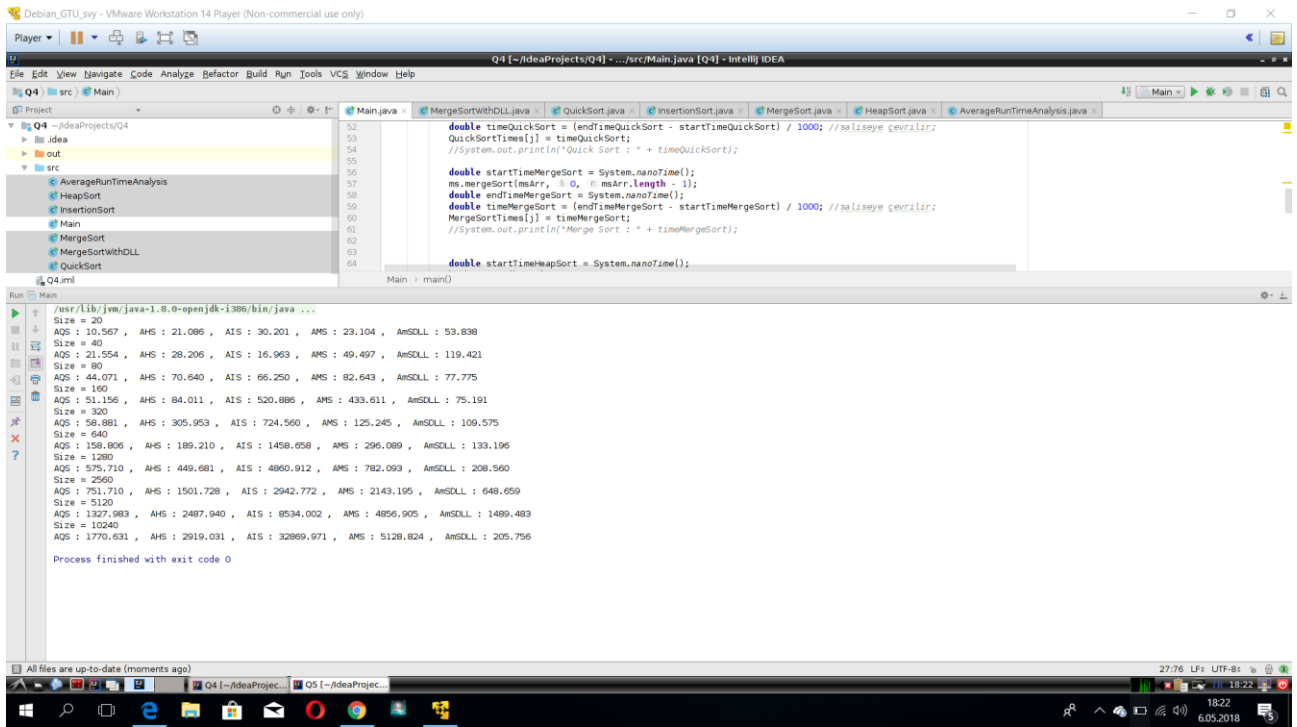
Size = 1280 => 449.681 micro saniye (10 tane testin ortalaması)

Size = 2560 => 1501.728 micro saniye (10 tane testin ortalaması)

Size = 5120 => 2487.940 micro saniye (10 tane testin ortalaması)

Size = 10240 => 2919.031 micro saniye (10 tane testin ortalaması)





3.5.2 Wort-case Performance Analysis

SIZE : 100

Worst Case Heap Sort :62.93

SIZE : 1000

Worst Case Heap Sort : 140.73

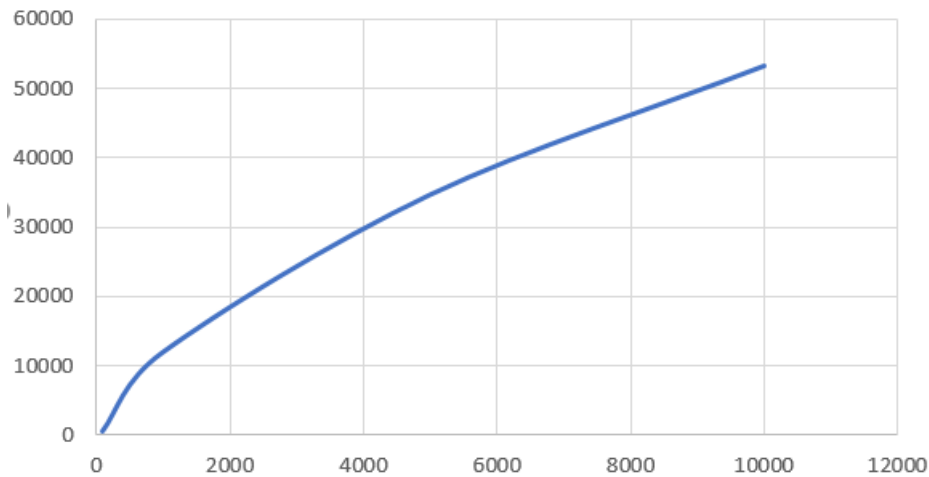
SIZE : 5000

Worst Case Heap Sort : 500.613

SIZE : 10000

Worst Case Heap Sort : 563.837

Grafik Başlığı



4 Comparison the Analysis Results

