Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

MESUT BUNALDI
111044077

Course Assistant:
FATMA NUR ESİRCİ

1 Double Hashing Map

This part about Question1 in HW5

1.1 Pseudocode and Explanation

To String Method for MyMap object :

```
   ***** ToString method *******************************************
new StringBuilder
string append "{ "
for i = 0 to  table.length
  if (table[i] != null and table[i] != DELETED)
    String str assign table[i].getKey() + "=" + table[i]
    stringbuilder append(str)

  myBuild.deleteCharAtmyBuild.lastIndexOf ","
string append "}"
return stringbuilder
```

Hash method for table index:

```
**********has2 method ***************************************
index=PRIME- (key.hashCode()-PRIME)
while (index<0){
  index+=table.length

return index
```

Find method for map element:

```
find(Object key){
  // Calculate the starting index.
  index = key.hashCode() % table.length
  i=1;
  if (index < 0)
   index += table.length
  while (table[index] != null than
     and !key.equals(table[index].getKey)
   index=(key.hashCode()+i*has2(key))%table.length;
   while (index<0){
     index+=table.length;

   if index >= table.length than
     index = 0

   return index
```

Put Method for add Element on Map Object:

```
put(K key, V value)
index = find(key)
if table[index] == null than
  table[index] = new EntryClass<>(key, value)
  numKeys++
  double loadFactor =(numKeys + numDeletes) / table.length
  if (loadFactor > LOAD_THRESHOLD) than
    call rehash() method
  return null

oldVal = table[index].getValue()
table[index].setValue(value)
return oldVal
```

Rehash Method for Collision :

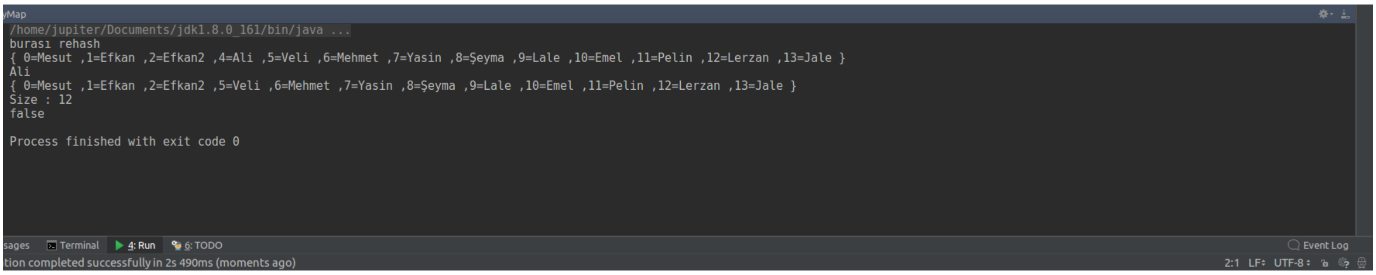
```
Entry<K, V>[] oldTable = table
table = new Entry[2 * oldTable.length + 1]
numKeys = 0
numDeletes = 0
for (int i = 0; to table.length
  if (oldTable[i] != null) && (oldTable[i] != DELETED) than
    put -oldTable[i].getKey(), oldTable[i].getValue()-
```

Remove element from Map :

```
remove(Object key)
int index = find(key)
 if table[index] equal to null than
   return null
 oldValue = table[index].getValue
 table[index] = DELETED
 numKeys--
 return oldValue
```

1.2 Test Cases

Situation1:

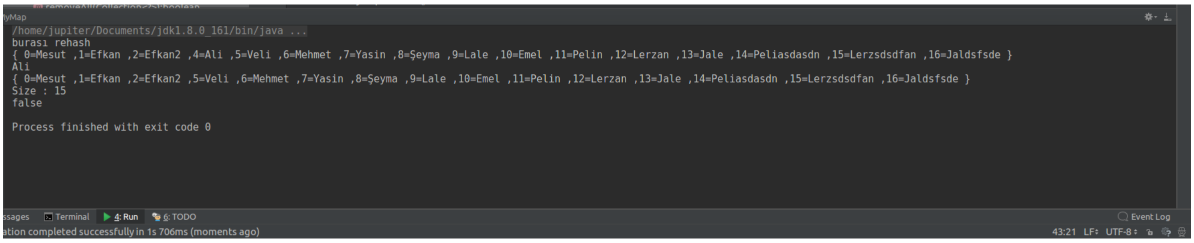


```
/home/jupiter/Documents/jdk1.8.0_161/bin/java ...
burası rehash
{ 0=Mesut ,1=Efkan ,2=Efkan2 ,4=Ali ,5=Veli ,6=Mehmet ,7=Yasin ,8=Şeyma ,9=Lale ,10=Emel ,11=Pelin ,12=Lerzan ,13=Jale }
Ali
{ 0=Mesut ,1=Efkan ,2=Efkan2 ,5=Veli ,6=Mehmet ,7=Yasin ,8=Şeyma ,9=Lale ,10=Emel ,11=Pelin ,12=Lerzan ,13=Jale }
Size : 12
false

Process finished with exit code 0
```

tion completed successfully in 2s 490ms (moments ago)

Situation2:



```
/home/jupiter/Documents/jdk1.8.0_161/bin/java ...
burası rehash
{ 0=Mesut ,1=Efkan ,2=Efkan2 ,4=Ali ,5=Veli ,6=Mehmet ,7=Yasin ,8=Şeyma ,9=Lale ,10=Emel ,11=Pelin ,12=Lerzan ,13=Jale ,14=Peliasdasdn ,15=Lerzsdsdfan ,16=Jaldsfsde }
Ali
{ 0=Mesut ,1=Efkan ,2=Efkan2 ,5=Veli ,6=Mehmet ,7=Yasin ,8=Şeyma ,9=Lale ,10=Emel ,11=Pelin ,12=Lerzan ,13=Jale ,14=Peliasdasdn ,15=Lerzsdsdfan ,16=Jaldsfsde }
Size : 15
false

Process finished with exit code 0
```

ation completed successfully in 1s 706ms (moments ago)

2 Recursive Hashing Set

This part about Question2 in HW5

2.1 Pseudocode and Explanation

Contains method for HashTable :

```
contains(element)
  index = element.hashCode() % mainArr.length
  if mainArr[index] == null or mainArr[index].getVal()!=element than
    return false
  else if(mainArr[index].getVal()==element) than
    return true
  else
    return contains(element)
```

Add method for HashTable:

```
add(table, element)
  index = element.hashCode() % table.length
  if (table[index] equal to null) than
    table[index] = new EntryClass<>(element)
    size++
    return true
  else if (table[index] != null && table[index].getVal()!=element)than
    table[index].next = new EntryClass[capacity]
    add(table[index].next, element)
    return true
  return false
```

Remove element from HashTable:

```
remove2(table, element)
  index = element.hashCode() % table.length
  if (table[index] == null || table[index].getVal()!=element ) than
    return false
  else if(table[index].getVal()==element)
    table[index].setVal(DELETED)
    deletedSize++
    return true
  else
    return remove2(table[index].next,element)
```

2.2 Test Cases

Inputs :

```
System.out.println("Ekleme işlemi : "+tmp.add(9));
System.out.println("Ekleme işlemi : "+tmp.add(e));
System.out.println("Aynı elemanı ekleme işlemi : "+tmp.add(e));
System.out.println("IsEmpty =  " + tmp.isEmpty());
System.out.println("Ekleme işlemi : "+tmp.add(9));
System.out.println("IsEmpty =  " + tmp.isEmpty());
System.out.println("Contains = "+tmp.contains(1));
System.out.println("Contains = "+tmp.contains(9));
```

Outputs :

```
Ekleme işlemi : true
Ekleme işlemi : true
Aynı elemanı ekleme işlemi : false
IsEmpty =  false
Ekleme işlemi : false
IsEmpty =  false
Contains = false
Contains = true
```

3 Sorting Algortihms

3.1 MergeSort with DoubleLinkedList

This part about Question3 in HW5

3.1.1 Pseudocode and Explanation

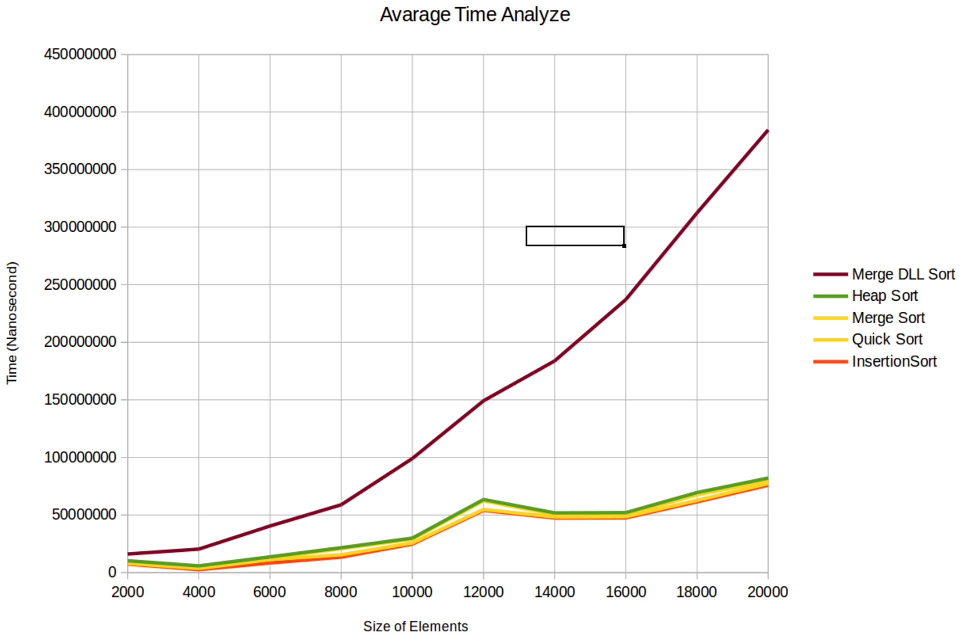
Merge for Doble Linked List:

```
merge(LinkedList outputSeq, LinkedList leftSeq, LinkedList rightSequence)
    i=0
    j=0
    k=0
    while(i<leftSequence.size() and j<rightSequence.size())
      if(leftSequence.get(i).compareTo(rightSequence.get(j))<0) than
        outputSequence.set(k++,leftSequence.get(i++))
      else
        outputSequence.set(k++,rightSequence.get(j++))
    while(i<leftSequence.size())
      outputSequence.set(k++,leftSequence.get(i++))
    while(j<rightSequence.size())
      outputSequence.set(k++,rightSequence.get(j++))
```

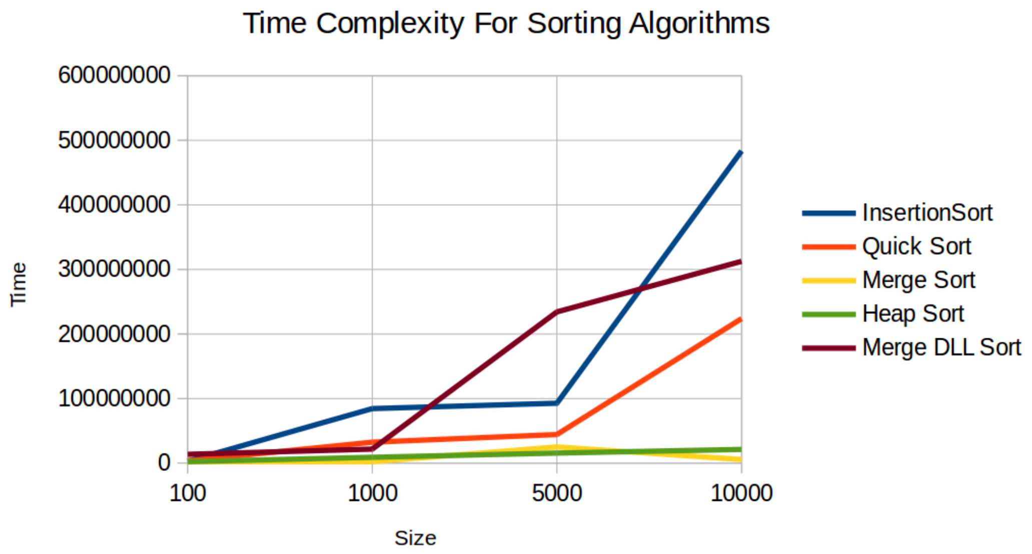
Sort for Double Linked List :

```
sort(LinkedList mylist)
 if (mylist.size() > 1) than
   leftSize = mylist.size() / 2
   rightSize = mylist.size() - leftSize
   LinkedList<T> leftList = new LinkedList
   LinkedList<T> rightList = new LinkedList
   for int i = 0 to leftSize
     leftList.add(mylist.get(i))
   for int i = leftSize to myList.size
     rightList.add(mylist.get(i))
   sort(leftList)
   sort(rightList)
   merge(mylist, leftList, rightList)
```

3.1.2 Average Run Time Analysis



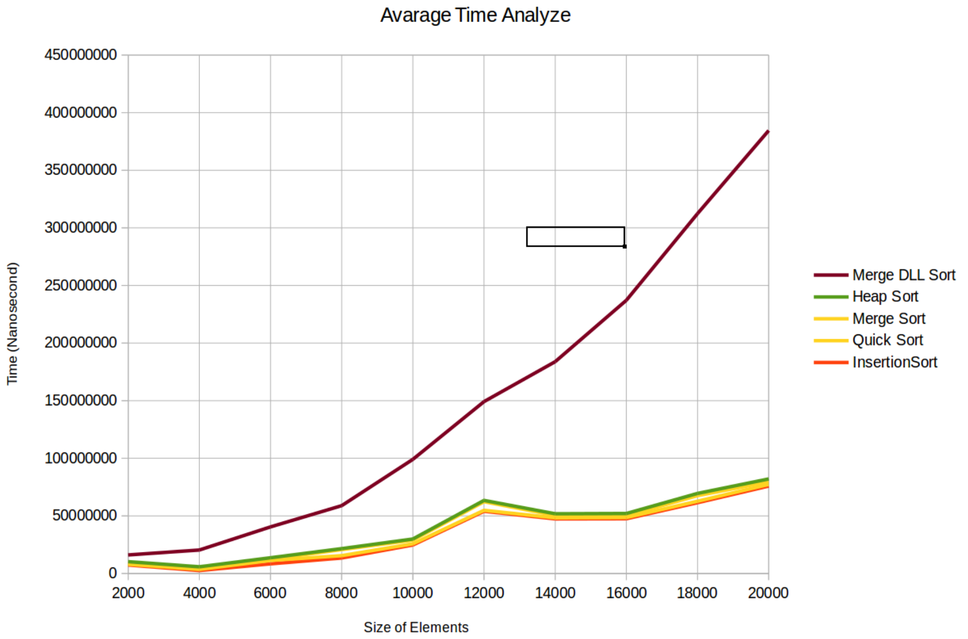
3.1.3 Wort-case Performance Analysis



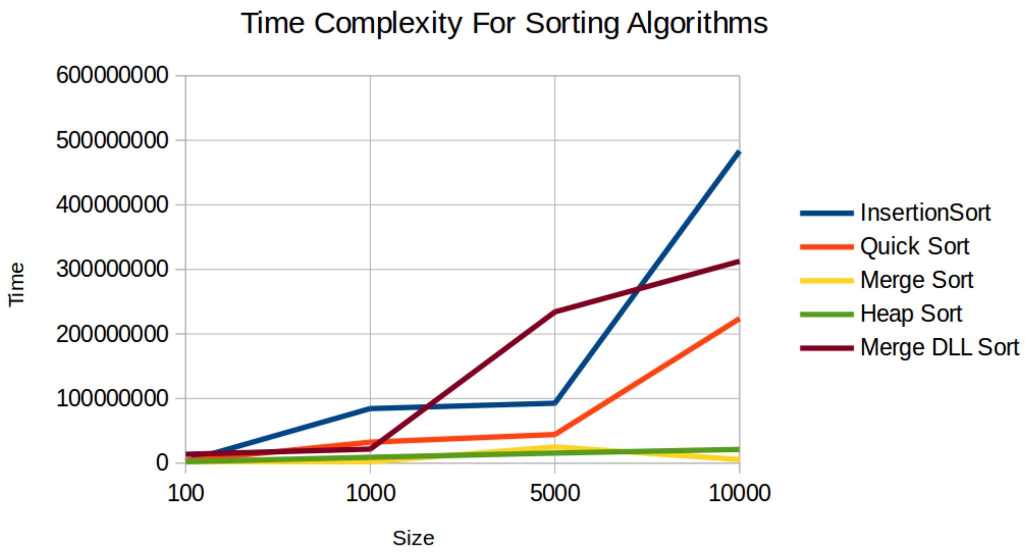
3.2 MergeSort

This part about code in course book.

3.2.1 Average Run Time Analysis

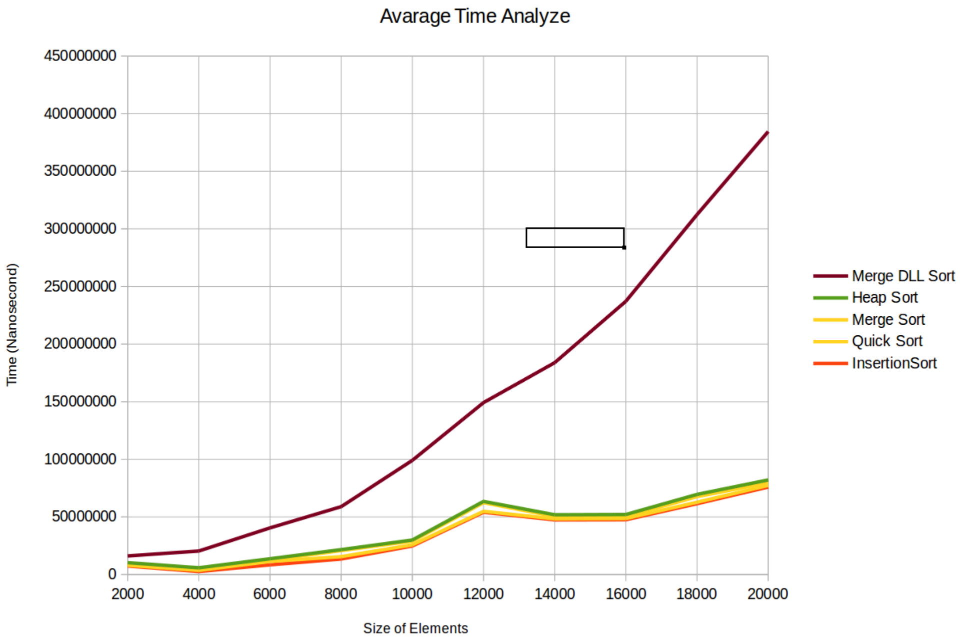


3.2.2 Wort-case Performance Analysis

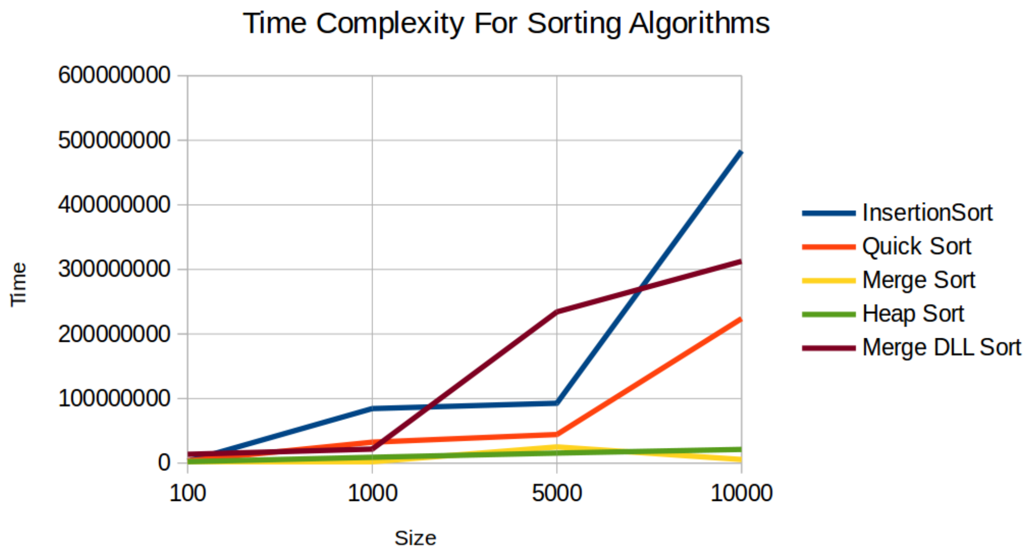


3.3 Insertion Sort

3.3.1 Average Run Time Analysis

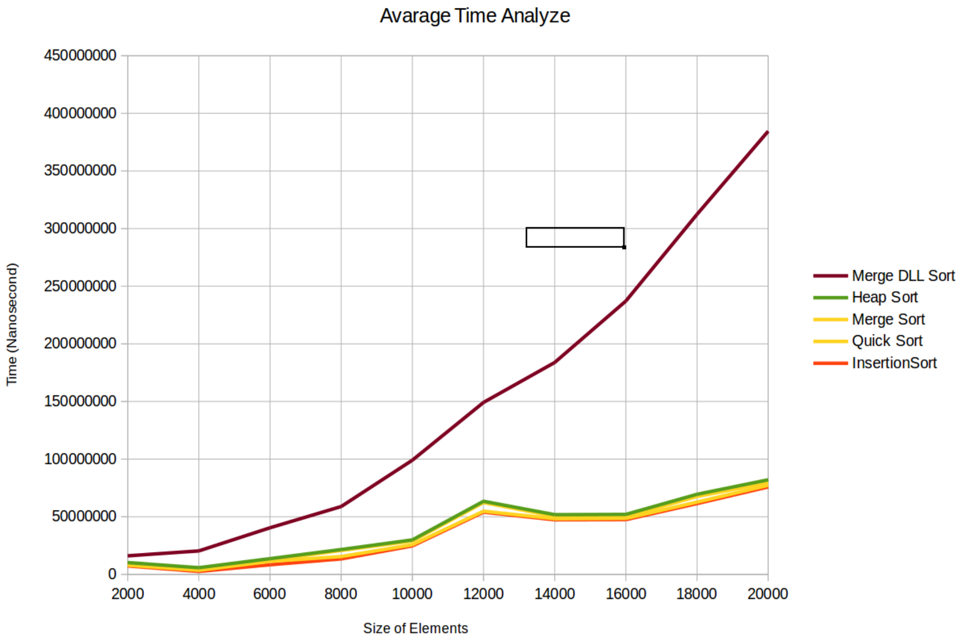


3.3.2 Wort-case Performance Analysis

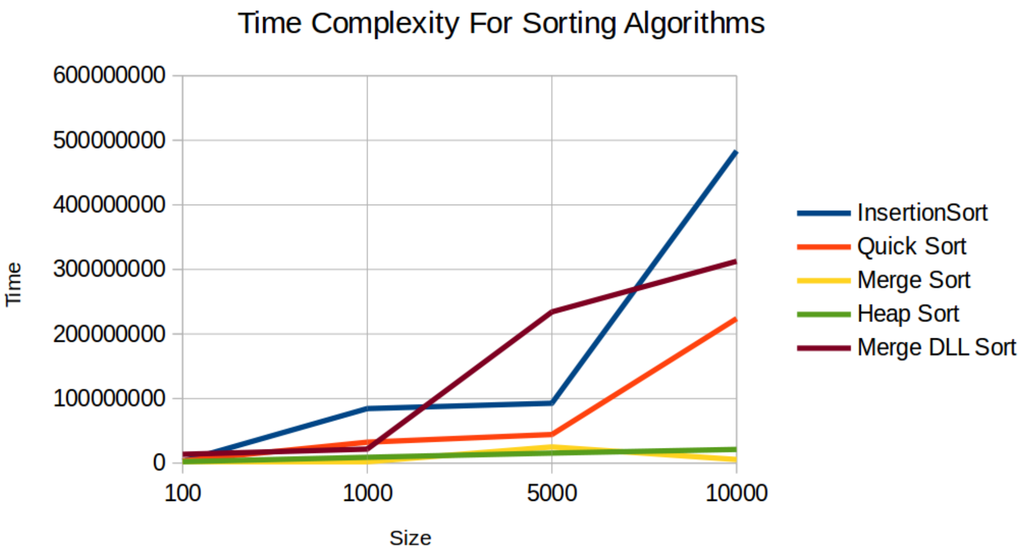


3.4 Quick Sort

3.4.1 Average Run Time Analysis

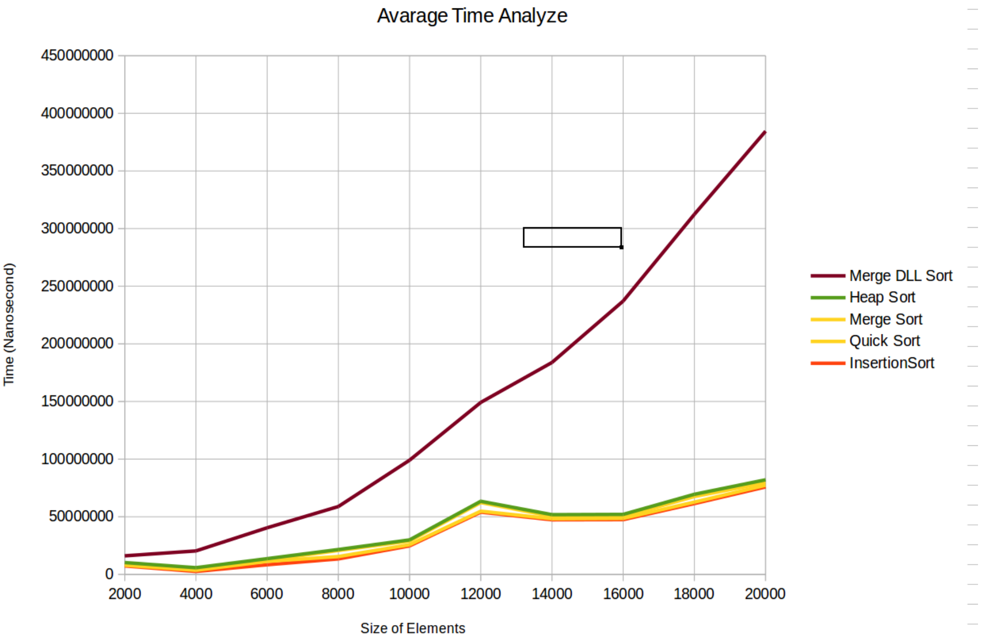


3.4.2 Wort-case Performance Analysis

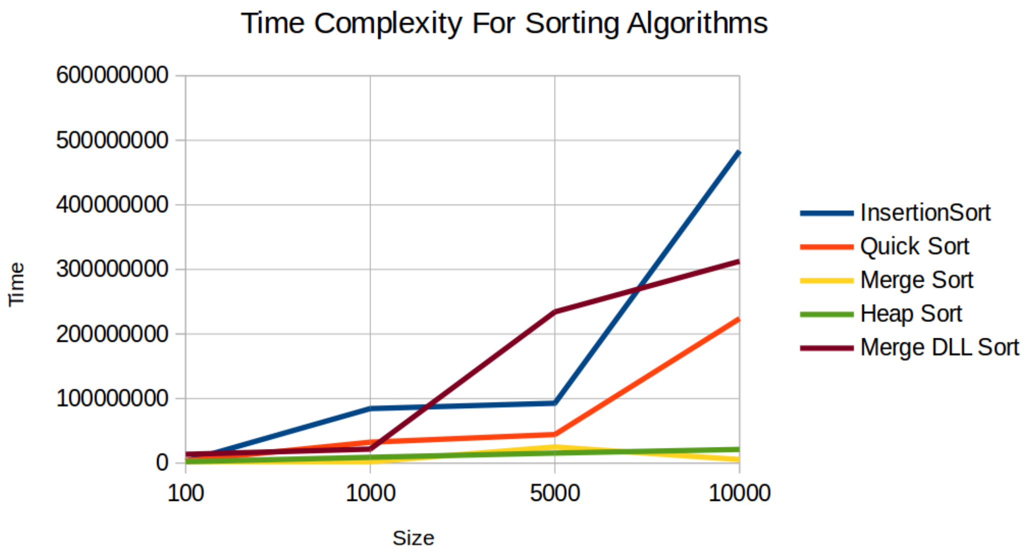


3.5 Heap Sort

3.5.1 Average Run Time Analysis



3.5.2 Wort-case Performance Analysis



4 Comparison the Analysis Results

Sorting algoritmaları küçük değerler için çok büyük zaman maliyetleri sergilememekle beraber veri miktarı arttıkça zaman maliyetleri farklılaşmaktadır.

Zaman maliyeti açısından büyükten küçüğe sıralama yapmak gerekirse:

1- Insertion Sort Algorithm : Zaman geçtikçe parabolik olarak artış göstermektedir. Çalışma hızı için O(n^2) diyebiliriz.

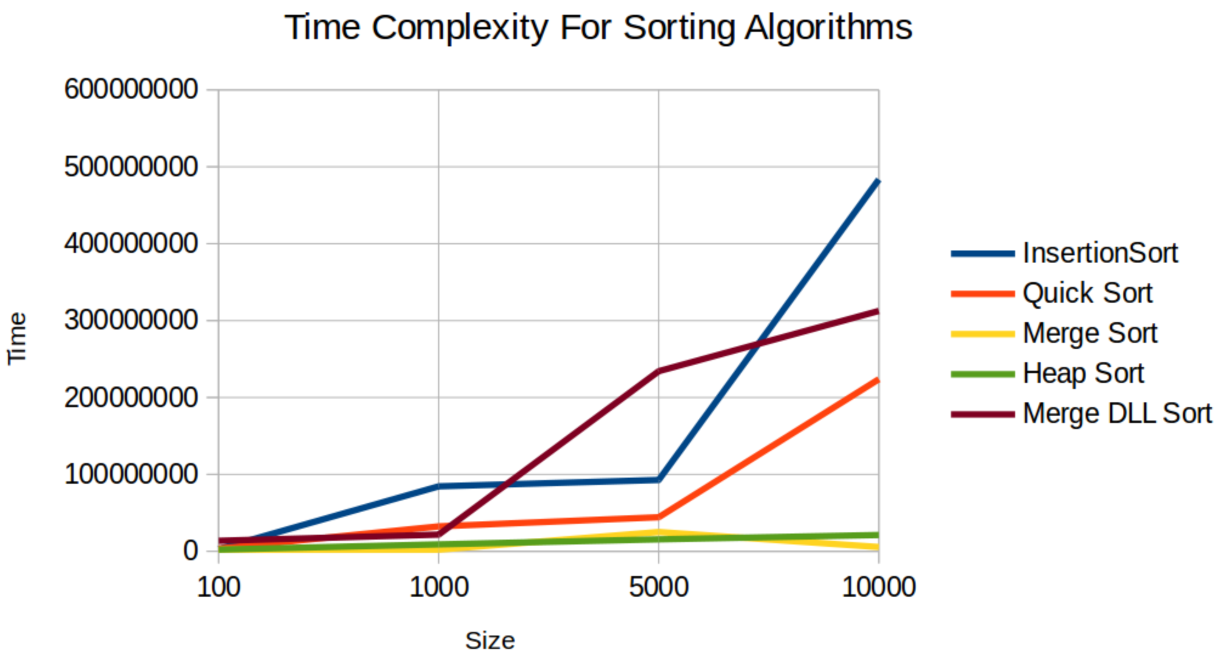
2-Merge DoubleLinkedList Sort : Zaman geçtikçe lineer artış göstermektedir. Bu algoritma için çalışma hızı O(n) diyebiliriz.

3-Quick Sort: Merge Double LinkedList Sort algoritması ile hemen hemen aynı çalışma zamanı maaliyetine sahiptirler. O(n).

4-Heap Sort : Zaman maliyeti en düşük algoritmalardan biridir. Çalışma zamanı O(log n) diyebiliriz.

5: Merge Sort : Zaman maliyeti en düşük olan algoritmalardan bir diğeridir. Bu algoritmanın da çalışma zamanı için O(log n) diyebiliriz.

Size	InsertionSort	Quick Sort	Merge Sort	Heap Sort	Merge DLL Sort
100	2924061	3470646	1996716	2102823	13579226
1000	84452720	32401165	2026862	8922721	21696998
5000	92722249	44193488	25072292	15459821	234130735
10000	483480908	223788766	5530554	21121876	312508214


Time Complexity For Sorting Algorithms
Time
Size
InsertionSort
Quick Sort
Merge Sort
Heap Sort
Merge DLL Sort