Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

EMİRHAN KARAGÖZOĞLU
151044052

Course Assistant: Fatmanur Esirci

1 Double Hashing Map

1.1 Pseudocode and Explanation

Double Hash Map class impliments Map interface that has get, isEmpty, put, remove and size methods. There is an inner Entry class of Double Hash Map. The entry class keeps key and value. Double Hash Map class has an Entry array(table) and private helper find and rehash method.
Pseudocode of find method:

1. elemanın hashCode() % table.length ile indeksini hesapla
2. eğer index < 0 ise
3. index += table.length
4. table[index] null ve aranan eleman olmadığı sürece ...
5. index += (Prime – (elemanın hashCode()% Prime))%table.length //-> Double Hash
6. eğer index >= table.length
7. index = 0 //->başa dönme
8. return index

Pseudocode of rehash method:

1. table referansının bi kopyasını al
2. table'a eski size * 2 lik yer al
3. tabledaki eleman ve silinmiş elemanların sayısını sıfırlar
4. eski table daki DELETED olmayan elemanları yeni table'a ekle

Pseudocode of get method:

1. aranan elemanın table'daki index'ini bul (find methodu ile)
2. table[index] null değilse
3. return table[index].value
4. table[index] null ise
5. return null

Pseudocode of put method:

1. aranan elemanın table'daki index'ini bul (find methodu ile)
2. eğer table[index] null ise
    1. table[index] 'e eklenecek elemanı koy
    2. numKey'i 1 artır.
    3. LoadFactor'ü kontrol et ve gerek varsa rehash yap
    4. return null
3. eğer table[index] null ise
    1. table[index]'in yedeğini al
    2. table[index] 'e eklenecek elemanı koy
    3. return table[index]'in yedeği

Pseudocode of remove method:

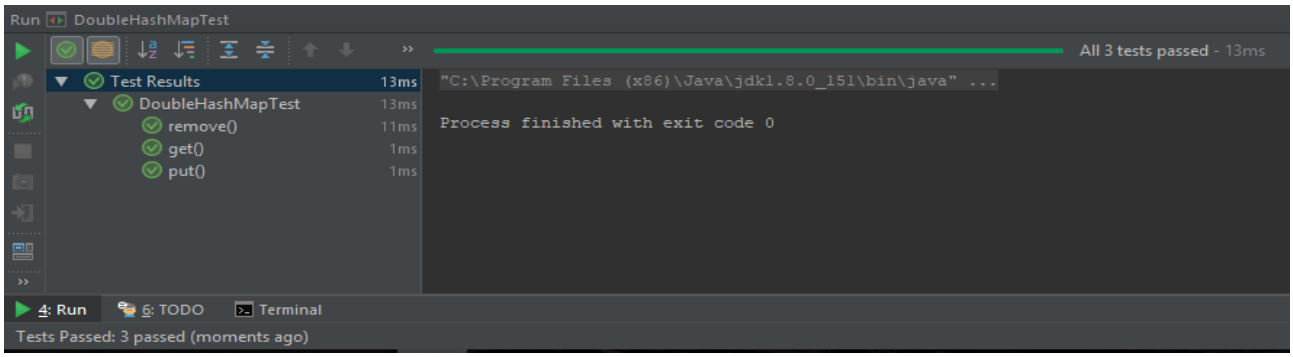
1. aranan elemanın table'daki index'ini bul (find methodu ile)
2. eğer table[index] null değilse
    1. table[index]' in yedeğini al
    2. table[index]'e DELETED ata.
    3. numKey'i 1 azalt numDeletes'i 1 artır
    4. return table[index]'in yedeği
3. eğer table[index] nullsa
    1. return null

1.2 Test Cases

```
----------------------------------------------------------------------
null - (23 first) - (34 third) - (12 second) - null - (5 fourth) - null - null - null - null - null -
Size of Hash Map: 4

Put with same key 34
null - (23 first) - (34 chanced third) - (12 second) - null - (5 fourth) - null - null - null - null - null -
Size of Hash Map: 4

Key 12 deleted
null - (23 first) - (34 chanced third) - DELETED - null - (5 fourth) - null - null - null - null - null -
Size of Hash Map: 3

Can't put DELETED index
null - (23 first) - (34 chanced third) - DELETED - (4 fifth) - (5 fourth) - null - null - null - null - null -
Size of Hash Map: 4

Rehashed. DELETED's didn't move new has table
(0 0) - (1 1) - (2 2) - (3 3) - (4 4) - (23 first) - (6 6) - (5 5) - null - null - null - (34 chanced third) - null - null - (7 7) - null - null - null - null - null - null -
Size of Hash Map: 10
----------------------------------------------------------------------
null - null - null - null - null - (third three) - null - null - null - (second two) - null - (first one) - (fourth four) -
Size of Hash Map: 4

Put with same key 'first'
null - null - null - null - null - (third three) - null - null - null - (second two) - null - (first chanced one) - (fourth four) -
Size of Hash Map: 4

Key 'first' and 'second' deleted
null - null - null - null - null - (third three) - null - null - null - DELETED - null - DELETED - (fourth four) -
Size of Hash Map: 2
----------------------------------------------------------------------
```

İlk örnekte Integer,String 11 size'lı bir Double Hash Map'e 4 eleman collusion'a sebep olacak şekilde eklendi ve double hash yapıldığı gözlendi. Ardından eklediğimiz bir key ile tekrar başka bir value eklemeye çalıştığımızda var olan elemanın value'sunun değiştirildiği gözlendi. Daha sonra Map'ten bir eleman silindi ve hash table'da o indexin DELETED olarak değiştiği gözlendi. Son olarak Map'e bir miktar eleman eklendi ve rehash yapılmak durumuna geldi. Rehash yapılmış table'a baktığımızda DELETED'lartın aktarılmadığını geriye kalan tüm elemanları yeni table'a eklendiğini gözlemledik.

İkinci örnekte String,String 13 size'lı bir Double Hash Map'e 4 eleman eklendi ve map size'i gözlendi. Sonra aynı key'e sahip farklı bir value ile ekleme yapıldı ve o key'e ait value nun değiştiği gözlendi. Ardından 2 adet eleman map'ten silindi ve size gözlendi.



- Remove methodu Unit testinde bir eleman eklendi ve bu eleman remove edildi. Methodun return değeri silinen elemanı döndüreceği için bu return değeri ile beklenen return değeri karışlaştırıldı. Ardından aynı key tekrar remove edilmeye çalışıldı ve mapte böyle bir eleman olmadığı için null döndürdü ve bu gösterildi.
- Get methodu Unit testinde mape birkaç eleman eklendi ve bunlardan biri get edildi. Beklenen değer ile karşılaştırıldı ve methodun çalıştığı gösterildi. Bir de mapte olmayan bir eleman get edildi ve null döndürdüğü gösterildi.
- Put methodu Unit testinde mapte var olan bir elemanın key i ile eleman eklendi ve put methodnun return değeri olan eklenecek key'in value' su gözlendi.

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 Algortihms

3.1 MergeSort with DoubleLinkedList

3.1.1 Pseudocode and Explanation

MergeSortDLL class has a public wrapper mergeSort method and private merge method.

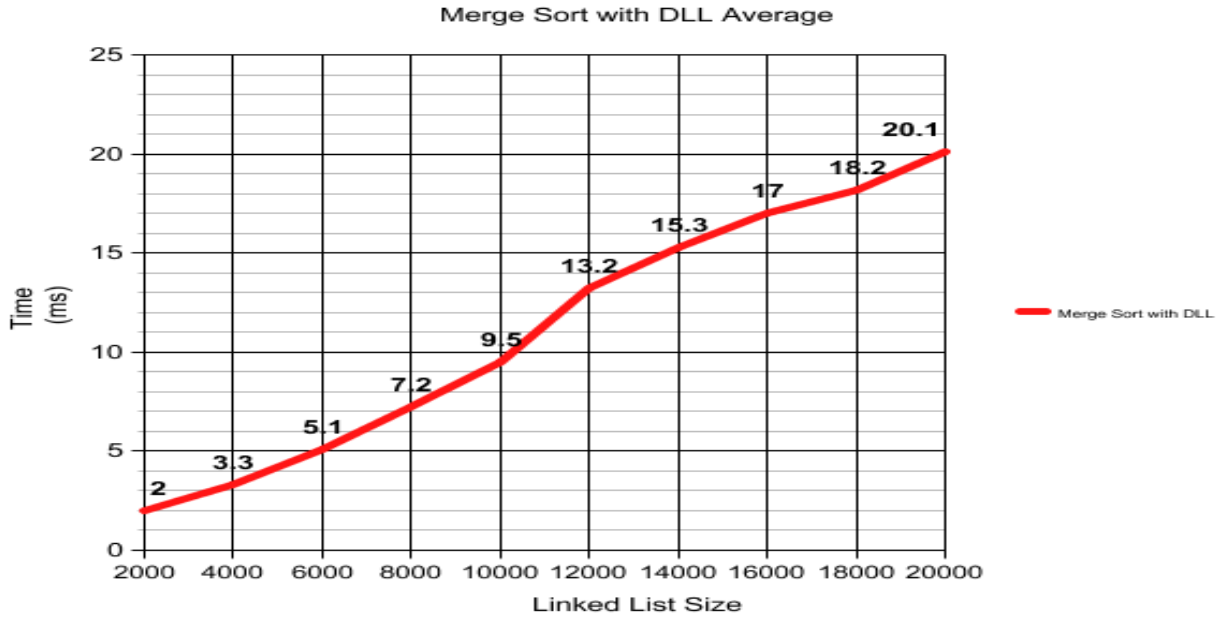
Pseudocode of merge method:

1. Clear output linked list
2. while not finished with either sequence
    1. compare current items from the two sequences, copy the smaller current item to the output sequence, and access the next item from the input sequence whose item was copied
3. copy any remaining items from the first sequence to output sequence
4. copy any remaining items from the second sequence to output sequence

Pseudocode of mergeSort method:

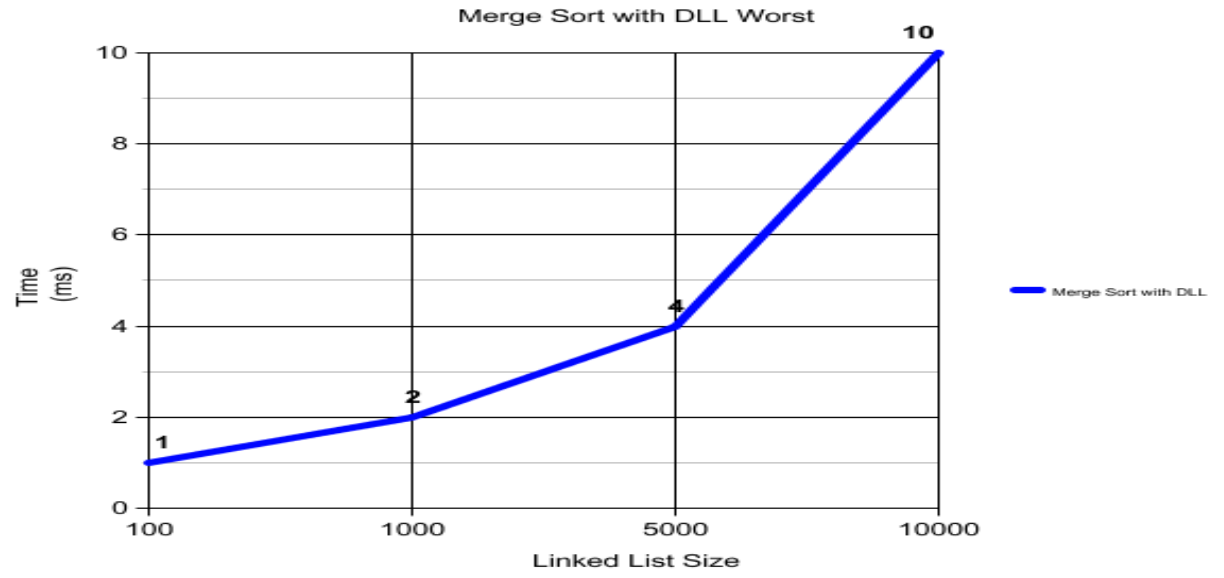
1. Split the linked list into two halves
2. Sort the left half
3. Sort the right half
4. Merge the two

3.1.2 Average Run Time Analysis



Merge sort with double linked list average run time graph. Time complextiy $T(N) = \Theta (n.\lg n)$ Normal Merge sorttan farkı linked listten kaynaklı asimptotik notasyonda gösterilmeyen constantlardır ve biraz daha yavaş çalışır.

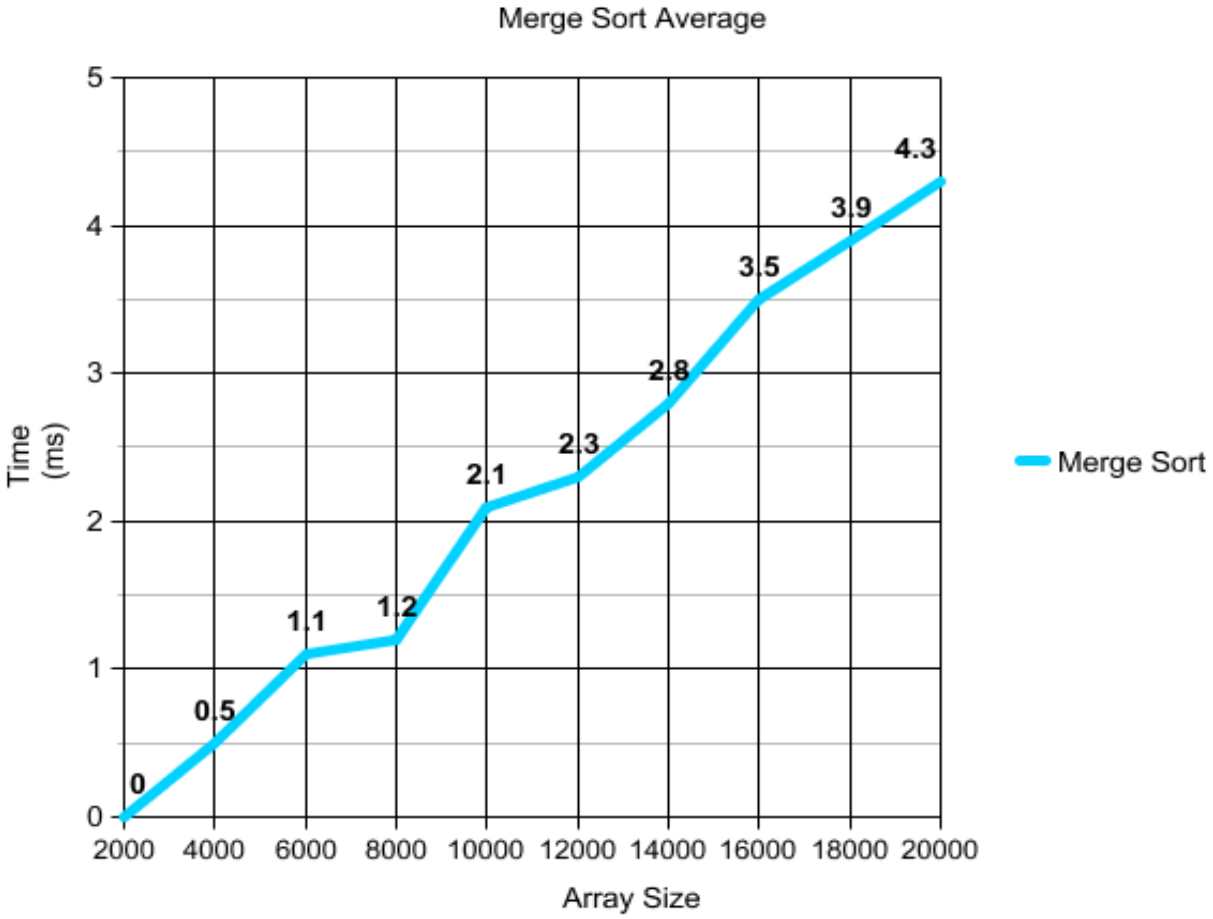
3.1.3 Wort-case Performance Analysis



Merge sort with double linked list worst case run time graph. Merge sort with dll için worst case de average case ile aynıdır. Time complextiy $T(N) = \Theta (n.\lg n)$. Ters sıralı bir array merge sort dll için worst casedir.

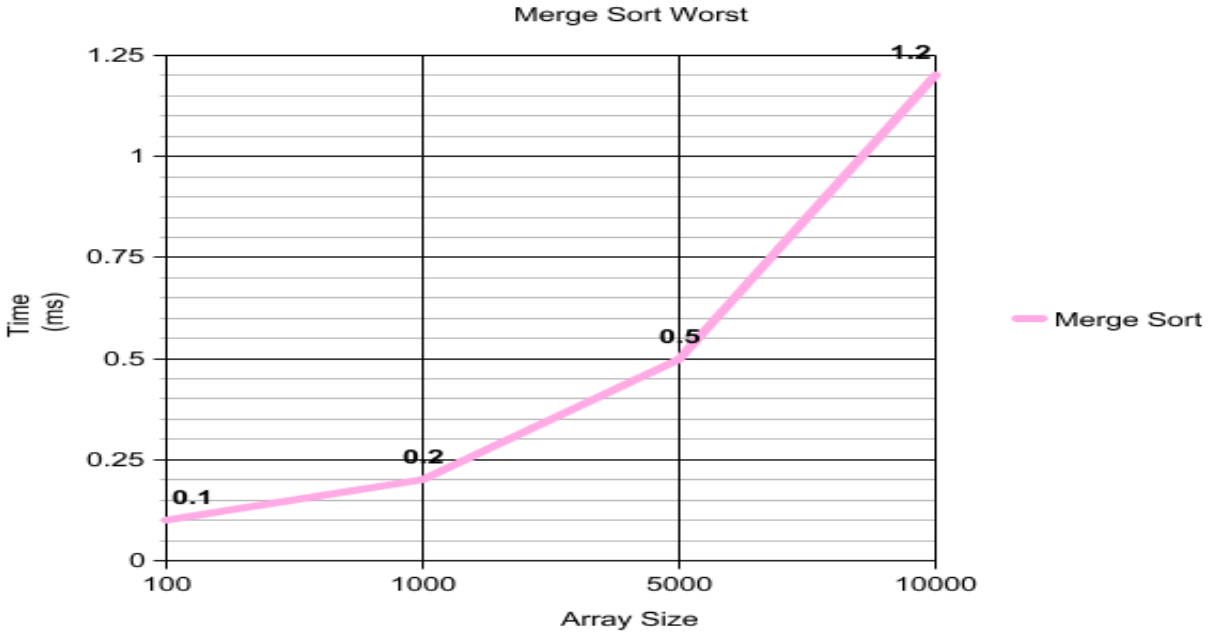
3.2 MergeSort

3.2.1 Average Run Time Analysis



Merge Sort average run time graph. Time complextiy $T(N) = \Theta(n.\lg n)$.

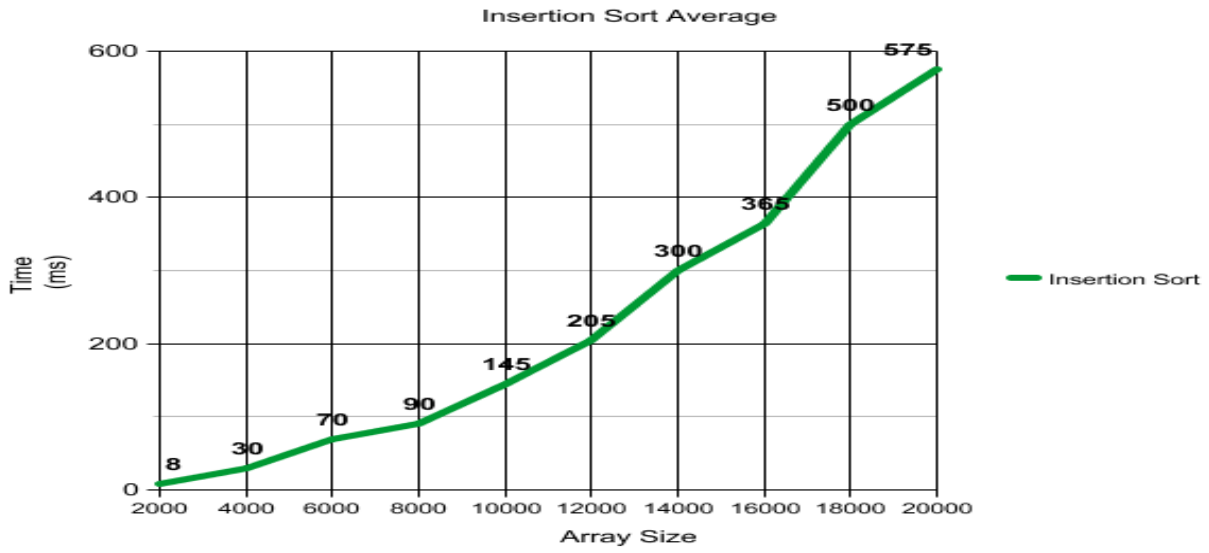
3.2.2 Wort-case Performance Analysis



Merge Sort worst case run time graph. Merge sort için worst case de average case ile aynıdır. Time complextiy $T(N) = \Theta(n.\lg n)$. Ters sıralı bir array merge sort için worst casedir.

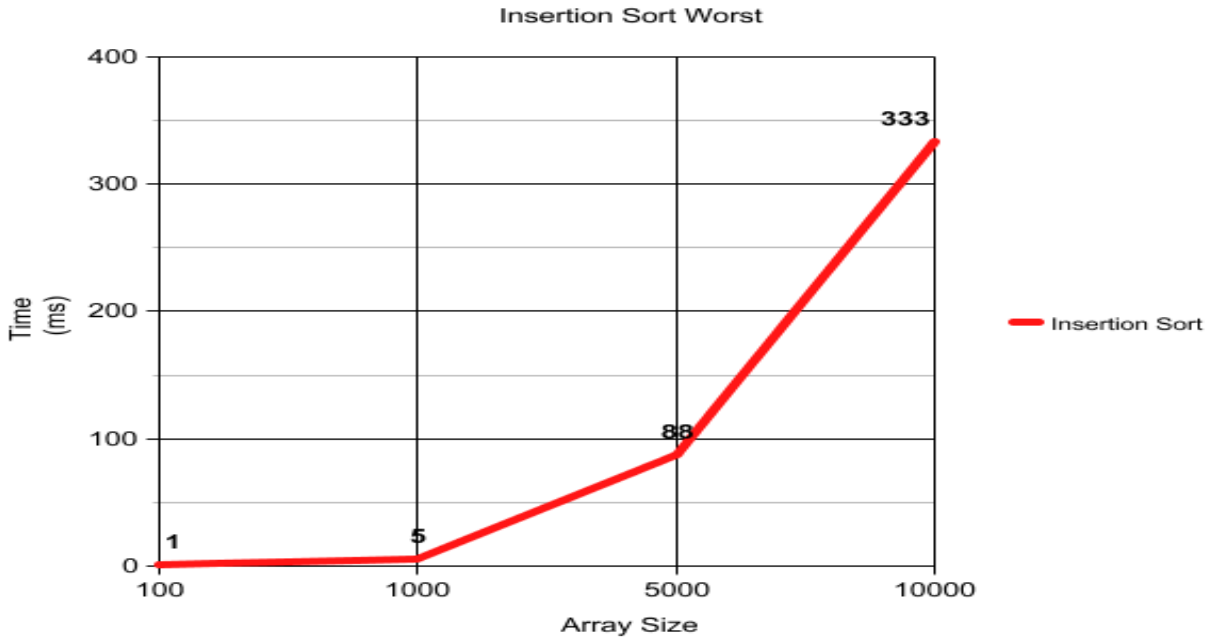
3.3 Insertion Sort

3.3.1 Average Run Time Analysis



Insertion Sort average run time graph. Time complextiy T(N) = Θ (n^2). Görüldüğü gibi array size'ı 2'ye katlandığında süre yaklaşık 4' e katlanmaktadır.

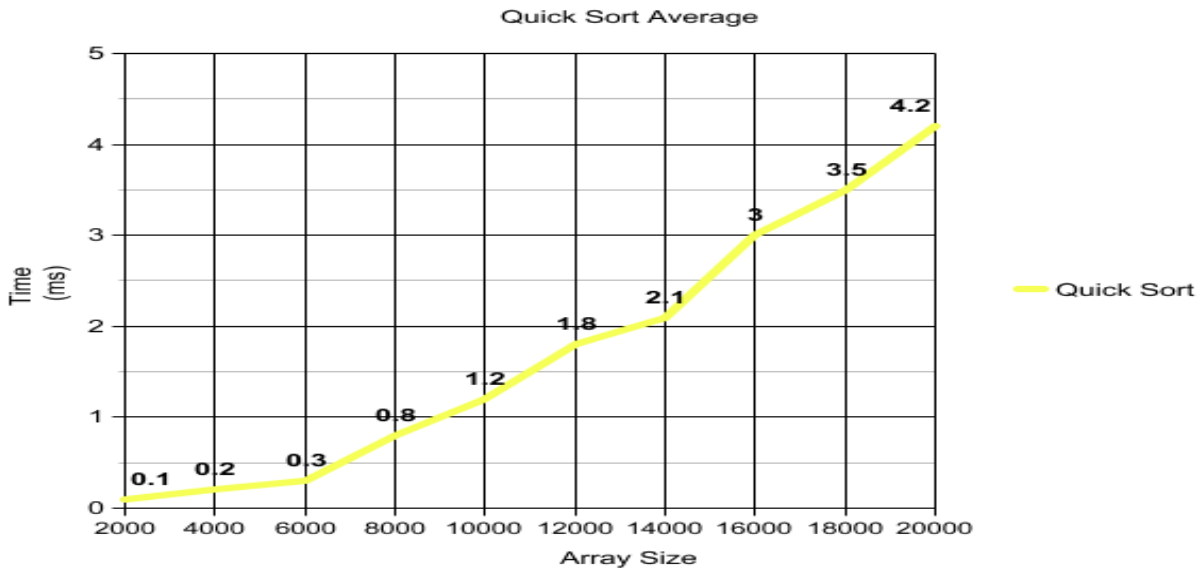
3.3.2 Wort-case Performance Analysis



Insertion Sort worst case run time graph. Insertion sort için worst case de average case ile aynıdır. Time complextiy T(N) = Θ (n^2). Ters sıralı bir array insertion sort için worst casedir.

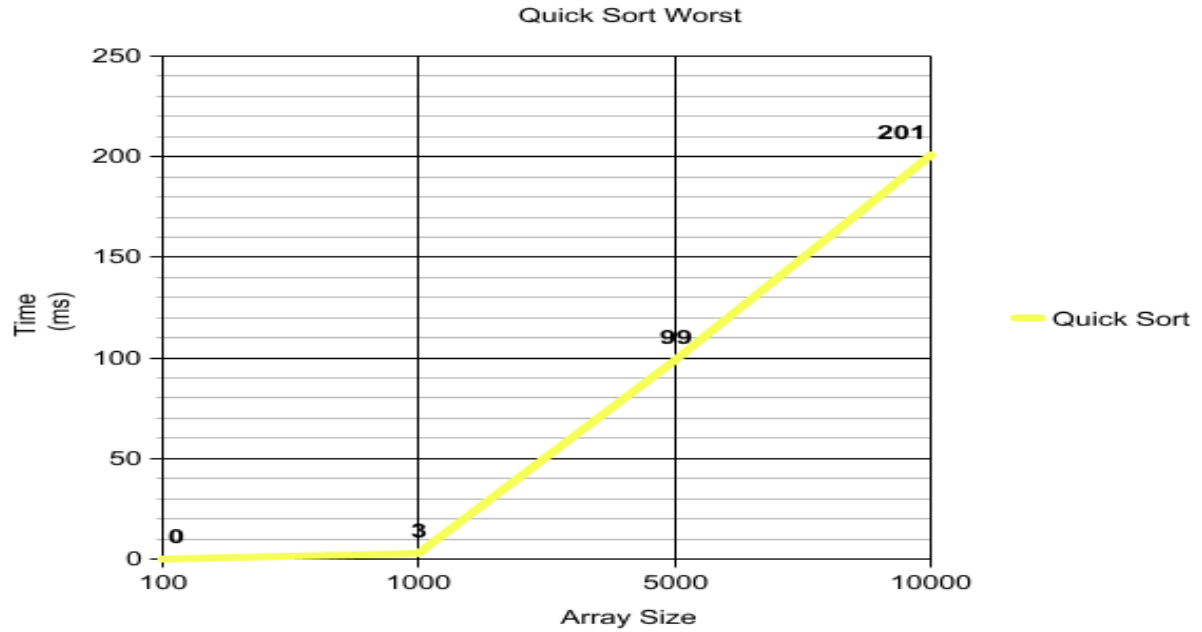
3.4 Quick Sort

3.4.1 Average Run Time Analysis



Quick sort average case run time graph. Time complextiy $T(N) = \Theta (n.\lg n)$.

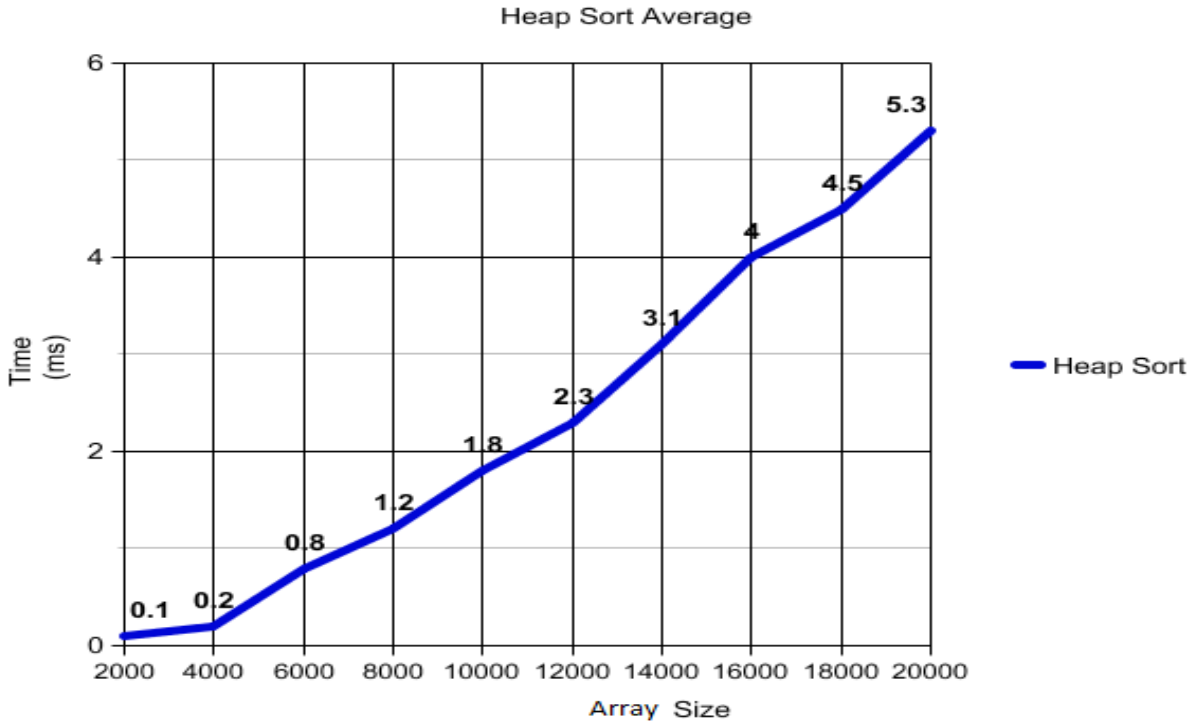
3.4.2 Wort-case Performance Analysis



Quick sort worst case run time graph. Time complextiy $T(N) = \Theta (n^2)$. Quick sort için pivotun en küçük eleman seçilmesi worst case durumudur. Bu durumda çok kötü performans vermektedir.

3.5 Heap Sort

3.5.1 Average Run Time Analysis

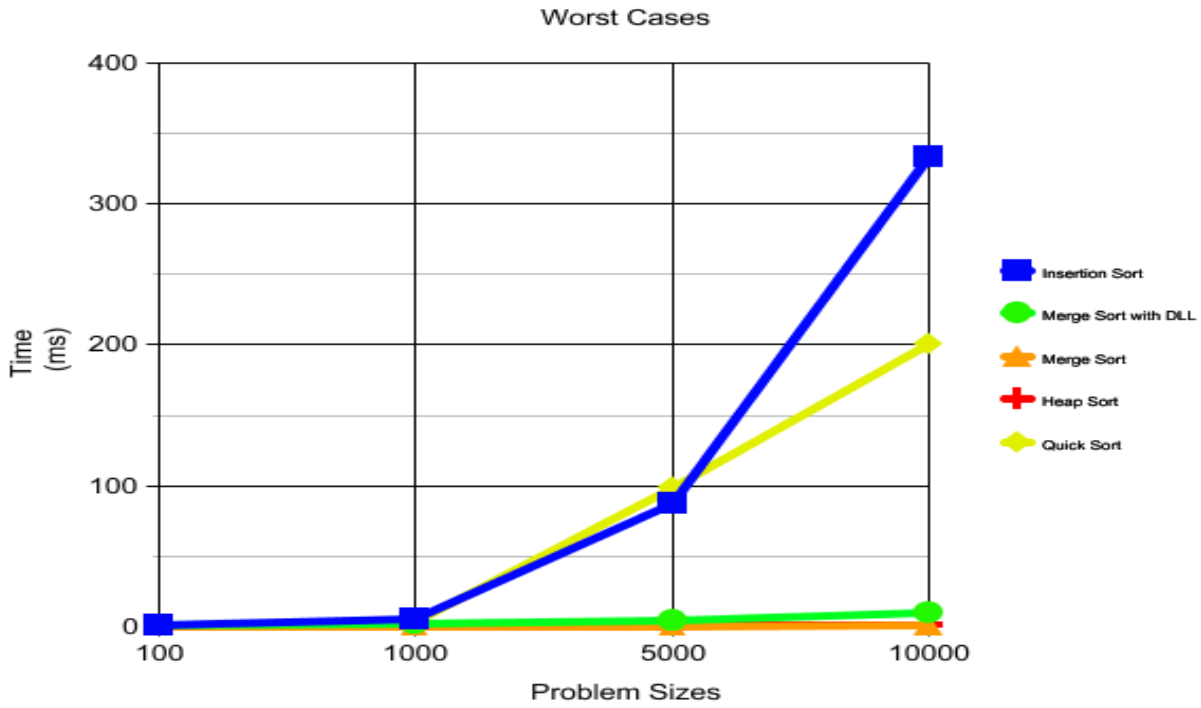


Heap Sort average run time graph. Time complextiy $T(N) = \Theta (n.\lg n)$.

3.5.2 Wort-case Performance Analysis

Heap sort worst case run time graph. Heap sort için worst case de average case ile aynıdır. Time complextiy $T(N) = \Theta (n.\lg n)$. Ters sıralı bir array merge sort için worst casedir.

4 Comparison the Analysis Results



5 sort algoritması için worst case durumlarını 4 farklı size'de gözlemlediğimizde ; insertion ve quick sortun açık ara kötü worst case performanslara sahip olduklarını söyleyebiliriz. Diğer taraftan heap merge ve merge dll sort algoritmaları worst case performansı olarak gayet iyi ve birbirlerine yakın performanslar göstermişler. Bu sonuçların sebebi insertion ve quick sort algoritmalarının worst case time complextiylerinin $\Theta(n^2)$, merge, merge with dll ve heap sort algoritmalarının ise $\Theta(n \lg n)$ olmasıdır.