Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

Vefik Fırat Akman
151044031

Course Assistant: : Fatma Nur Esirci

1 Double Hashing Map

Bu part da Java da İnterface olarak bulunan Map'i impelement ettim. İmplement ettiğim için map ile birlikte 12 metot geldi. **EntrySet()** metodu dışında 11'rini %100 doğru bir şekilde implement ettim.
Ayrıca kendim **rehash()** metodunu ekledim. Bu metot table'ın size ı dolunca size'ı ikiye katlayıp bu size ile birlikte tekrar elemanlarını hash'ler.
2 farklı table ile de test ettim. İlk table da key büyüklükleri random. İkinci table da ise key'ler küçükten büyüğe doğru sıralıdır.

Sonuç: 1.Soru tamamen ve doğru bir şekilde yapılmıştır.

1.1 Pseudocode and Explanation

```
DoubleHash<K,V> implements Map<K,V>
```

1. If(size == capacity) true ise rehash çağır devam et false ise direk devam et.
2. If Containskey(key) true döndürürse devam et. Yoksa 3 e git
3. keyin yerini bul value'sunu değiş. (Else girmez 11 e gider.)
4. Else{
5. İndex = H(key);
6. While(!completed)
7. If index boşsa devam et doluysa 4 de git.
8. index hücresine key,value'yu yerleştir. Completedı true yap (Else girmez 5 e gider.)
9. Else double hash metoduyla tekrar hash al. 2 ye git.
10. End of while
11. }
12. ++size;
13. Return;

1.2 Test Cases

Table1:

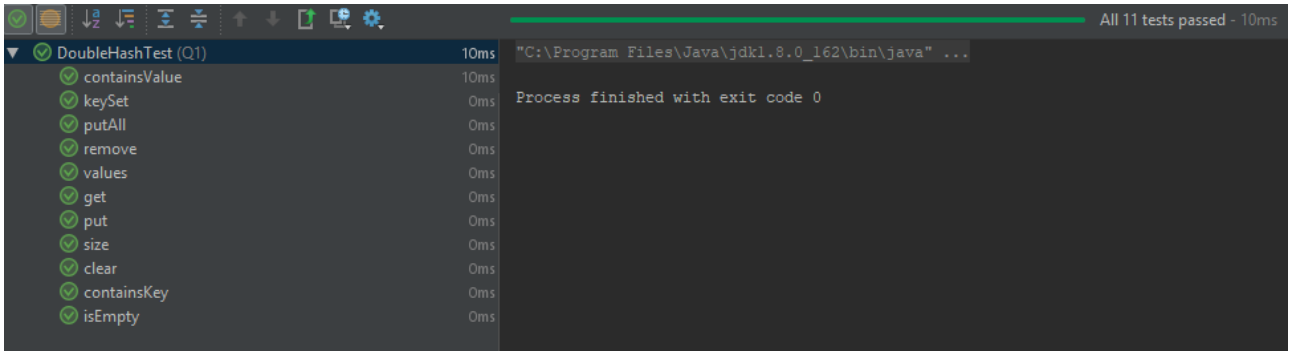
```
myDoubleHash.put(0,"Key 0 - Input Order = 1");
myDoubleHash.put(5,"Key 5 - Input Order = 2");
myDoubleHash.put(14,"Key 14 - Input Order = 3");
myDoubleHash.put(16,"Key 16 - Input Order = 4");
myDoubleHash.put(2,"Key 2 - Input Order = 5");
myDoubleHash.put(17,"Key 17 - Input Order = 6");
myDoubleHash.put(11,"Key 11 - Input Order = 7");
myDoubleHash.put(21,"Key 21 - Input Order = 8");
myDoubleHash.put(3,"Key 3 - Input Order = 9");
```

14 numaralı key'i 0'ra koymak isteyecek ama orası dolu olduğu için double hash yapıp yeni yer bulacak ve uygunsa oraya koyacak. Bu table da bu tarz senaryoları düşünüp yazdım.

Table2:

```
myDoubleHash2.put(0,"Key 0 - Input Order = 1");
myDoubleHash2.put(2,"Key 3 - Input Order = 2");
myDoubleHash2.put(5,"Key 5 - Input Order = 3");
myDoubleHash2.put(14,"Key 14 - Input Order = 4");
myDoubleHash2.put(17,"Key 17 - Input Order = 5");
myDoubleHash2.put(21,"Key 21 - Input Order = 6");
myDoubleHash2.put(32,"Key 32 - Input Order = 7");
myDoubleHash2.put(45,"Key 45 - Input Order = 8");
myDoubleHash2.put(52,"Key 52 - Input Order = 9");
myDoubleHash2.put(64,"Key 64 - Input Order = 10");
myDoubleHash2.put(71,"Key 71 - Input Order = 11");
```

Unit Test:



2 Recursive Hashing Set

Bu partı süre yetmediği için yapamadım.

3 Sorting Algortihms

3.1 MergeSort with DoubleLinkedList

Bu part da kendi DoubleLinkedList'imi ve bu DoubleLinkedList de çalışan kendi MergeSortumu yazdım.
DoubleLinkedList generic olup MergeSort da bu generic listede çalışabilmektedir. Bunun için DoubleLinkedList de Comparebla interfacini implement ettim.

Sonuç: 3.Soru tamamen ve doğru bir şekilde yapılmıştır.

3.1.1 Pseudocode and Explanation

```
DoubleLinkedList<T> implements Comparable<T>
```

addAll(DoubleLinkedList other)

- for Other'ın size'ı kadar döner
- add fonksiyonu otherın her elemanıyla tek tek çağırılır
- Fonksiyon biter.

Add(T data)

- Data ile yeni node oluşturur.
- Head null mı bakar
- Head null ise head e yeni node u atar. Ve fonksiyon biter.
- Head null değilse{
- Head.next null olana kadar listede ilerler.
- Next null olunca next e yeni node u atar.

Get(index)

- Temp node oluşturulur head atanır.
- For index e kadar 0'dan başlayarak döner
- Temp node = temp node'un nexti diye döngü bitene kadar listede ilerlenir
- Döngü biter.2
- Şu anki tempin gösterdiği node'un datası döndürülür

removeFirst()

- head in datasını T tipinde bir variable a atar.
- Headi headin enxtine eşitler
- Garbage collector eski headi siler
- T tipinde ki variable return edilir.

Merge DoubleLinkedList Classının metodları:

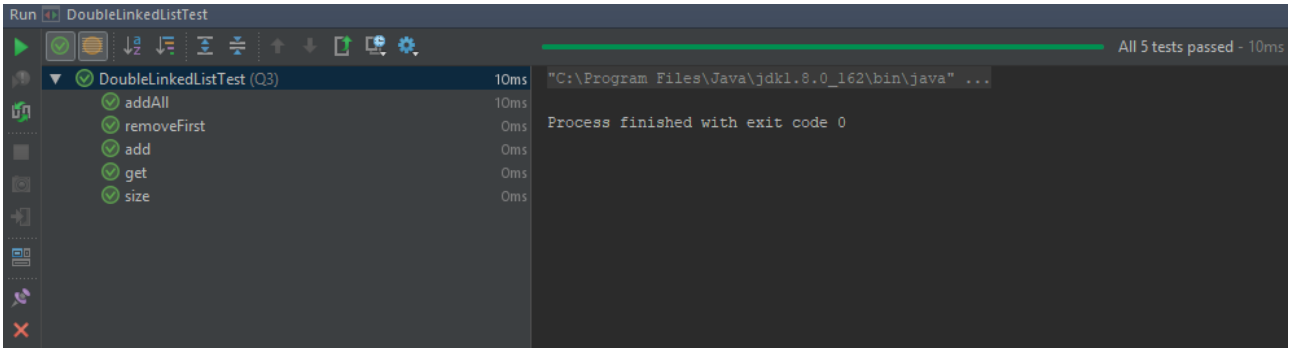
Sorter(DoubleLinkedList<T> list)// recursive bir metottur.

- Öncelikle base case olarak size'ın 1 olup olmadığına bakar.
- 1 veya 1 den küçükse return yaparak geri toplanmaya başlar.
- Değilse devam eder.
- Listenin orta kısmını listenin size'nı ikiye bölerek bulur ve middle a atar.
- addNum variable'ı 0 ra eşitlenir.
- Size kadar giden döngü boyunca left isminde ki listeye eleman eklemeye başlar.
- Eğer eklenen eleman sayısı yani addNum artık middledan küçük değilse kalan elemanları right isminde ki listeye eklemeye başlar.
- Size ulaşıldığında döngü biter.
- Ana liste ortadan left ve right olmak üzere ayrılmıştır.
- Left tekrar sortera yollanır ve return lefte atanır.
- Right tekrar sortera yollanır ve return righta atanır.
- sorter dan dönen left ve rightı alır merge metodu ile birleştirilip döndürülür.
- Dönen array result arrayine atanır.
- Return result

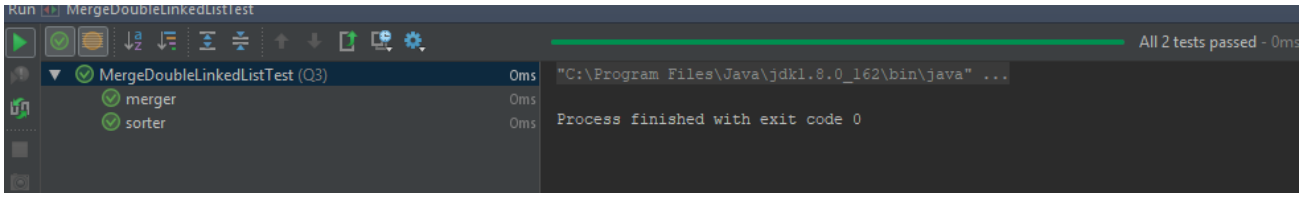
Merger(DoubleLinkedList<T> left,DoubleLinkedList<T> right)

- While left ve rightlardan birinin size'ı 0 a eşit olana kadar döner{
- İf leftin ilk elemanı rightın ilk elemanından küçükse{
- Left.removeFirst ile leftin ilk elemanını siler
- removeFirst fonksiyonundan dönen ilk elemanın datasını result a addler.
- }
- Else leftin elemanı büyükse{
- Right.removeFirst ile rightın ilk elemanını siler
- removeFirst fonksiyonundan dönen ilk elemanın datasını result a addler.
- Döngü bittiğinde hangi tarafta eleman kalmışsa onun kalan tüm elemanlarını resulta ekler.
- Return result;

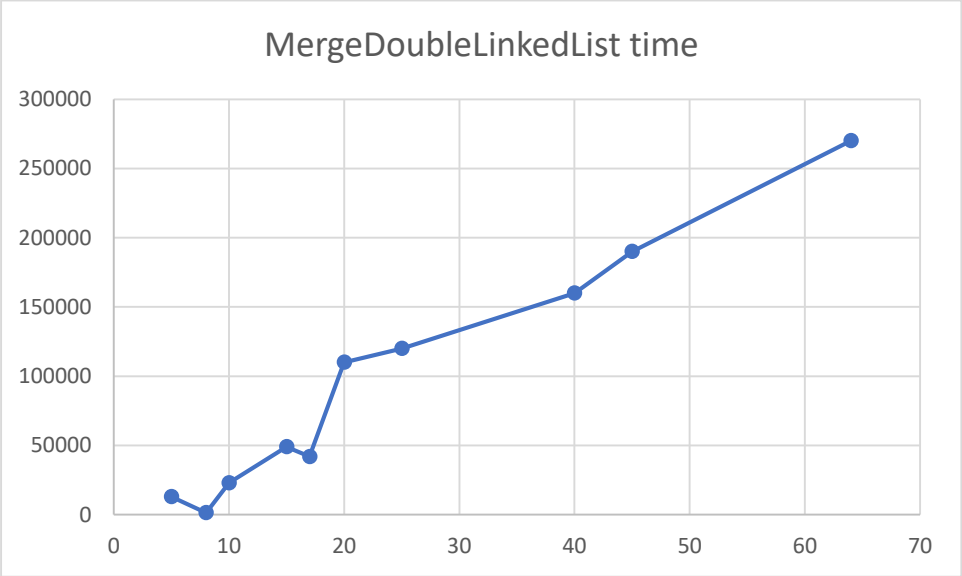
Unit Test:
DoubleLinkedList:



MergeDoubleLinkedList;

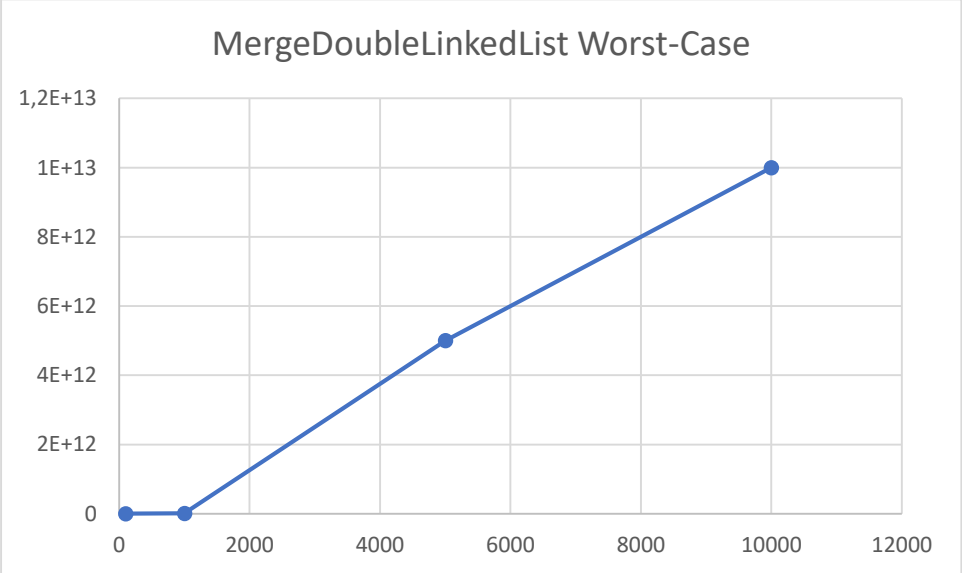


3.1.2 Average Run Time Analysis



Bu olduğu gibi düzgün çıktı. Lakin diğer tablolarda ilk sorted işlemenin yüksek çıkması tarzı bir sorun var.

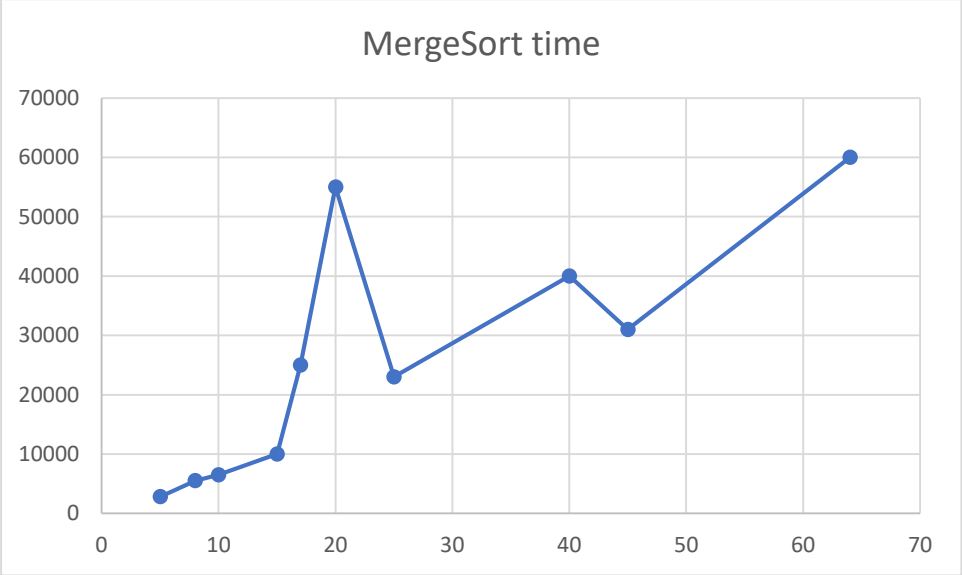
3.1.3 Wort-case Performance Analysis



3.2 MergeSort

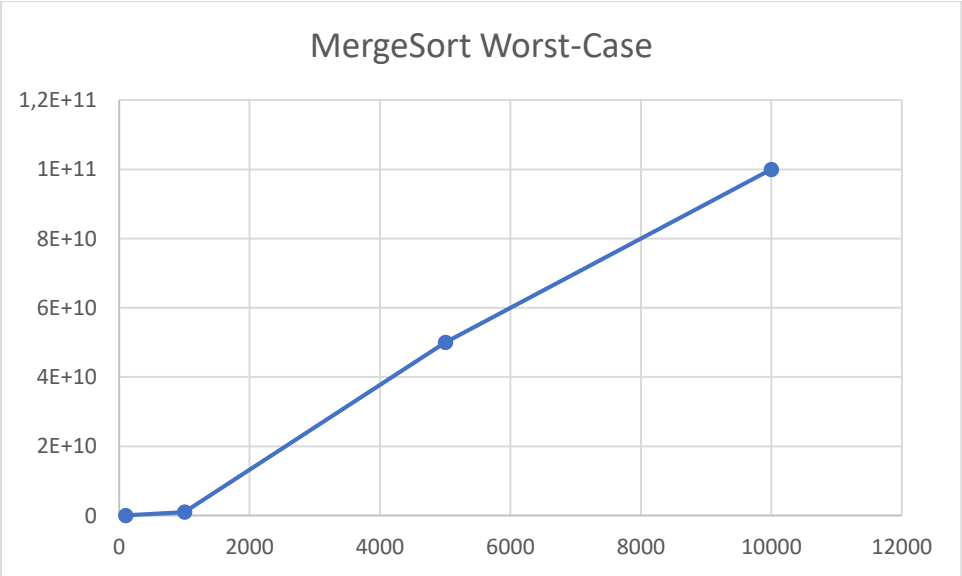
MergeSort kodunu kitaptan aldım.

3.2.1 Average Run Time Analysis



20 size'lı array ilk hesaplanan arraydir. Ve diğer arrayleri de yollasam oraya ilk hesaplanan array her zaman çok yüksek çıktı. Tablo da 55.000 yazsa da normal de 550.000 çıktı. Tablo okunaklı olsun diye 10 a böldüm.

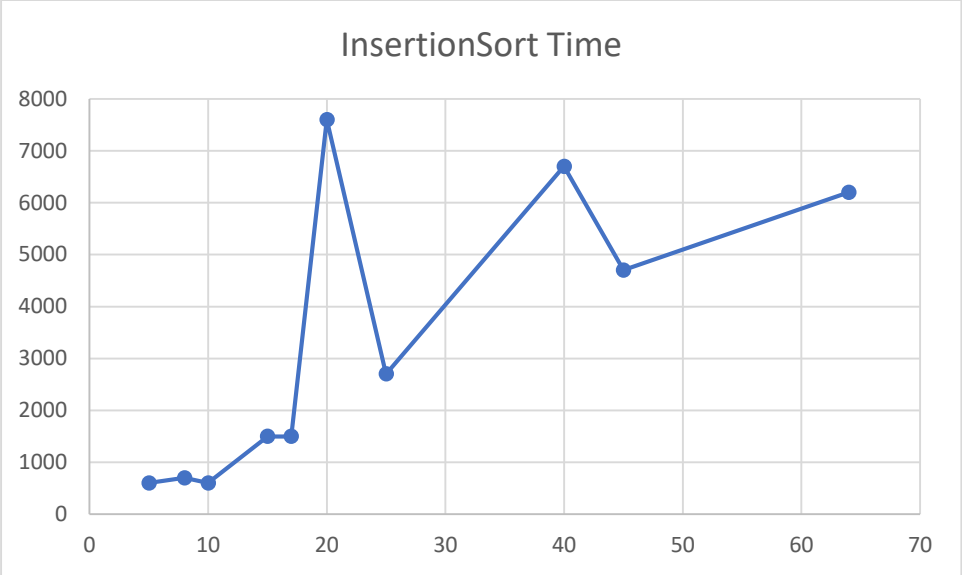
3.2.2 Wort-case Performance Analysis



3.3 Insertion Sort

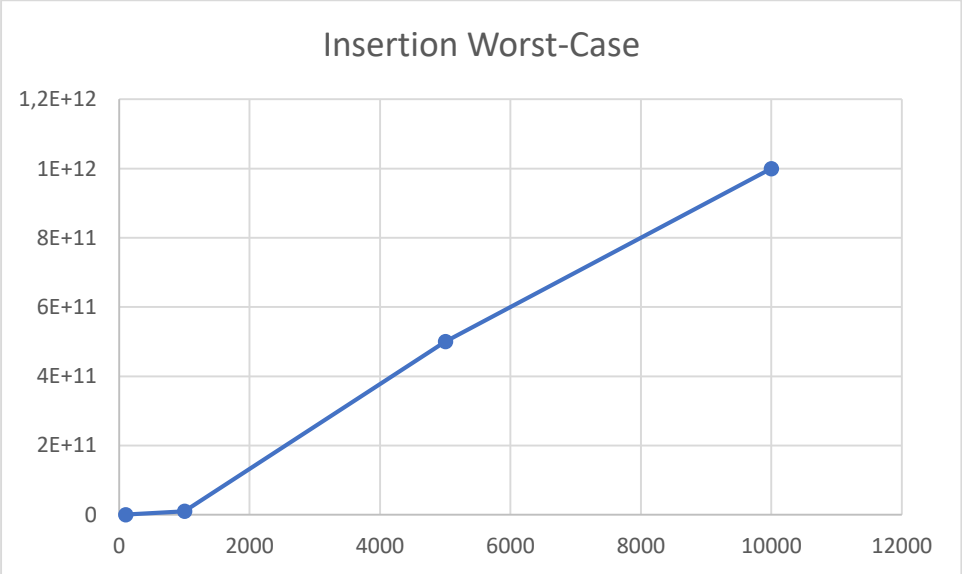
InsertionSort kodunu kitaptan aldım.

3.3.1 Average Run Time Analysis



Burda da 20 elemanlı array InsertionSort edilen ilk arraydi yine 10 a böldüm çıkan sonucu.

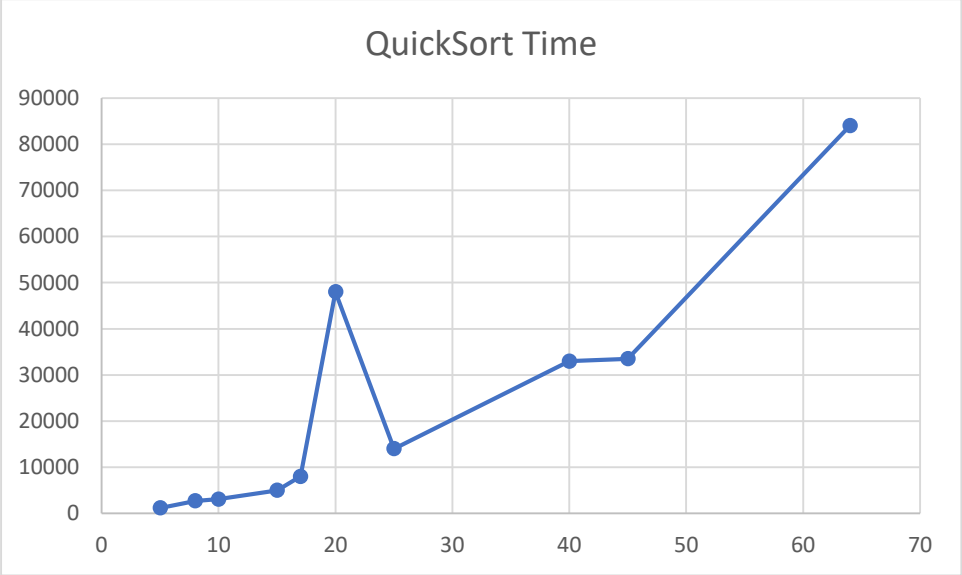
3.3.2 Wort-case Performance Analysis



3.4 Quick Sort

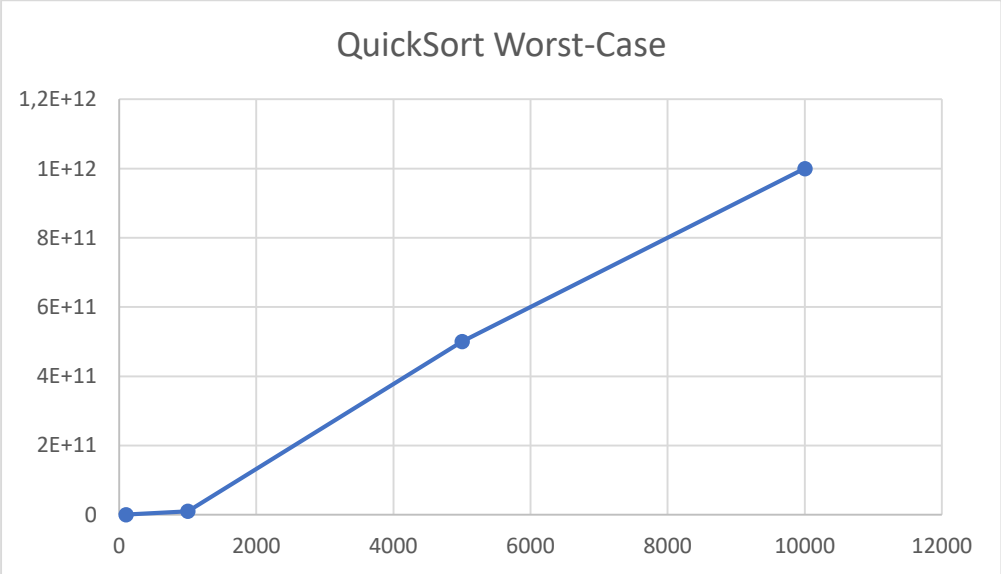
QuickSort kodunu kitaptan aldım.

3.4.1 Average Run Time Analysis



Burda da 20 elemanlı array QuickSort edilen ilk arraydi yine 10 a böldüm çıkan sonucu.

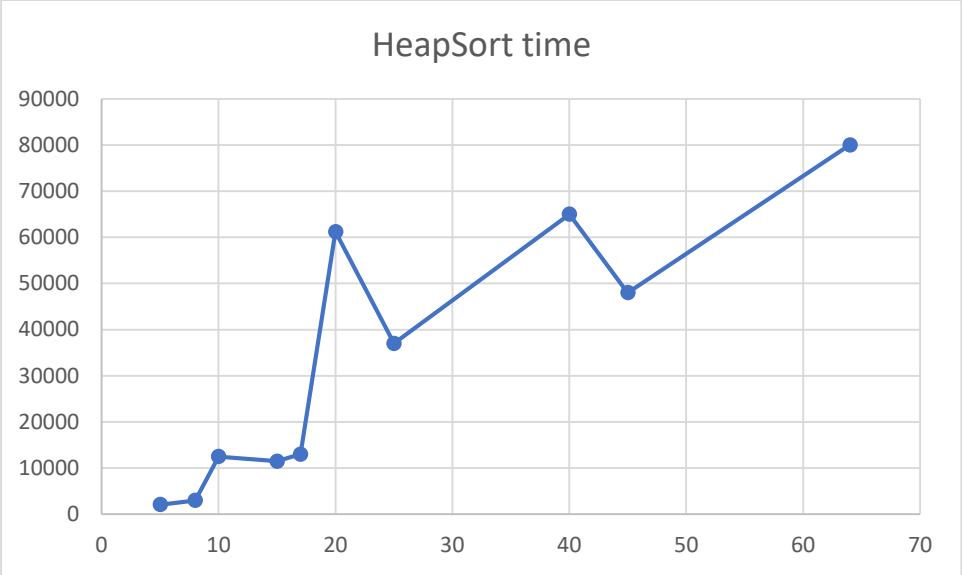
3.4.2 Wort-case Performance Analysis



3.5 Heap Sort

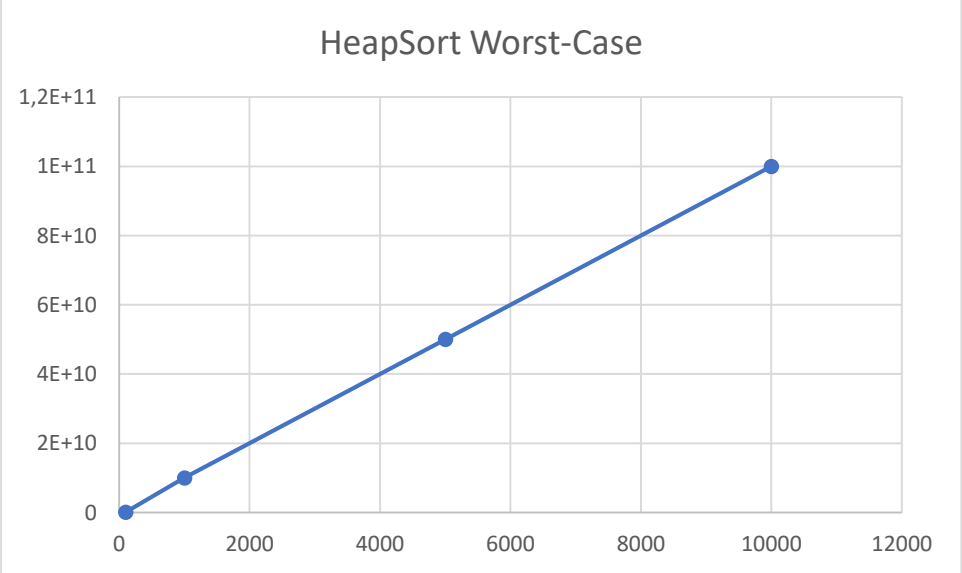
HeapSort kodunu kitaptan aldım.

3.5.1 Average Run Time Analysis



Burda da 20 elemanlı array HeapSort edilen ilk arraydi yine 10 a böldüm çıkan sonucu.

3.5.2 Wort-case Performance Analysis



Sonuç: 4.soru tamamen ve doğru şekilde yapılmıştır. 10 farklı size da arrayle ve 10 kez run edilerek yapılmıştır.

4 Comparison the Analysis Results