Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

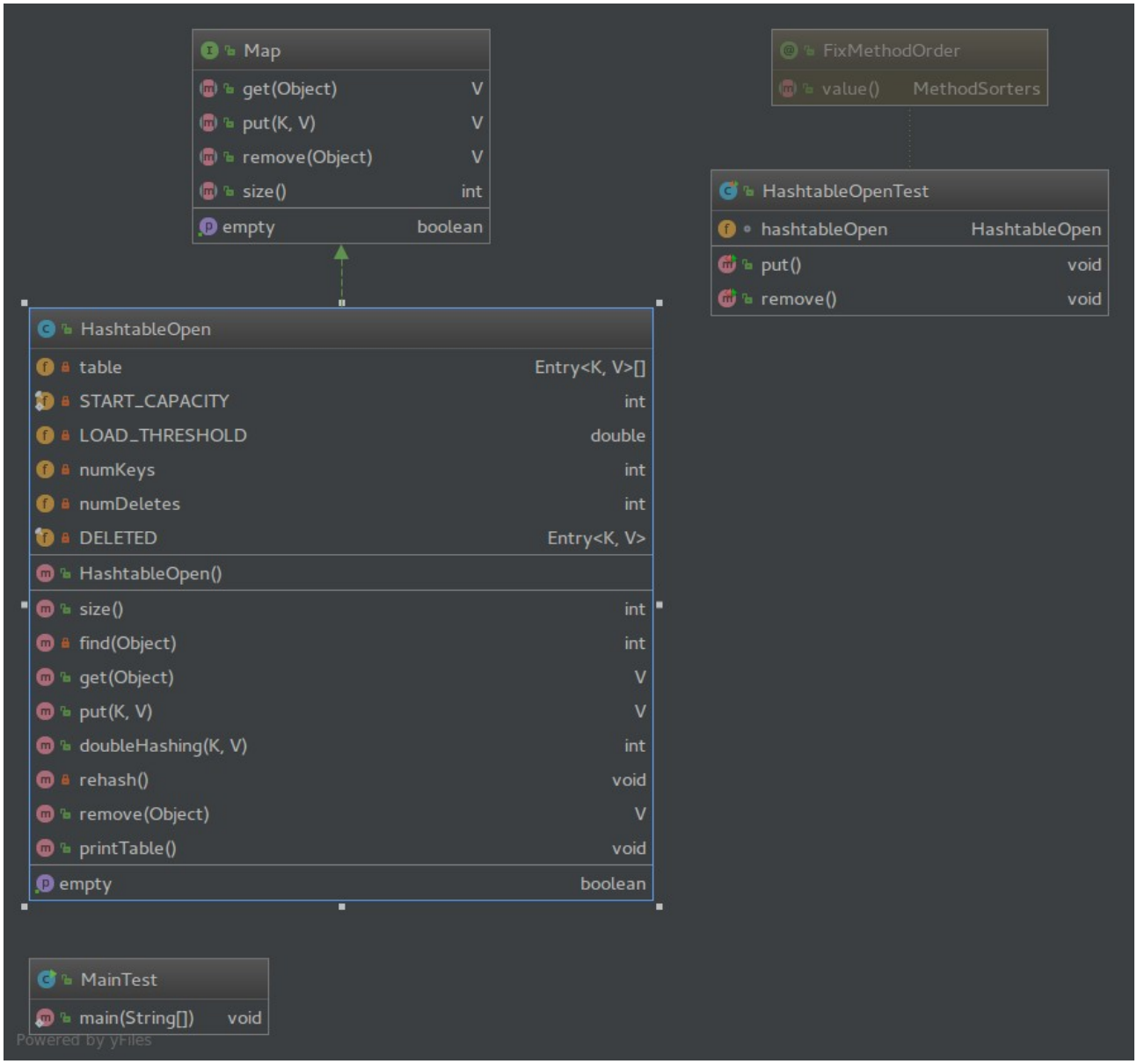
STUDENT NAME: YAĞMUR KARAMAN
STUDENT NUMBER: 141044016

Course Assistant: Fatmanur Esirci

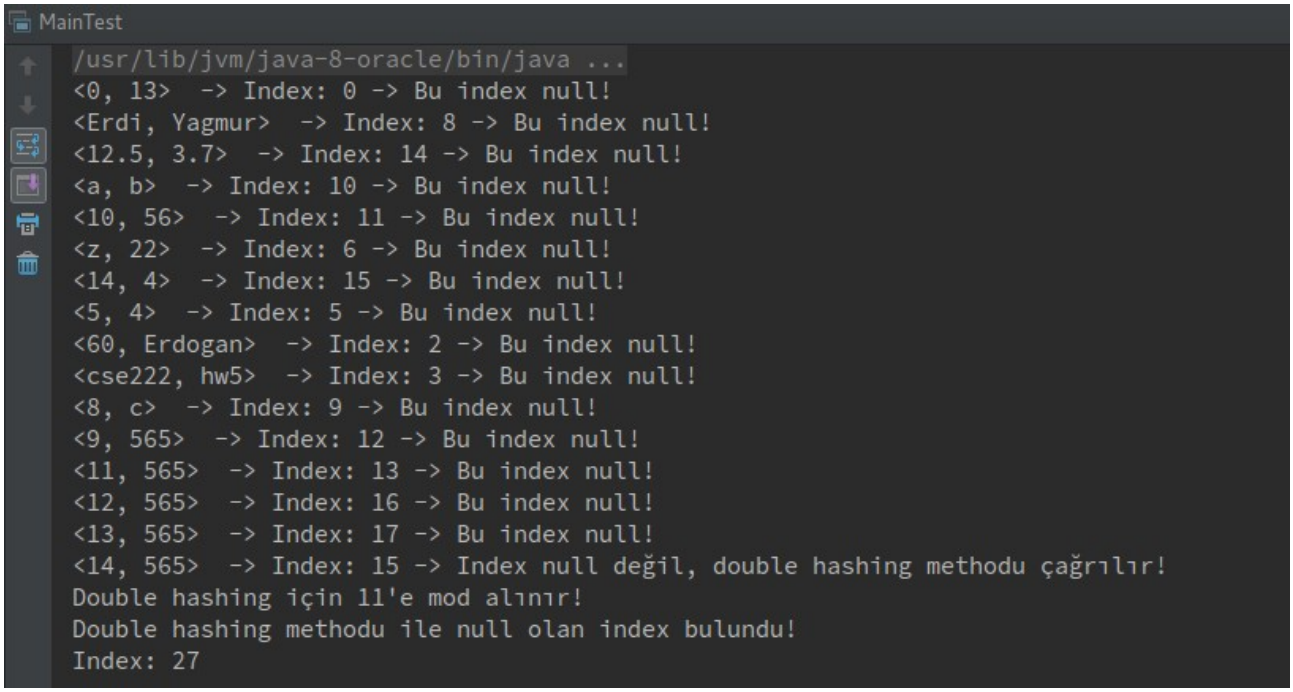
1 Double Hashing Map

1.1 Pseudocode and Explanation

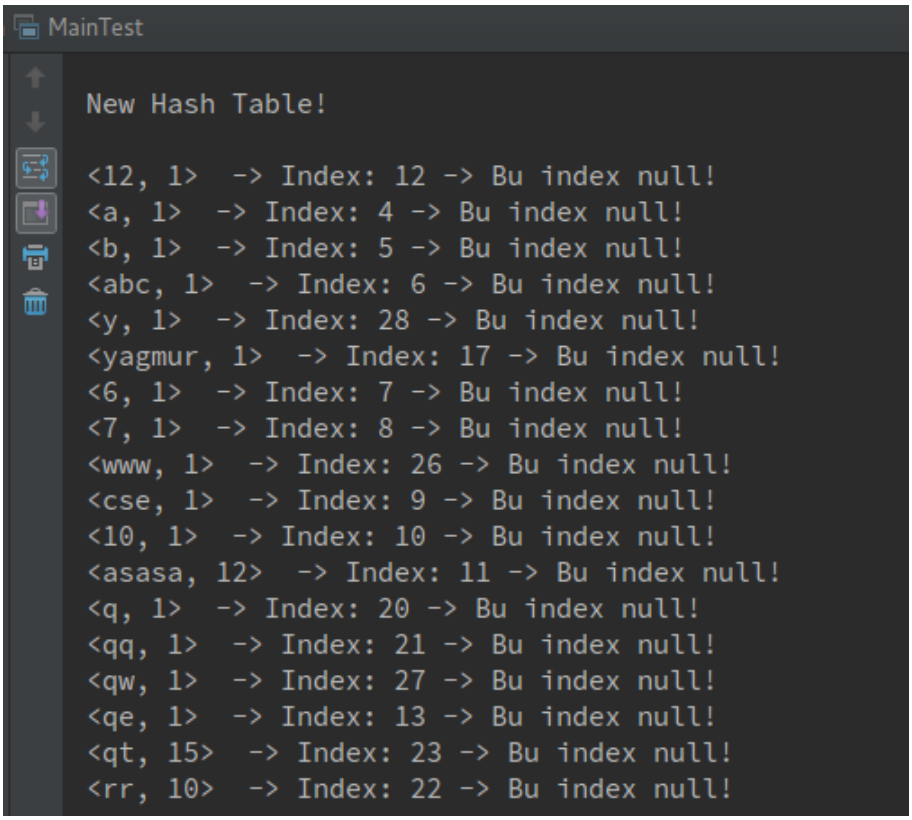
1. put methodu ile <K,V> eklenmek istendiğinde find(key) methodu ile key'in hashCode() methodu çağrılır ve index hesaplanır.
2. table[index] null ise gelen <K,V> entry'si bu index'e koyulur.
3. table[index] null değil ise doubleHashing(key, value) methodu çağtılır ve key ile yeni index hesaplanır.
4. table[index] null olmadığı sürece bu işlem sürekli yapılır.
5. null olan index bulunduğunda da Entry o index'e koyulur.



1.2 Test Cases



1. Bu test için table size 29 olarak belirlenmişti.
2. <14, 565> entry'si geldiğinde index 15 olarak hesaplandı ve bu index boş olmadığı için double hash methodu kullanıldı, yeni index 27 olarak bulundu.

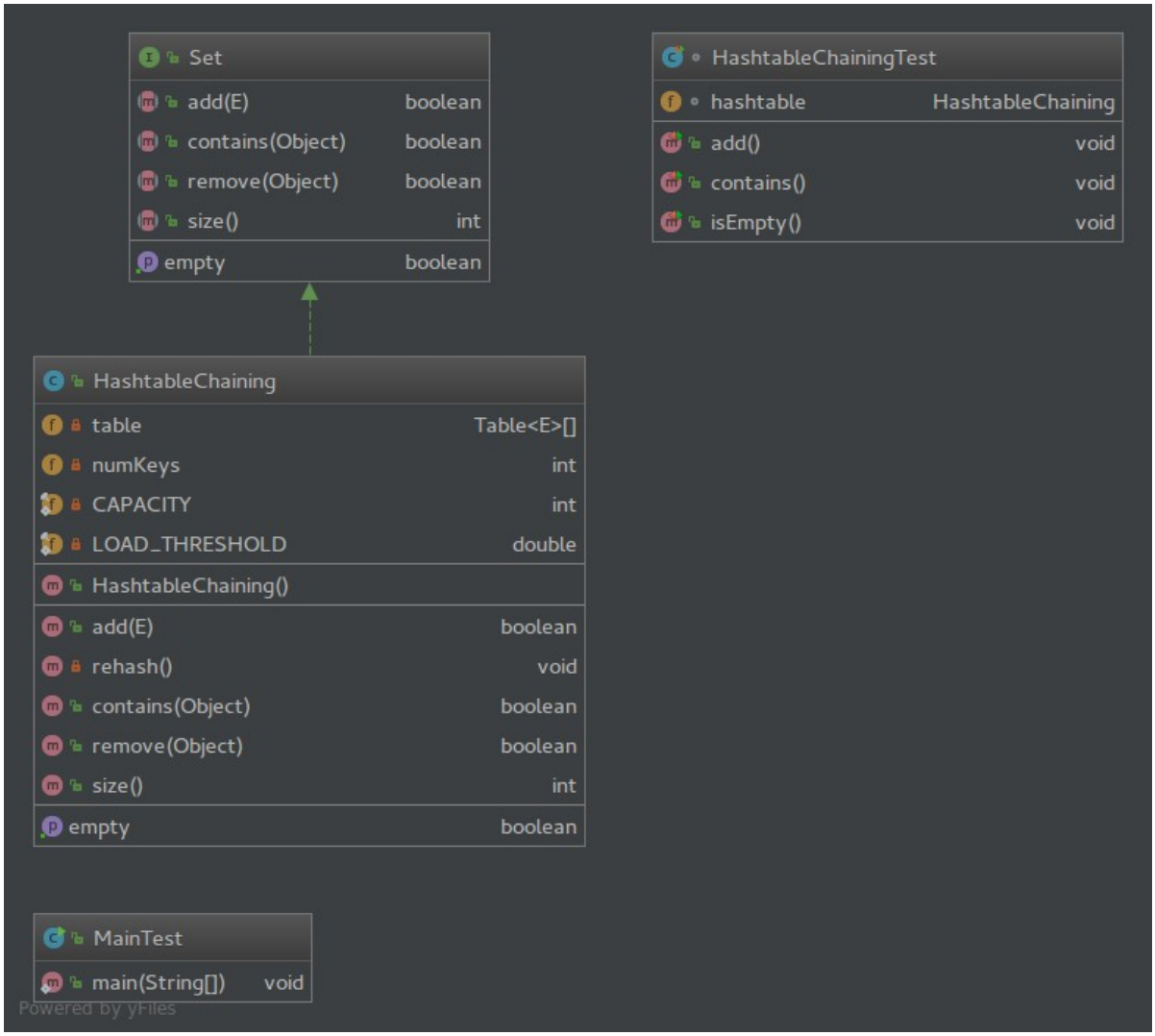


1. Bu test için hash table size 31 olarak belirlenmişti.
2. Burada herhangi bir collision olmadı.

2 Recursive Hashing Set

2.1 Pseudocode and Explanation

1. Hash Table'a yeni bir eleman eklenmek istenildiğinde, key.hashCode() % table.length ile index hesapladım.
2. Hesaplanan index tabloda boş ise elemanı buraya ekledim.
3. Boş değilse yeni bir tablo yer aldım, yeni tabloyu eski tablonun next'i olarak oluşturdum. Elemanı bu tablodaki indexe ekledim.



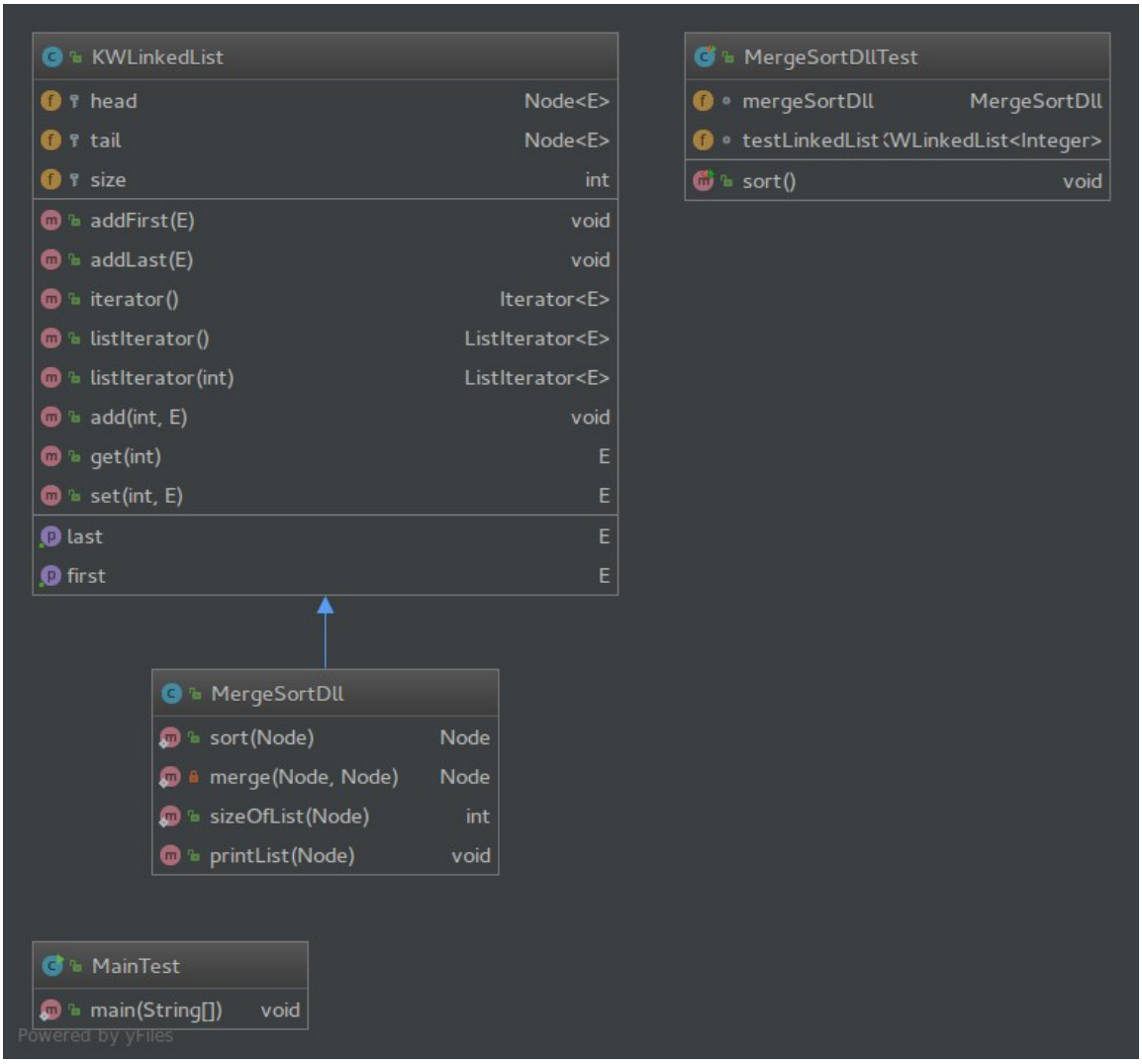
3 Sorting Algortihms

3.1 MergeSort with DoubleLinkedList

3.1.1 Pseudocode and Explanation

Pseudocode:

1. Assignment *head* of list to temp
2. Set *halfSize* to *linkedListSize* divided by 2
3. if *next of head* is *null*
4. R*eturn head of linkedlist*
5. *while halfSize-1* is bigger than 0
6. Assignment next of temp to temp
7. Decrement *halfSize*
8. Assignment head of right part of list, next of temp
9. Assignment null to next of temp
10. Assignment head of list to temp
11. Recursively apply the merge sort algorithm to temp
12. Recursively apply the merge sort algorithm to second
13. Apply the merge method using *temp* and *second* as the input and the original list as the output



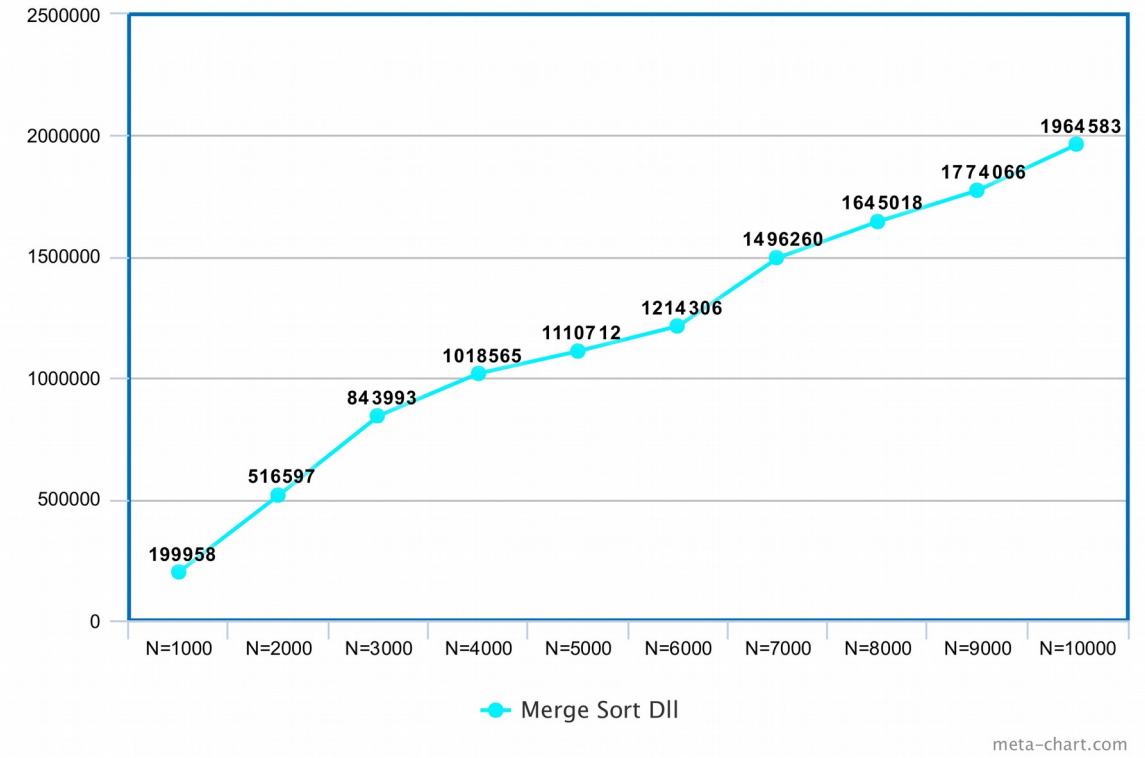
-Double Linked List class'ı için kitaptaki KWLinkedList class'ını kullandım, bu class double linked list yapısını implement etmektedir.

-Merge Sort, çalışma prensibi olarak gelen listi her zaman 2 parçaya ayırır, ayrılacak parça kalmayana kadar. Bunu en küçük parçaya kadar yapar ve en sonda da tüm parçaları sort edip merge ederek sorted list oluşturur.

-Bunu implement ederken her zaman 2 parçaya ayrılan listlerin head'lerini tuttum ve onlarla işlemler yaptım. 2 parçaya ayrılacak list kalmayana kadar sürekli right ve left listler oluşturdum, sonra da merge methoduyla parçalanmış listleri sort edip birleştirdim.

3.1.2 Average Run Time Analysis (ns cinsinden)

-Her bir N değeri için 10 kez çalıştırılıp ortalaması alınmıştır.



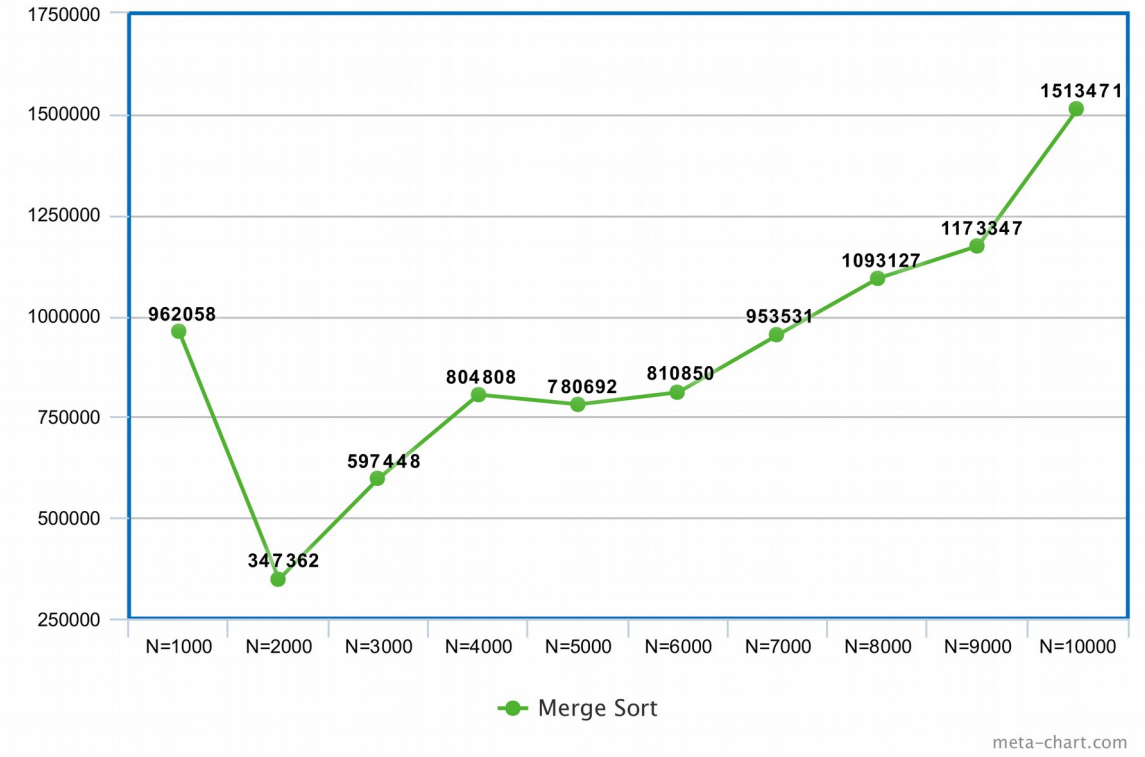
3.1.3 Wort-case Performance Analysis

This part about Question5 in HW5

3.2 MergeSort

3.2.1 Average Run Time Analysis (ns cinsinden)

-Her bir N değeri için 10 kez çalıştırılıp ortalaması alınmıştır.



3.2.2 Wort-case Performance Analysis

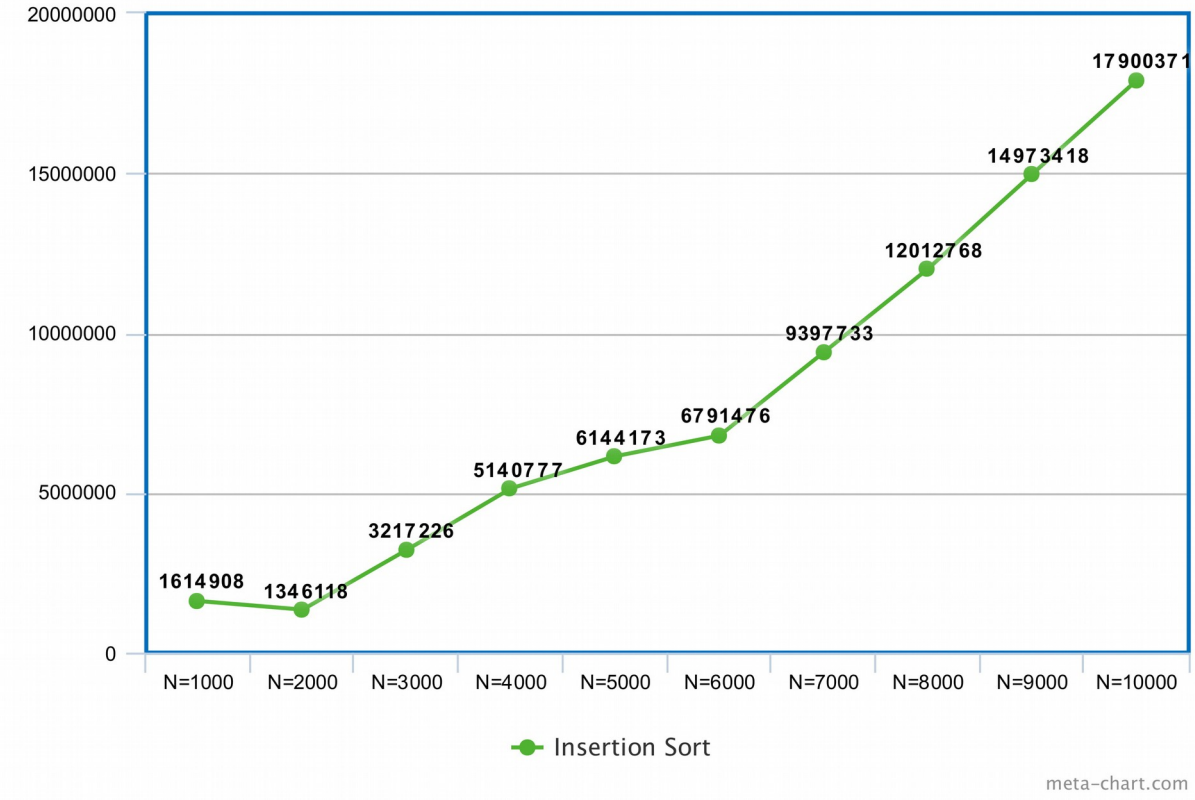
-Worst-case analiz için array tersten sıralandı.

```
-------------------------------------
      RUNNING-TIME FOR SIZE 100
-------------------------------------
MergeSort Time: 236917 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 1000
-------------------------------------
MergeSort Time: 2056490 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 5000
-------------------------------------
MergeSort Time: 1790718 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 10000
-------------------------------------
MergeSort Time: 1325160 ns
```

3.3 Insertion Sort

3.3.1 Average Run Time Analysis (ns cinsinden)

-Her bir N değeri için 10 kez çalıştırılıp ortalaması alınmıştır.



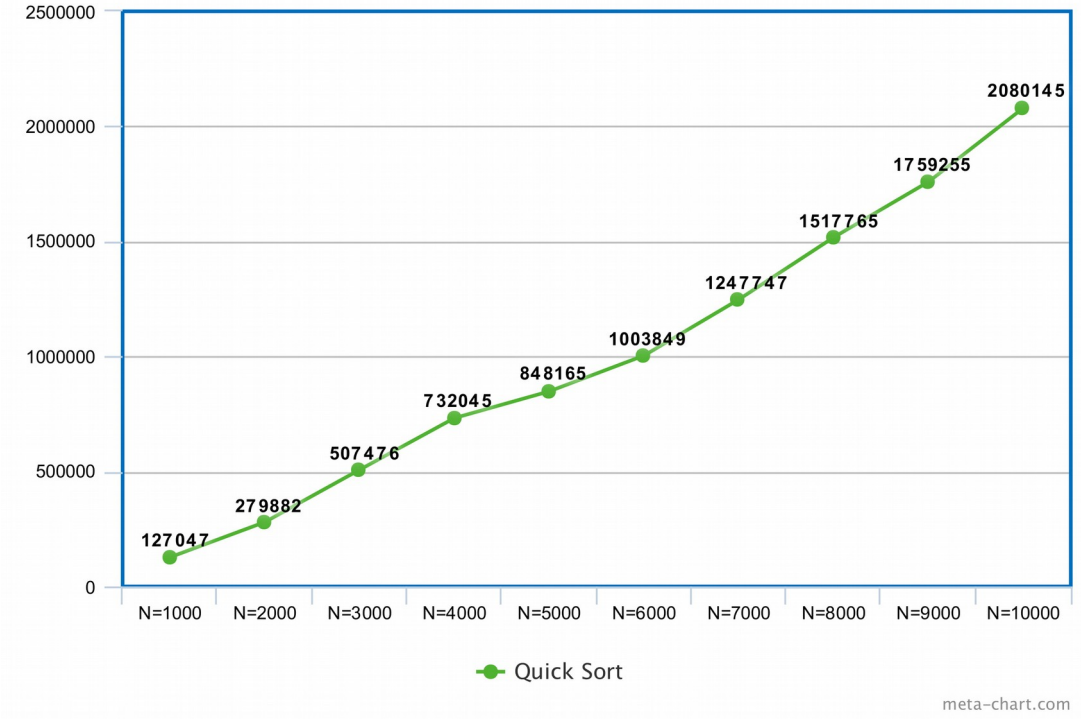
3.3.2 Wort-case Performance Analysis

```
-------------------------------------
      RUNNING-TIME FOR SIZE 100
-------------------------------------
InsertionSort Time: 487544 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 1000
-------------------------------------
InsertionSort Time: 14886994 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 5000
-------------------------------------
InsertionSort Time: 36231217 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 10000
-------------------------------------
InsertionSort Time: 108508861 ns
```

3.4 Quick Sort

3.4.1 Average Run Time Analysis (ns cinsinden)

-Her bir N değeri için 10 kez çalıştırılıp ortalaması alınmıştır.



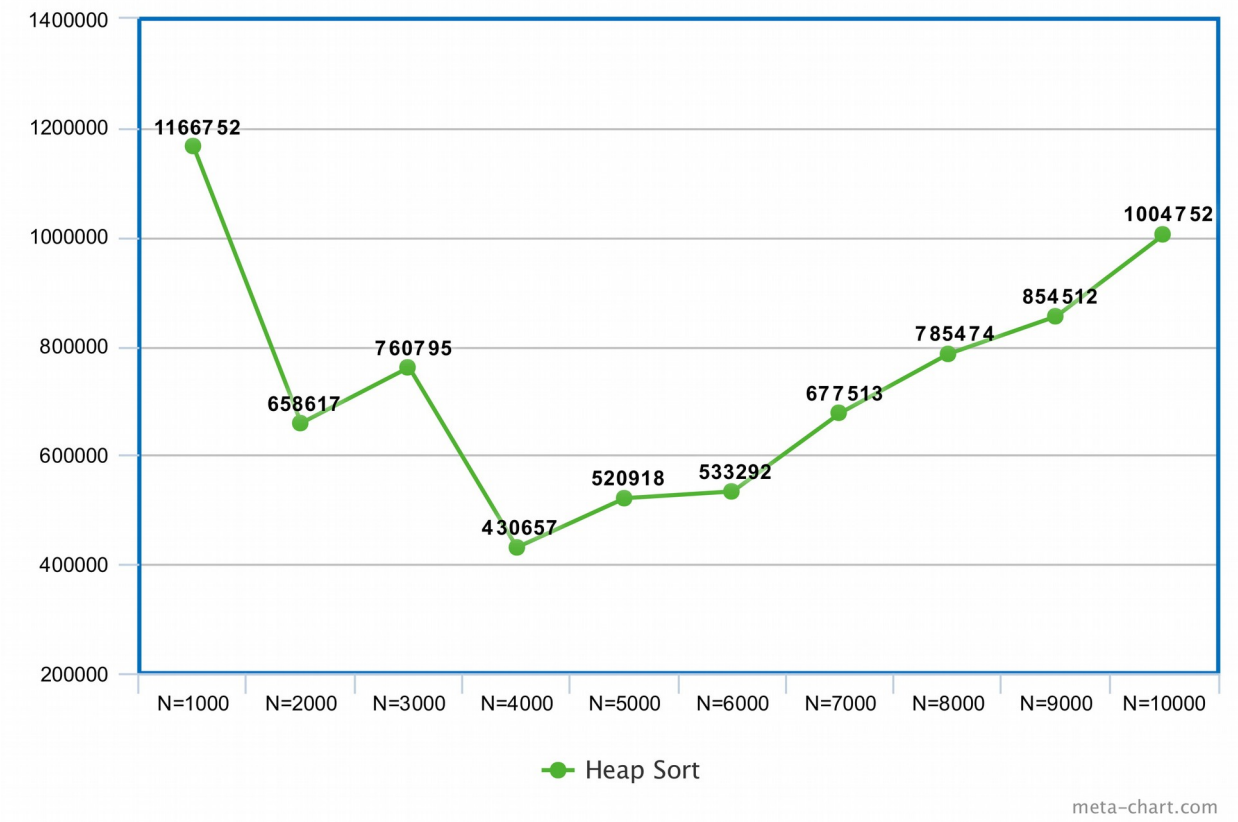
3.4.2 Wort-case Performance Analysis

```
-------------------------------------
      RUNNING-TIME FOR SIZE 100
-------------------------------------
QuickSort Time: 443257 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 1000
-------------------------------------
QuickSort Time: 18660760 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 5000
-------------------------------------
QuickSort Time: 28687051 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 10000
-------------------------------------
QuickSort Time: 107517839 ns
```

3.5 Heap Sort

3.5.1 Average Run Time Analysis (ns cinsinden)

-Her bir N değeri için 10 kez çalıştırılıp ortalaması alınmıştır.



3.5.2 Wort-case Performance Analysis

```
-------------------------------------
      RUNNING-TIME FOR SIZE 100
-------------------------------------
HeapSort Time: 428035 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 1000
-------------------------------------
HeapSort Time: 2976017 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 5000
-------------------------------------
HeapSort Time: 15565608 ns

-------------------------------------
      RUNNING-TIME FOR SIZE 10000
-------------------------------------
HeapSort Time: 9202675 ns
```

4 Comparison the Analysis Results

N değeri yükseldikçe BubbleSort, SelectionSort ve InsertionSort algoritmalarının da çalışma zamanında artış meydana geldi. QuickSort N=5000 değerine kadar SelectionSort'tan daha yavaş gibi görünse de N değeri 10000'e çıktığında SelectionSort'un çalışma süresi QuickSort'u geçti. MergeSort ve ShellSort neredeyse aynı çalışma süresini verdi, grafikte de çizgileri hemen hemen üst üste görülmektedirler. HeapSort, MergeSort ve ShellSort'a göre biraz daha yavaş çıktı.

Double Linked List kullanılarak yapılan Merge Sort tüm algoritmalardan daha yavaş çıktı, çünkü linkedlist üzerinde gezmek, bir elemana ulaşmak linear time süre alır, array olsaydı index ile constant sürede ulaşabilirdik.