Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

AHMET ERGANİ
161044011

Course Assistant:Fatmanur Esirci

1 Double Hashing Map

This part about Question1 in HW5

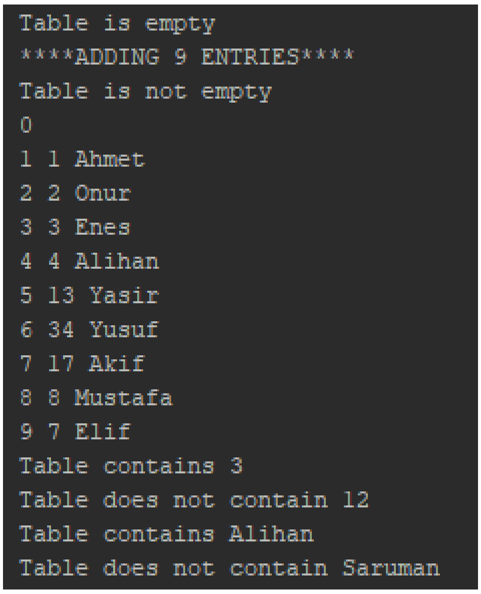
1.1 Pseudocode and Explanation

I held all the map entries in an array named table .I held size and capacity in class as fields and assigned starting capacity to 10. If the entry count exceeds (capacity -1) I resized the array. I also implemented Map.Entry to utilize Entry class better

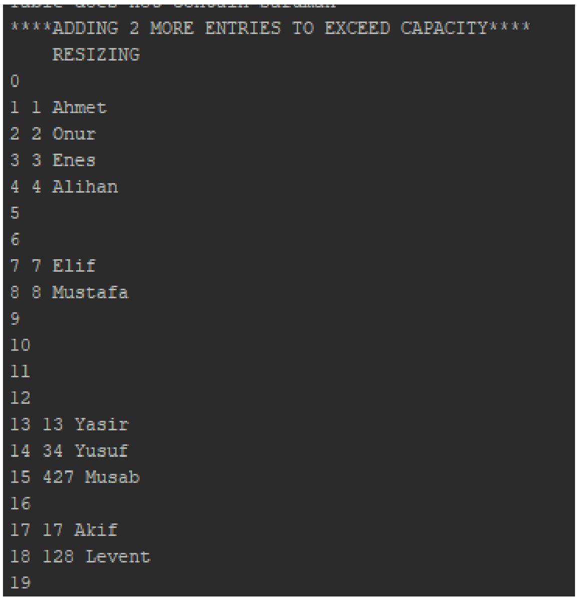
```
containsKey(key)       //Check all the table to find the given key
{
   for -> table size
      if(key exists)
         true
   false
}
containsKey(value)     // Check all the table to find the given value
{
   for -> table size
      if(value exists)
         true
   false
}
get(key)               // Checks the table like it is going to add this key to table till it finsd the Entry
                       // that has the given key
{
   if(table[key.hash].key == key
      return table[key.hash].value
   else
       key.hash2
       i = 1
       while
            if(table[hash1 + i * hash2].key == key)
                return table[hash1 + i * hash2].value
            i++
            if(i == size)
               return null
}
put(key, value)        //Creates the entry with given values and tries to add it to table[key.hashCode]
                       //Uses double hashing if a collision occurs till it inserts the Entry. Resizes if
                       //necessary
{
   if(table almost full)
      resize
   if(table[key.hash] == empty
      table[key.hash] = Entry(key,value)
   else
       key.hash2
       i = 1
```

```
        while
             if(table[hash1 + i * hash2] == empty)
                  table[hash1 + i * hash2] = Entry(key,value)
}
remove(key)             // Check all the table to find the address of given key. Deletes the Entry if it
                        // finds and adds every entry to table again. Returns null if it can't find it
{
   for -> size of table
      if(key exists)
          delete
           add every Entry again
      else
           return null
}
putAll(map)             //Creates a set of parameter's entries. Adds all of them to table using put()
{
   set = map.Entry
   for -> size of map
       this.put(set.key, set.value)
}
keySet()                //Creates a set. Searches the table. adds the keys of each non-empty index
{
   Set a
   for -> size of table
       if(not empty)
           a.add(table[i].key)
}
values()                //Creates an arrayList. Searches the table. adds the values of each non-empty
                        //index
{
   arrayList a
   for -> size of table
       if(not empty)
           a.add(table[i].value)
}
entrySet()              //Creates a set. Searches the table. adds the Entries of each non-empty index
{
   Set a
   for -> size of table
       if(not empty)
           a.add(table[i])
}
Resize(newTable)        //doubles the capacity. Creates new table. Add every existing entry to it
{
   Capacity = capacity * 2
   Size = 0
   For -> size of newTable
       if(newTable[i] not empty)
           put(newTable[i])
}
display()               //Print every index and if not null, key and value of the Entry in that index
{
   for -> size of table
       if(empty)
           print index \n
       else
           print index + table[i].key + table[i].value \n
}
```

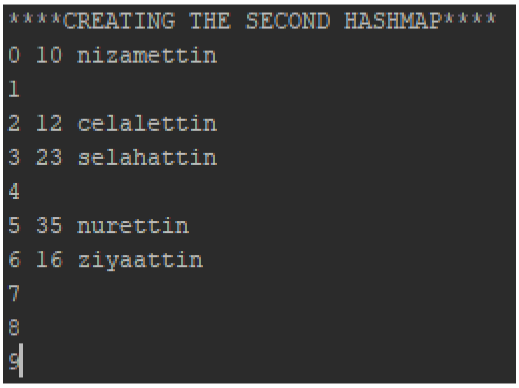
1.2 Test Cases



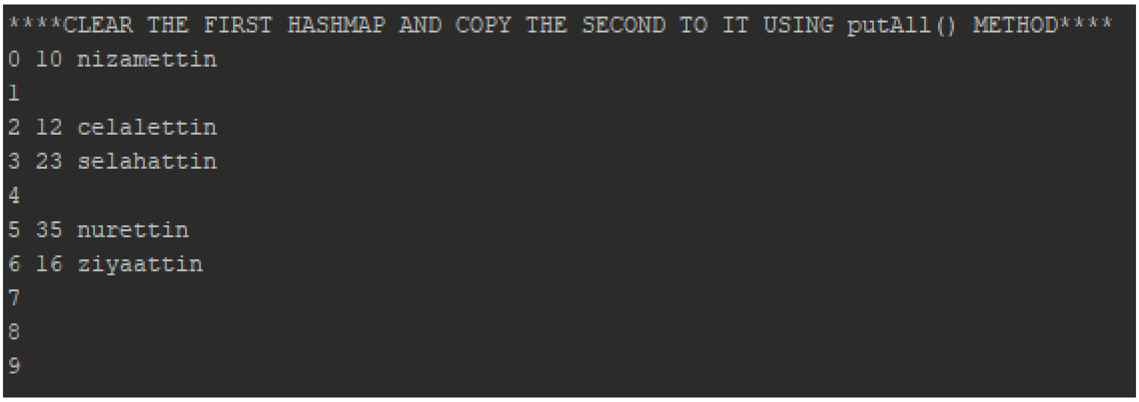
```
Table is empty
****ADDING 9 ENTRIES****
Table is not empty
0
1 1 Ahmet
2 2 Onur
3 3 Enes
4 4 Alihan
5 13 Yasir
6 34 Yusuf
7 17 Akif
8 8 Mustafa
9 7 Elif
Table contains 3
Table does not contain 12
Table contains Alihan
Table does not contain Saruman
```



```
****ADDING 2 MORE ENTRIES TO EXCEED CAPACITY****
    RESIZING
0
1 1 Ahmet
2 2 Onur
3 3 Enes
4 4 Alihan
5
6
7 7 Elif
8 8 Mustafa
9
10
11
12
13 13 Yasir
14 34 Yusuf
15 427 Musab
16
17 17 Akif
18 128 Levent
19
```



```
****CREATING THE SECOND HASHMAP****
0 10 nizamettin
1
2 12 celalettin
3 23 selahattin
4
5 35 nurettin
6 16 ziyaattin
7
8
9
```



```
****CLEAR THE FIRST HASHMAP AND COPY THE SECOND TO IT USING putAll() METHOD****
0 10 nizamettin
1
2 12 celalettin
3 23 selahattin
4
5 35 nurettin
6 16 ziyaattin
7
8
9
```

2 Recursive Hashing Set

Failed to implement.

2.1 Pseudocode and Explanation

Failed to implement due to bad time management.

2.2 Test Cases

Failed to implement due to bad time management.

3 Sorting Algortihms

3.1 MergeSort with DoubleLinkedList

This part about Question3 in HW5

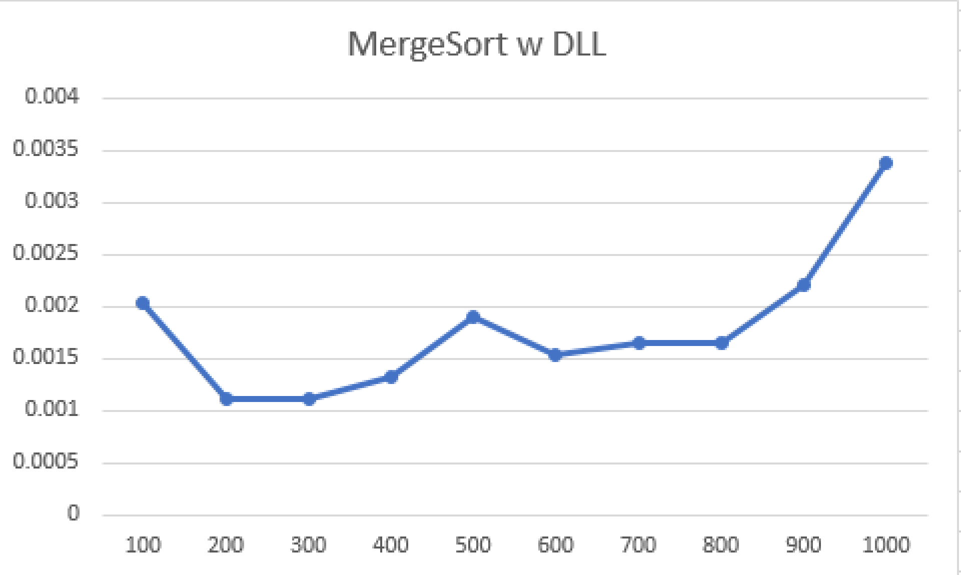
3.1.1 Pseudocode and Explanation

I implemented the basic Double Linked List node whilst extending Comparable class.

I held the beginning(head) and the end(tail) of List in fields. I used the tail for addition. I implemented split() and merge() methods and called them in a specific order in sort(). Finally I implemented the display () method to print the list

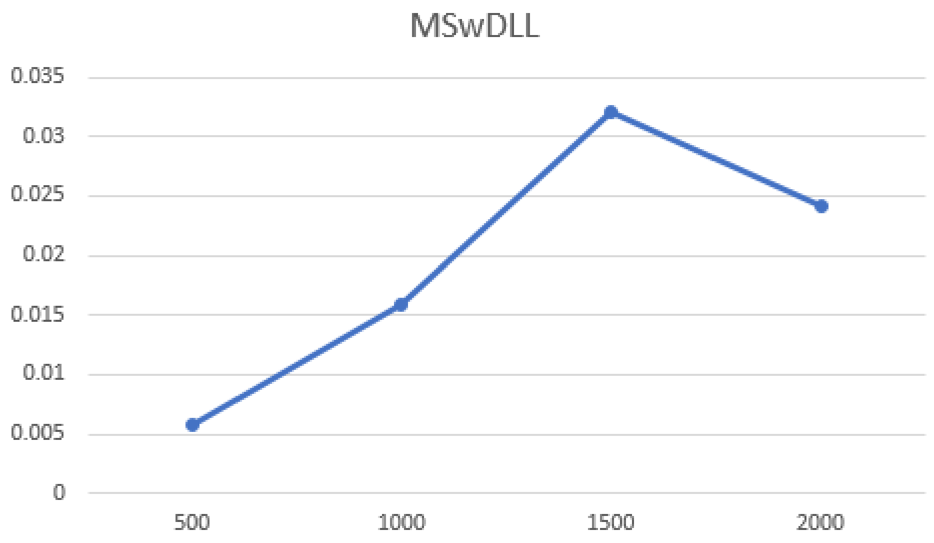
```
add(item)        Creates a node with item and links it to tail
{
    a = Node(item)
    tail.next = a
    temp = tail
    tail = tail.next
    tail.before = temp
}
mergeSort(node) //splits till each list becomes a single node than merges them again
{
    if(Node == null)
        return Node
    Node a = split(node)
    Node = mergeSort(a)
    a = mergeSort(a)
    return head = merge(node,a)
}

split () //creates 2 nodes. 1 of them increased 1by1 while the other increase 1by2
          //returns the slowly increasing one when the fast one reaches the end
{
    Node finish
    Node half
    While -> till finish reaches the end
           finish = finish.next.next
           half = half.next
    Node result = half.next
    Result.next = null
    Return result
}

merge(first,second)  //returns the other if one of them is null. Otherwise links the two list to each
                     //other according to comparison
{
   if(first == null)
      return second
   if(second == null)
      return first
   if(first < second)
         first = first + second //recursive
         return first
   else
         second = second+first //recursive
         return second
}
display() traverses the linked list starting from the head. Prints the data in each step
{
    Node temp = head
    while(temp.next != null)
         print temp
         temp = temp.next
}
```

3.1.2 Average Run Time Analysis



Gelen arraylerin tamamen random olması sebebiyle sonuçlar düzensiz. Mesela 600 elemanlı bir Linked Listin 500 elemanlıdan daha kısa sürede sortlanmasının muhtemel sebebi 500 elemanlıda daha fazla karşılaştırma ve yer değiştirme yapılmasıydı.

3.1.3 Wort-case Performance Analysis

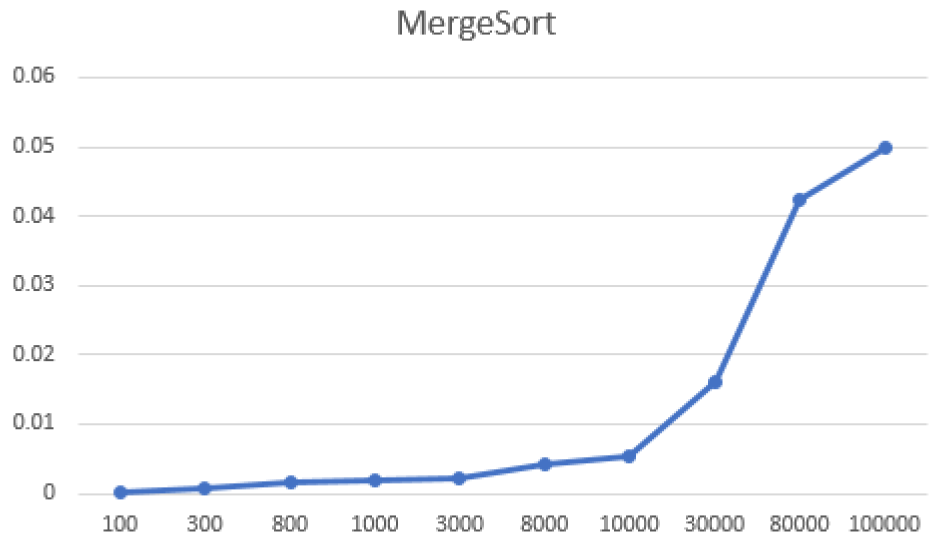


Worst case için buna benzer bir linked liste ihtiyacımız var {0,2,4,6......,1,3,5,7.......} Çünkü bu tarz bir linked list mergeSort ile sortlanırken max sayıda comparison yapılır

3.2 MergeSort

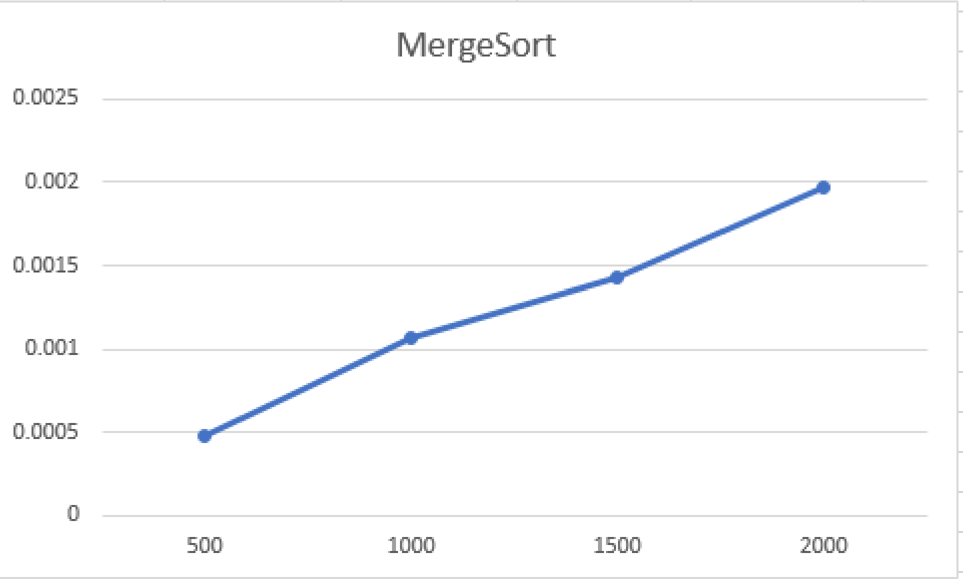
This part about code in course book.

3.2.1 Average Run Time Analysis



Veri sayısı arttıkça işlem süresi quadratik olarak artıyor

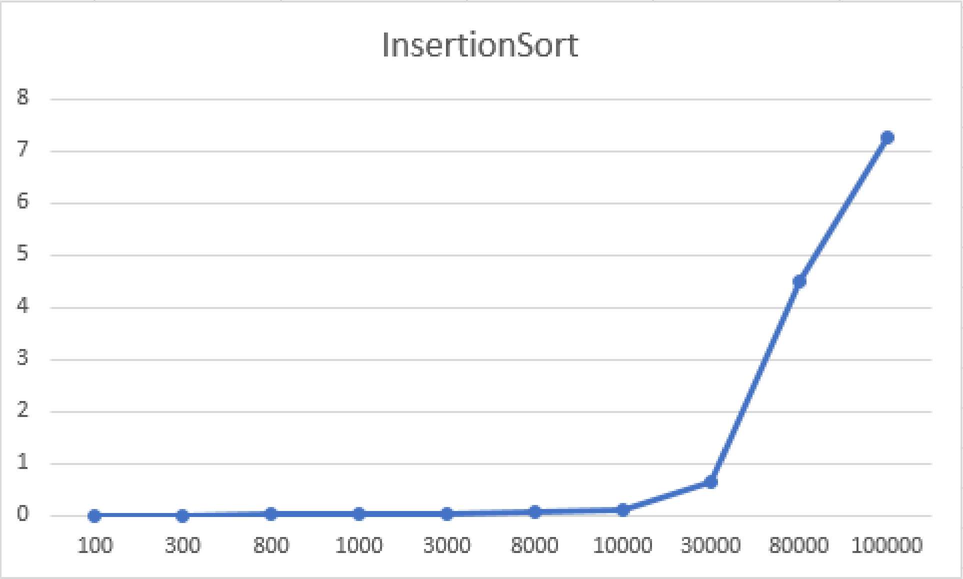
3.2.2 Wort-case Performance Analysis



Merge Sort için Worst case daha önce belirtilmişti

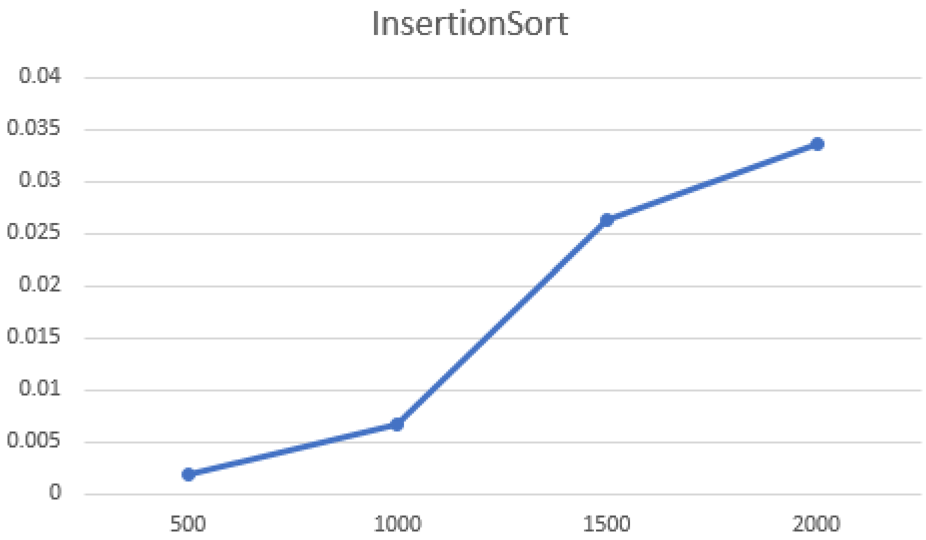
3.3 Insertion Sort

3.3.1 Average Run Time Analysis



En uzun zaman alan en basit algoritmalardan biri. Veri sayısı artışından en fazla etkilenen sort algoritması

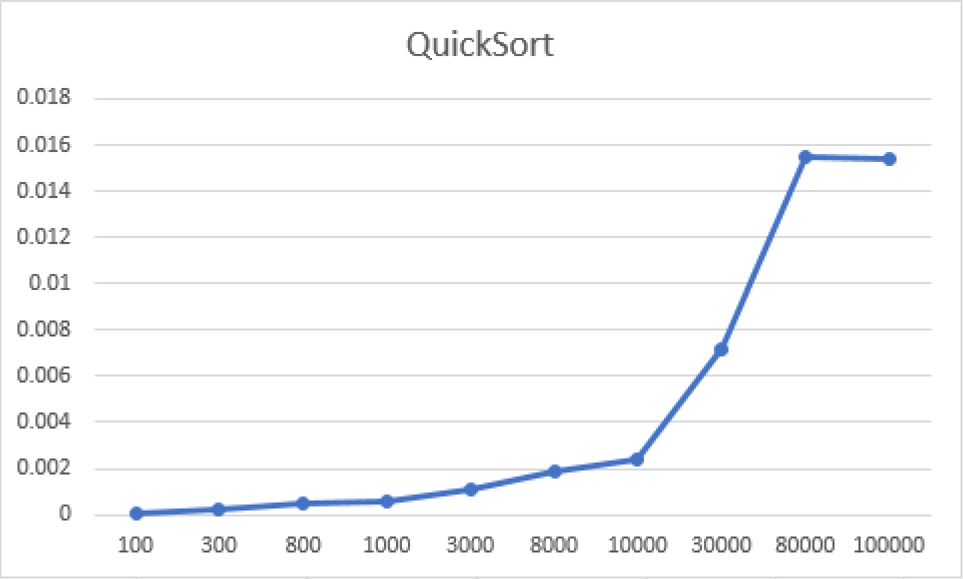
3.3.2 Wort-case Performance Analysis



Ters sıralı bir liste gönderildiğinde bütün elemanların eklenirken arrayi dolanması gerekeceği için en uzun süreyi bu alır

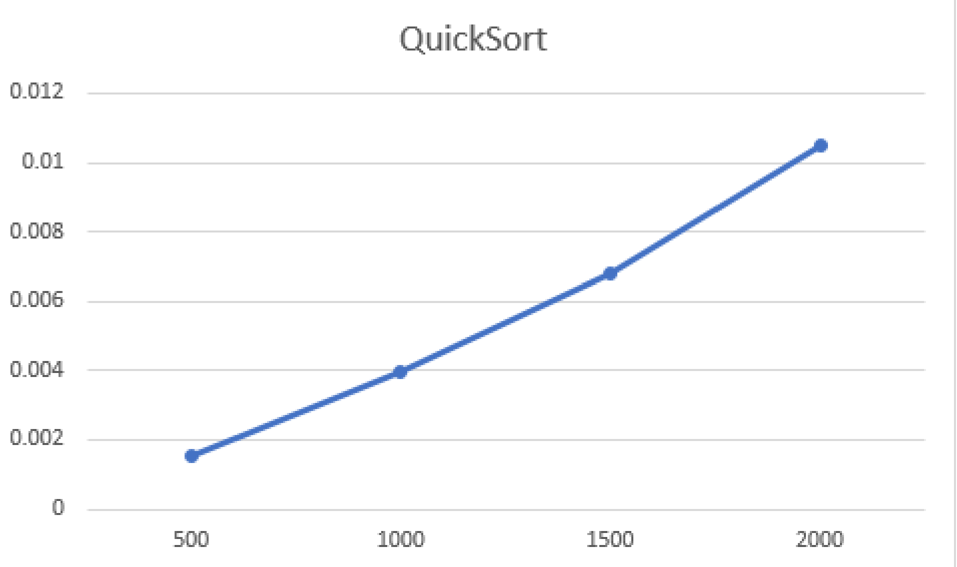
3.4 Quick Sort

3.4.1 Average Run Time Analysis



En hızlı çalışan sorting algoritması. Veri sayısı artışından diğer sorting algoritmaları kadar etkilenmiyor ve lineer artış gösteriyor. Özellikle de liste random elemanlardan oluşuyorken çok efficient bir algoritma

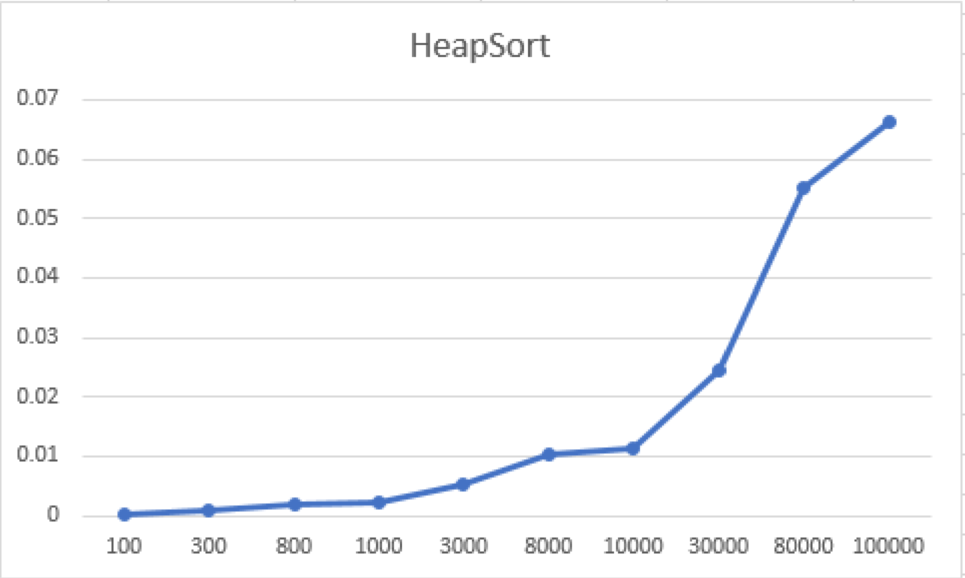
3.4.2 Wort-case Performance Analysis



Gelen arrayin sıralı olduğu durumlarda quickSortun yavaşladığını açıkça görebiliyoruz. Yine de artış hala lineer

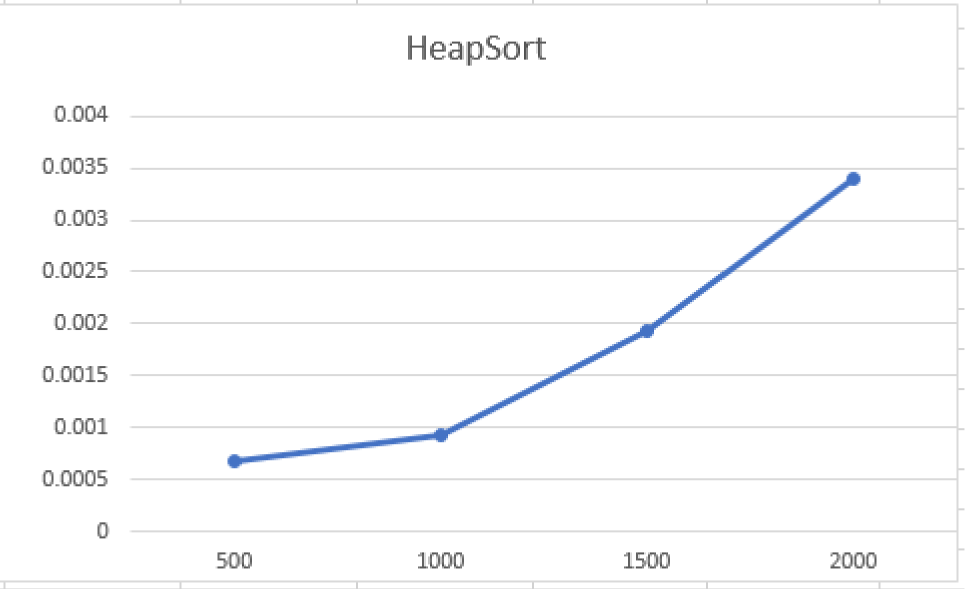
3.5 Heap Sort

3.5.1 Average Run Time Analysis



Ortalama sürede çalışan bir algoritma. Artış lineer gerçekleşiyor. Worst case bulmakta zorlanılan bir sorting algoritması

3.5.2 Wort-case Performance Analysis



Tersten sıralı bir array gönderdiğimizde algoritmanın daha yavaş çalıştığını görebiliyoruz. Ancak yine de tam bir worst Case sayılmayabilir

4 Comparison the Analysis Results

Average Case Graphs (y = second, x = element count)