Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

EFKAN DURAKLI
161044086

Course Assistant: Fatma Nur ESİRCİ

1 Double Hashing Map

Bu bölümde open adressing kullanılarak hash map implement edildi. Colizyonlar double hashing yöntemi kullanılarak çözüldü.

1.1 Pseudocode and Explanation

Kitaptaki hash map örneğinde colizyonlar lineer probing kullanılarak çözülmüş. Bu hash mapte colizyonlar double hashing kullanılarak çözüldü.Kitaptaki örnek ten farklı olarak sadece find metodu değişiyor. Find metodununun pseudocode'u aşağıdaki gibidir.

```
find(key)
set index to key.hasCode() % table.length
if (index < 0)
    add index to table.length
set i to 1
while (table[index] =! null && key != table[index]
    set index to key.hashCode() + i*hashCode2(key) % table.length
    if (index < 0)
        add index to table.length
    i++
return index
```

Double hashing yönteminde collision olduğunda ikinci bir hascode kullanılarak yeni index hesaplanır. İkinci hash code'un fomülü şu şekildedir.
hashCode2(key) = PRIME – (key.hashCode() % PRIME)
Buradaki PRIME hash table'ın boyutundan küçük en büyük asal sayıdır.

1.2 Test Cases

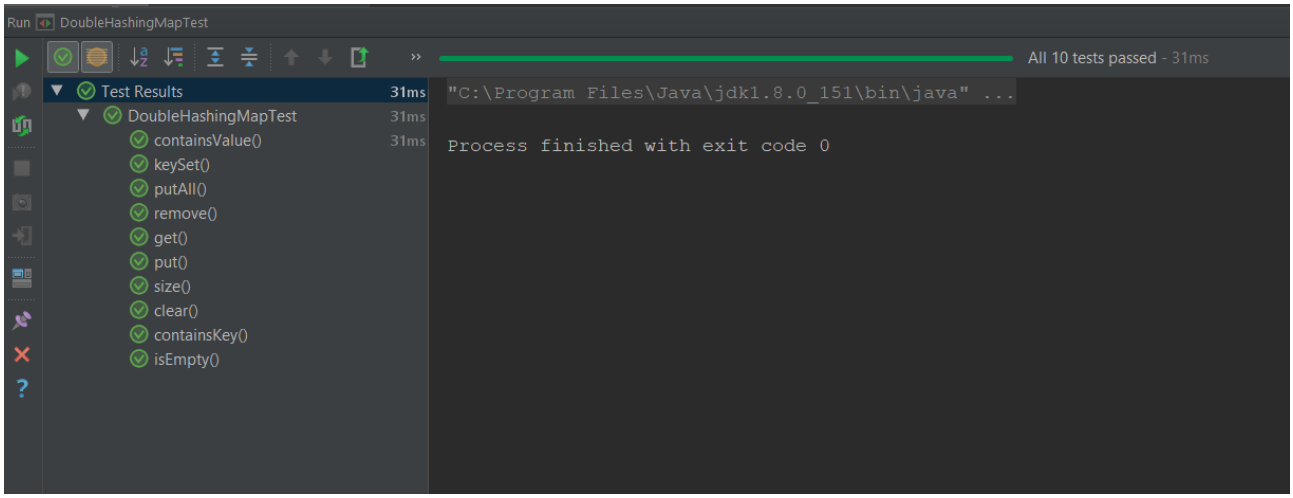
Test Case 1
Bu testte büyüklüğü 11 olan bir hash table kullanıldı. Key olarak öğrenci numaraları value olarak öğrenci isimleri kullanıldı. Rehash fonksiyonunun test edilebilmesi için 11 den fazla eleman eklendi.
Yazılan bütün metodlar test edildi.Colizyon oluşturan durumlar test edildi. Bu testin ekran görüntüsü aşağıdaki gibidir.

```
----DOUBLE HASHING MAP TEST 1 ----
--------------------------------------------------
After adding element to map
{151044011=Zeynep, 151044043=Yusuf, 131044010=Taip, 151044095=Yusuf, 161044042=Yasir, 181041038=Zeyneb, 91044066=Suleyman, 141044094=Taalai, 111044074=Sinan,
Size of map = 14
--------------------------------------------------
After removing 151044011=Zeynep
{151044043=Yusuf, 131044010=Taip, 151044095=Yusuf, 161044042=Yasir, 181041038=Zeyneb, 91044066=Suleyman, 141044094=Taalai, 111044074=Sinan, 131044056=Taner,
Size of map = 13
--------------------------------------------------
Values of map
[Yusuf, Taip, Yusuf, Yasir, Zeyneb, Suleyman, Taalai, Sinan, Taner, Simge, Yavuz, Tarık, Yunus]
--------------------------------------------------
Key set of map
[141044080, 151044035, 151044043, 131044010, 91044066, 151044010, 141044094, 161044042, 151044095, 111044074, 131044056, 131044058, 181041038]
--------------------------------------------------
get(141044080) = Yunus
Map contains key 111044074
Map does not contain key 161044086
Map contains value Sinan
Map does not contain value Efkan
After clearing map
{}
```

Test Case 2
Bu testte büyüklüğü 17 olan bir hash table kullanıldı. Key olarak şehir isimleri value olarak nüfuslar kullanıldı. Rehash fonksiyonunun test edilebilmesi için 17 den fazla eleman eklendi.
Yazılan bütün metodlar test edildi.Colizyon oluşturan durumlar test edildi. Bu testin ekran görüntüsü aşağıdaki gibidir.

```
----DOUBLE HASHING MAP TEST 2 ----
--------------------------------------------------
After adding element to map
{Hakkari=275761, Kayseri=1376722, Edirne=433830, Bayburt=80417, Trabzon=786326, Aksaray=402404, Kilis=136319, Denizli=1018735, Ankara=5445026, Sivas=621301, I
Size of map = 20
--------------------------------------------------
After removing Aksaray=402404 and Edirne=433830
{Hakkari=275761, Kayseri=1376722, Bayburt=80417, Trabzon=786326, Kilis=136319, Denizli=1018735, Ankara=5445026, Sivas=621301, Istanbul=15029231, Siirt=324394,
Size of map = 18
--------------------------------------------------
Values of map
[275761, 1376722, 80417, 786326, 136319, 1018735, 5445026, 621301, 15029231, 324394, 2005515, 585252, 246672, 602086, 372373, 231511, 1413041, 433830]
--------------------------------------------------
Key set of map
[Hakkari, Trabzon, Batman, Kilis, Tokat, Sivas, Manisa, Kayseri, Kastamonu, Gaziantep, Bayburt, Siirt, Karaman, Isparta, Ankara, Denizli, Istanbul, Erzincan]
--------------------------------------------------
Map contains key Erzincan
Map does not contain key Erzurum
Map contains value 433830
Map does not contain value 1000000
After clearing map
{}
```

Unit Test
Bütün metodlar için unit test yapıldı. Bütün testler başarılı bir şekilde gerçekleştirildi. Test sonuçlarının ekran görüntüsü aşağıdaki gibidir.



2 Recursive Hashing Set

Bu bölümde hash set implement edildi. Colizyonlar chaining yöntemiyle çözüldü. Colizyon olduğu zaman yeni bir hash table oluşturulur. Bu yeni oluşturulan tablonun büyüklüğü eski tablonun büyüklüğünden küçük en büyük asal sayı seçilir.

2.1 Pseudocode and Explanation

Bu map için add, remove, contains, size ve isEmpty metodları implement edildi.Add metodu recursive yazıldı.Bu yüzden add metodu için private helper metot yazıldı. Add,remove ve contains metodlarının pseudocode'ları aşağıdaki gibidir.

add(table, element)

```
set index to key.hasCode() % table.length
if (index < 0)
    add index to table.length
if (table[index].getElement() == null)
    set table[index].element to element
    size++
    return true
if (table[index].getElement() == element)
   return false
else
   if (nextTable == null)
       set capacity of next table
   return add(table[index].nextTable, element
remove(element)
set index to key.hasCode() % table.length
if (index < 0)
    add index to table.length
set current to table[index]
while (currant != null)
      if (curent.getElement() != null && current.getElement() == element)
          set current.element to null
          decrement size 1
          return true
      if (current.nextTable != null)
          set index to key.hasCode() % table.netTable.length
          current = curent.nextTable[index]
      else
          break
return false
```

contains(element)

```
set index to key.hasCode() % table.length
if (index < 0)
    add index to table.length
set current to table[index]
while (currant != null)
      if (curent.getElement() != null && current.getElement() == element)
          return true
      if (current.nextTable != null)
          set index to key.hasCode() % table.netTable.length
          current = curent.nextTable[index]
      else
          break
return false
```

2.2 Test Cases

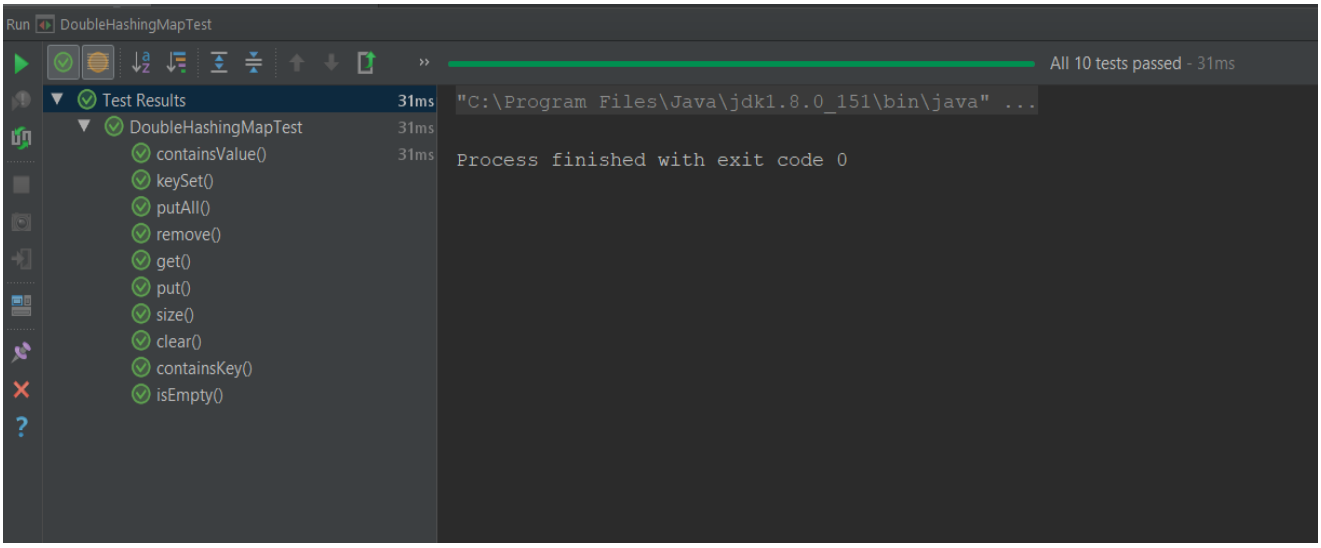
Test Case 1
Bu testte büyüklüğü 53 olan bir hash table kullanıldı. Eleman olarak integer sayılar eklendi. Colizyon oluşturan durumlar test edildi. Bu testin ekran görüntüsü aşağıdaki gibidir.

```
----- RECURSIVE HASH SET TEST 1 -----
Set is empty
After adding element to set
[0, 53, 106, 212, 1, 108, 2, 55, 178, 126, 400, 350, 300, 250, 200, 202, 150, 100, 632, 50]
Size of set = 20
Set is not empty
After removing element 53 and 100
[0, 106, 212, 1, 108, 2, 55, 178, 126, 400, 350, 300, 250, 200, 202, 150, 632, 50]
Size of set = 18
Set contains element 106
Set does not contain element 12356
```

Test Case 2
Bu testte büyüklüğü 13 olan bir hash table kullanıldı. Setin elemanları string tipinde meyvelerdir. Colizyon oluşturan durumlar test edildi. Bu testin ekran görüntüsü aşağıdaki gibidir.

```
----- RECURSIVE HASH SET TEST 2 -----
Set is empty
After adding element to set
[Papaya, Watermelon, Orange, Kiwifruit, Strawberries, Nectarine, Pomegranate, Grapefruit, Cherries, Onions, Mushrooms, Apricots, Pear, Apple, Carrots, Lemon,
Size of set = 20
Set is not empty
After removing element Apple, Onions, Watermelon
[Papaya, Orange, Kiwifruit, Strawberries, Nectarine, Pomegranate, Grapefruit, Cherries, Mushrooms, Apricots, Pear, Carrots, Lemon, Grapes, Banana, Mandarin,
Size of set = 17
Set contains element Orange
Set does not contain element Salad greens
```

Unit Test
Bütün metodlar için unit test yapıldı.Bütün testler başarılı bir şekilde gerçekleştirildi.Test sonuçlarının ekran görüntüsü aşağıdaki gibidir.



3 Sorting Algortihms

3.1 MergeSort with DoubleLinkedList

Bu bölümde merge sort algoritması java'nın ikili bağlı listesi kullanılarak gerçekleşitrildi. Bu algoritmanın zaman analizini yapacak olursak;
$T_{sort}(n) = T_{sort}(n/2) + T_{sort}(n/2) + T_{merge}(n) = T_{sort}(n/2) + T_{sort}(n/2) + O (n^2)$

$T_{sort}(n) = O (n^2.\log(n))$

$T_{merge}(n) = O (n^2)$ (Bu algoritmayı gerçeklerken, listeye elemanları eklemek için liste baştan sona gezilir. Her döngüde listenin elemanlarına get metoduyla erişildiği için merge işleminin çalışma zamanı $O(n^2)$ olur.)

3.1.1 Pseudocode and Explanation

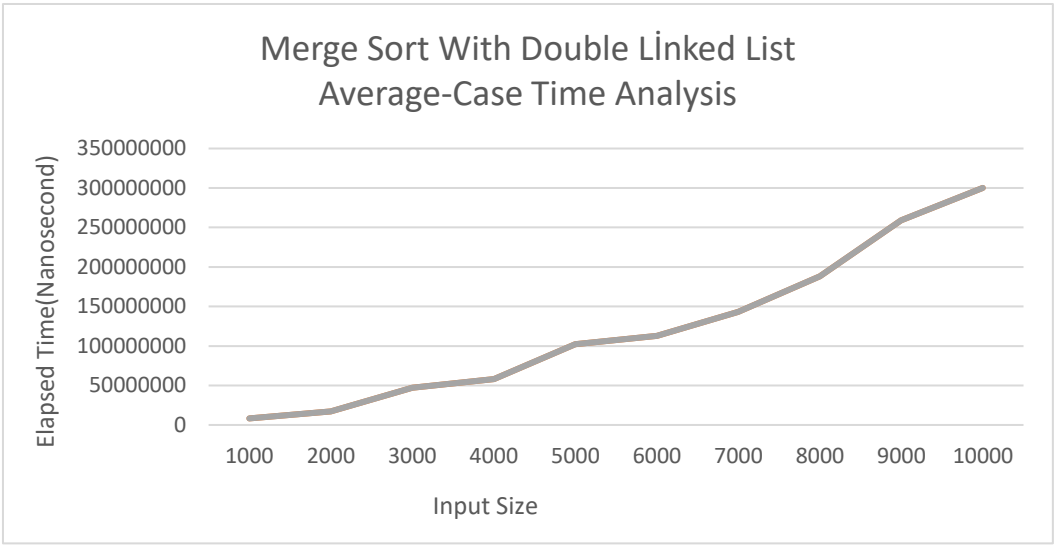
```
MergeSort ( list )
        if ( list.size > 1 )
           leftListSize = list.size / 2
           rightListSize = list.size – leftListSize
           leftList = new List
           rightList = new List
           for i = 0 to leftListSize
              add leftList to list[i]
           for i = 0 to rightListSize
              add rightList to list[ leftListSize + i ]
           MergeSort ( leftList )
           MergeSort ( rightList )
           Merge ( list, leftList, rightList )

Merge (  destinationList, leftList, rightList )
         i = 0
        j = 0
        k = 0
        while (   i < leftList.size && j < rightList.size )
            if ( leftList[i] < rigthList[j] )
                set destinationList[k] to leftList[i]
                i++
            else
                set destinationList[k] to rightList[j]
                j++
            k++
        if ( i >= leftList.size )
                while ( j < rightList.size )
                    set destinationList[k] to rightList[j]
                    j++
        else
                while ( j < leftList.size )
                    set destinationList[k] to leftList[i]
                    i++
```

3.1.2 Average Run Time Analysis

Avarage run time'ı bulmak için 10 farlı boyutta random liste oluşturuldu. Her boyuttaki liste için 10 farlı deneme yapıldı ve bu denemelerin ortalamaları alındı. Aşağıda sonuçları görebilirsiniz.

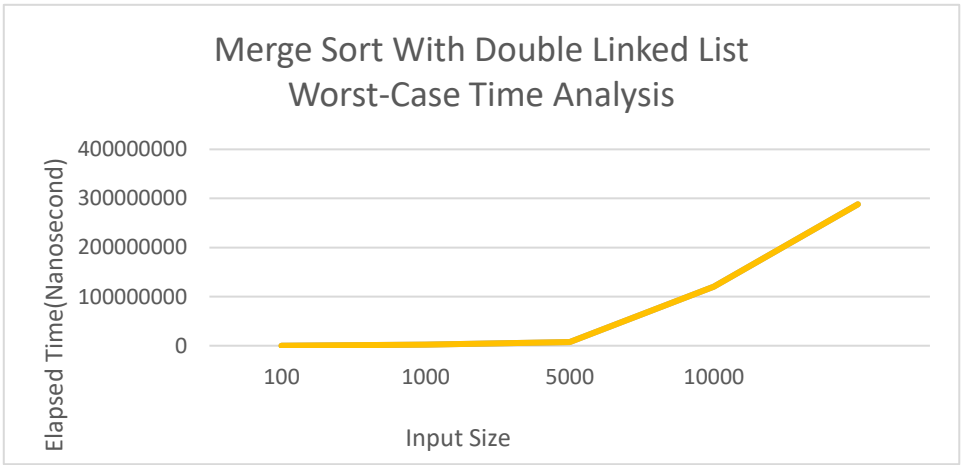
Size	Average Elapsed Time
1000	8285130
2000	17192663
3000	47032984
4000	57951370
5000	102408285
6000	112916717
7000	143203242
8000	188063789
9000	258979827,4
10000	300118560,8



3.1.3 Wort-case Performance Analysis

Worst-case için farklı boyutlarda random listeler oluşturuldu. Bu listeler tersten sıralandı. Tersten sıralanmış listeler test edilerek sonuçlar kaydedildi. Sonuçları aşağıda görebilirsiniz.

Input Size	Merge Sort With Dll
100	2082615
1000	7248715
5000	120227187
10000	288032662



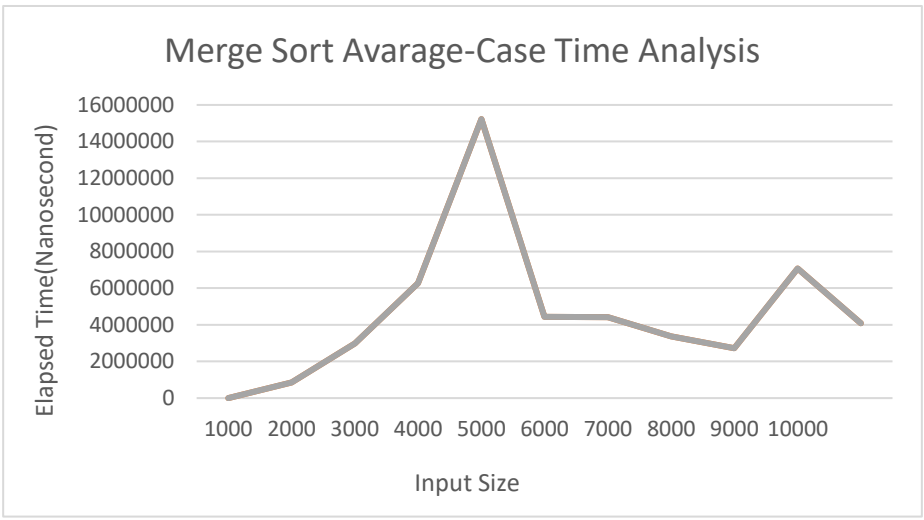
3.2 MergeSort

Merge Sort algoritması kitaptan alınarak average ve worst-case için test edildi.

3.2.1 Average Run Time Analysis

Average case için 10 faklı boyutta random array oluşturuldu. Her oluşturulan array için 10 farklı kere deneme yapıldı ve bu denemelerin ortalamaları alınarak grafik oluşturuldu. Sonuçları ve grafiği aşağıda görebilirsiniz.

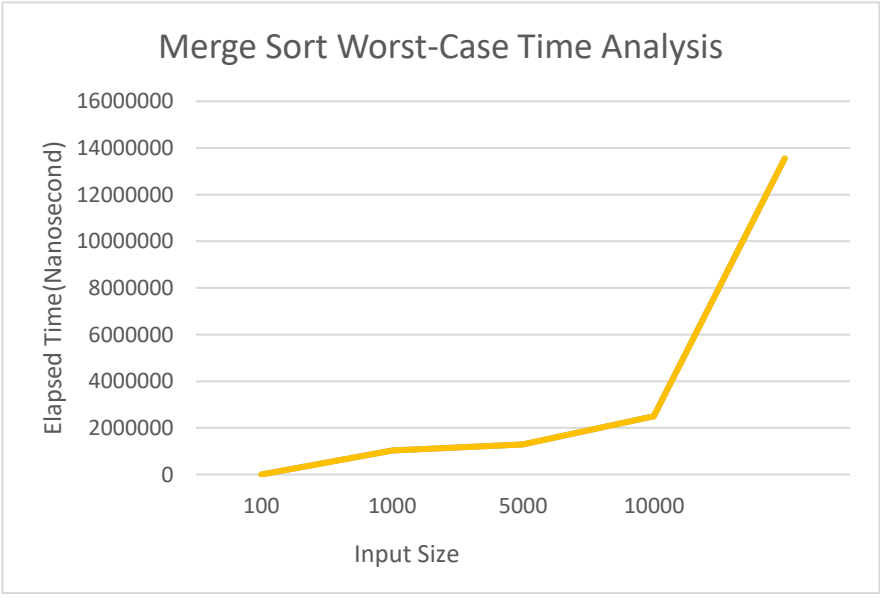
Size	Average Elapsed Time
1000	851806
2000	2986187
3000	6263932
4000	15224834
5000	4432480
6000	4419508
7000	3373398
8000	2719220
9000	7080556
10000	4082710



3.2.2 Wort-case Performance Analysis

Worst-case için farklı boyutlarda random arrayler oluşturuldu. Bu listeler tersten sıralandı. Tersten sıralanmış listeler test edilerek sonuçlar kaydedildi. Sonuçları aşağıda görebilirsiniz.

Input Size	Merge Sort
100	1029607
1000	1284135
5000	2502176
10000	13548286



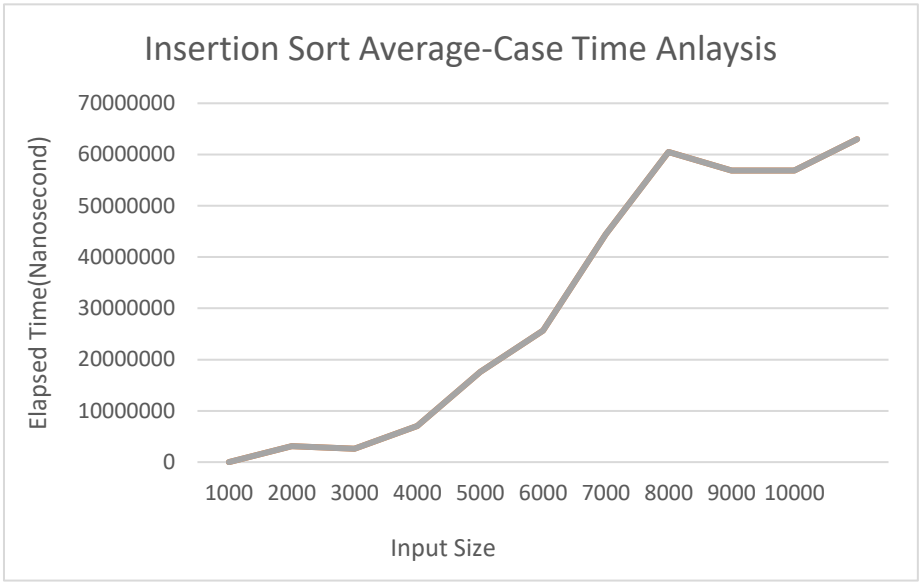
3.3 Insertion Sort

Insertion Sort algoritması kitaptan alınarak average ve worst-case için test edildi.

3.3.1 Average Run Time Analysis

Average case için 10 faklı boyutta random array oluşturuldu. Her oluşturulan array için 10 farklı kere deneme yapıldı ve bu denemelerin ortalamaları alınarak grafik oluşturuldu. Sonuçları ve grafiği aşağıda görebilirsiniz.

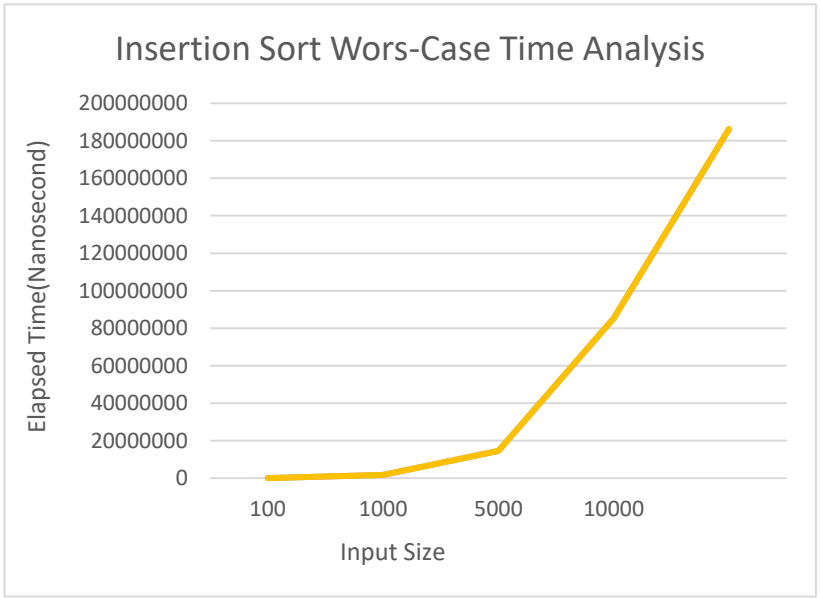
Size	Average Elapsed Time
1000	3112301,9
2000	2636088,3
3000	7072304,9
4000	17620517
5000	25650175
6000	44462211,4
7000	60503094,8
8000	56855750,6
9000	56855750,6
10000	62992542,4



3.3.2 Wort-case Performance Analysis

Worst-case için farklı boyutlarda random arrayler oluşturuldu. Bu listeler tersten sıralandı. Tersten sıralanmış listeler test edilerek sonuçlar kaydedildi. Sonuçları aşağıda görebilirsiniz.

Input Size	Insertion Sort
100	1675370
1000	14575431
5000	85316189
10000	186175428



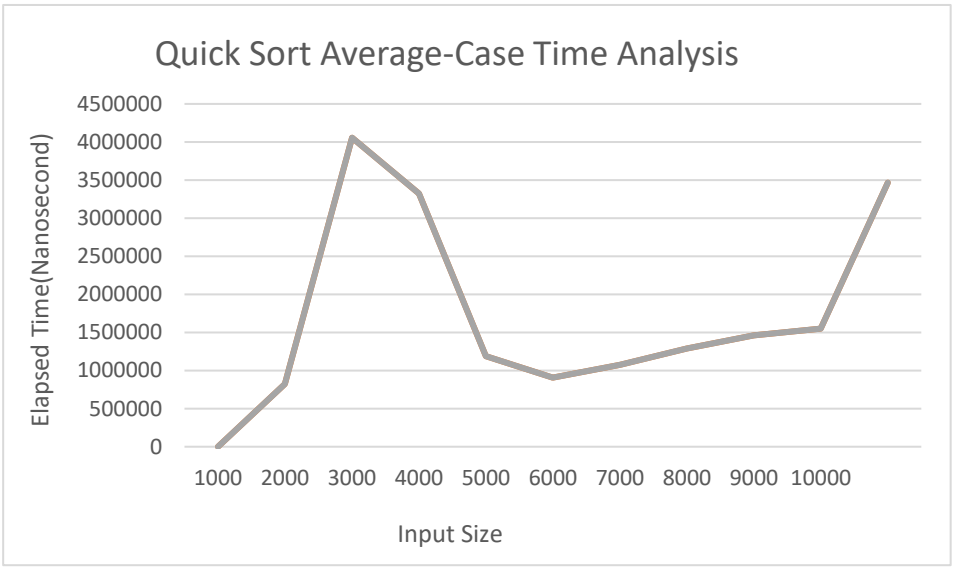
3.4 Quick Sort

Quick Sort algoritması kitaptan alınarak average ve worst-case için test edildi.

3.4.1 Average Run Time Analysis

Average case için 10 faklı boyutta random array oluşturuldu. Her oluşturulan array için 10 farklı kere deneme yapıldı ve bu denemelerin ortalamaları alınarak grafik oluşturuldu. Sonuçları ve grafiği aşağıda görebilirsiniz.

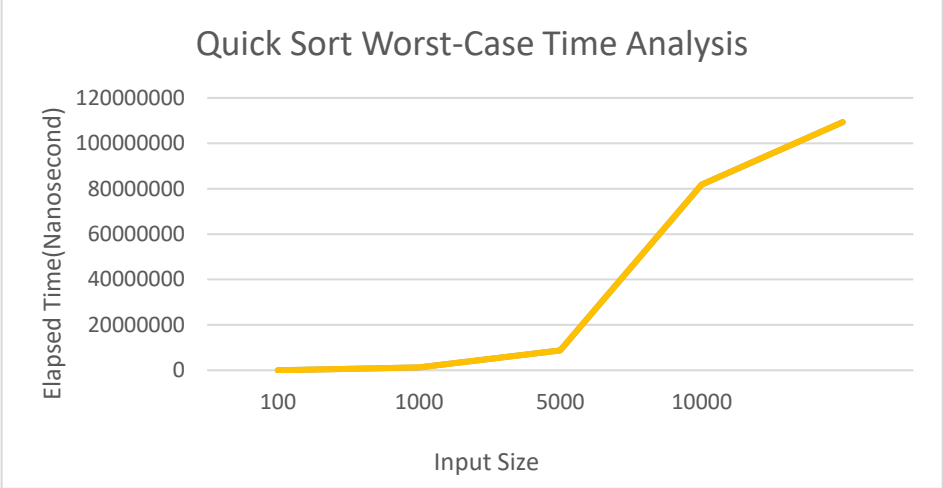
Size	Average Elapsed Time
1000	823438
2000	4056066
3000	3324216
4000	1186551
5000	907022
6000	1073984
7000	1288691
8000	1461236
9000	1550362
10000	3464781



3.4.2 Wort-case Performance Analysis

Worst-case için farklı boyutlarda random arrayler oluşturuldu. Bu listeler tersten sıralandı. Tersten sıralanmış listeler test edilerek sonuçlar kaydedildi. Sonuçları aşağıda görebilirsiniz.

Input Size	Quick Sort
100	1302199
1000	8797232
5000	81770859
10000	109316962



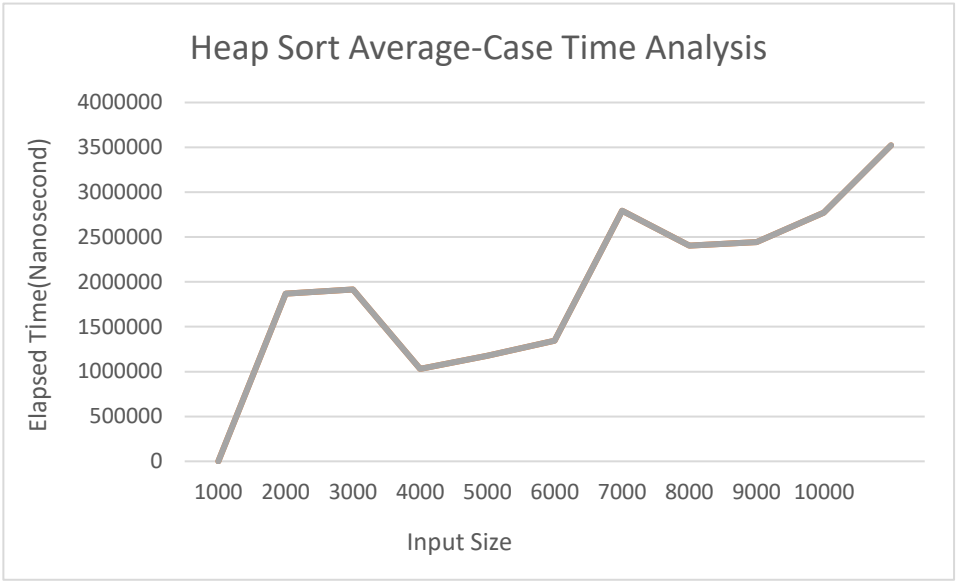
3.5 Heap Sort

Heap Sort algoritması kitaptan alınarak average ve worst-case için test edildi.

3.5.1 Average Run Time Analysis

Average case için 10 faklı boyutta random array oluşturuldu. Her oluşturulan array için 10 farklı kere deneme yapıldı ve bu denemelerin ortalamaları alınarak grafik oluşturuldu. Sonuçları ve grafiği aşağıda görebilirsiniz.

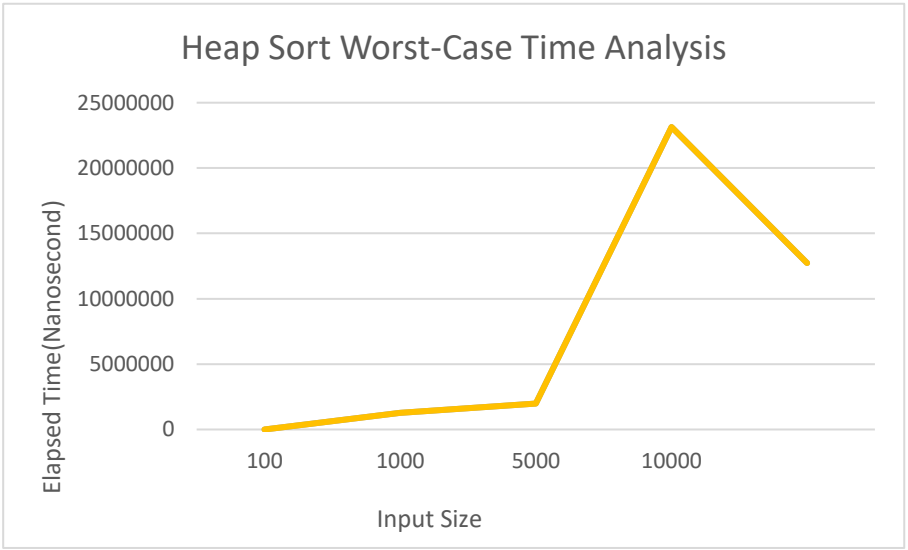
Size	Average Elapsed Time
1000	1869548
2000	1916636
3000	1032069
4000	1176904
5000	1343661
6000	2792499
7000	2405001
8000	2444658
9000	2773861
10000	3522830



3.5.2 Wort-case Performance Analysis

Worst-case için farklı boyutlarda random arrayler oluşturuldu. Bu listeler tersten sıralandı. Tersten sıralanmış listeler test edilerek sonuçlar kaydedildi. Sonuçları aşağıda görebilirsiniz.

Heap Sort	Input Size
1265251	100
1973004	1000
23136197	5000
12732154	10000



4 Comparison the Analysis Results

Beş adet sort algoritması en kötü durum için test edildi. En kötü durumu oluşturmak için şu yöntem izlendi. Öncelikle random integer array oluşturuldu. Bu array büyüten küçüğe sıralandı. Büyükten küğe sıralanmış array sort fonksiyonlarına gönderilerek sort edildi.Her bir sort algoritması için faklı boyutlarda inputlarala denendi. Çalışma zamanları kaydedilerek her bir algoritmanın grafiği oluşturuldu. Sonuçlar ve grafik aşağıdaki gibidir.

Input Size	Merge Sort	Heap Sort	Quick Sort	Insertion Sort	Mege Sort With Dll
100	1029607	1265251	1302199	1675370	2082615
1000	1284135	1973004	8797232	14575431	7248715
5000	2502176	23136197	81770859	85316189	120227187
10000	13548286	12732154	109316962	186175428	288032662