

**Gebze Technical University  
Computer Engineering**

**CSE 222 - 2018 Spring**

**HOMEWORK 5 REPORT**

**UĞURKAN ATEŞ  
151044012**

Course Assistant:FATMA NUR ESİRCİ

# 1 Double Hashing Map

This part about Question1 in HW5

## 1.1 Pseudocode and Explanation

I used 3 classes for this part.

MapInterfaceQ1 -> public interface class

This class was guide for later classes implemented this class.

It enforces those who implement this class will map int keys to long values

Has 3 Methods

-> long get key(): Returns value to spesified key.If error returns 0. Throws exception in failure

-> void put(key,value): Putting key value pair into with proper probing.

-> int size(): Returns number of pairs

OpenAdressQ1 -> class implements interface

This class implements methods in interface.It also has a inner (protected) class called Pair to keep objects in structure.

Pair Inner Class has basic getKey,getValue ,constructor and toString methods that expected from this kind of structure.

->ReHashing method used for getting new elements when to transfer items.

-> probing method used for looking up array cells implemented in design.

-> hash method returns hash code(array index) for a key in array

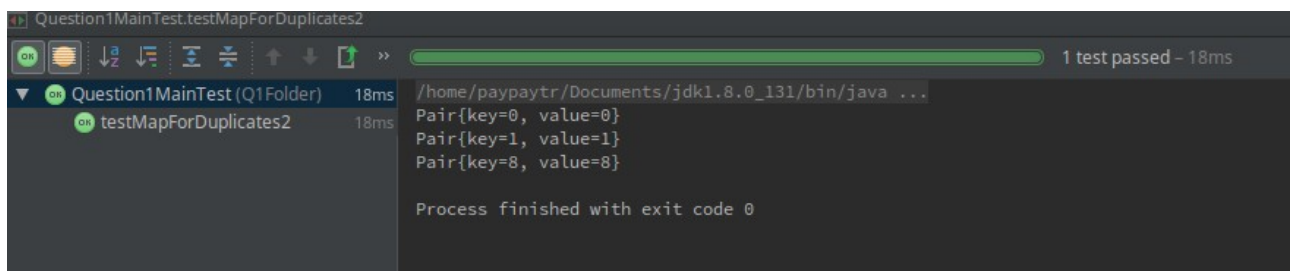
->printer method allows to all map elements printed correctly given toString methods of Pair no param Constructor getting array with default capacity.

Size,get,put methods are fairly simple methods do their jobs with error checks.

Put method calls rehash inside if needed in.

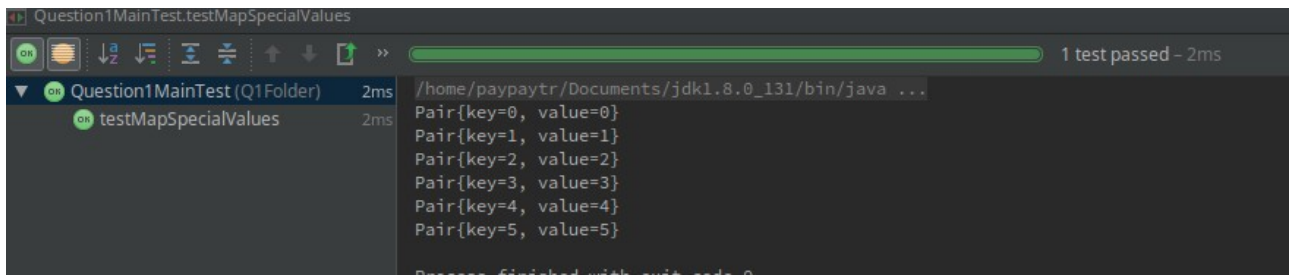
Question1MainTest contains mainTest and UnitTests with Junit. In this section we tested more than 2 hash table.

## 1.2 Test Cases



The screenshot shows an IDE window with the title "Question1MainTest.testMapForDuplicates2". The interface includes a toolbar with icons for running tests, a progress bar indicating "1 test passed - 18ms", and a test results pane. The results pane shows a tree view with "Question1MainTest (Q1Folder)" and "testMapForDuplicates2", both marked as passed with green checkmarks and a duration of 18ms. The output pane displays the following text: 

```
/home/paypaytr/Documents/jdk1.8.0_131/bin/java ...  
Pair{key=0, value=0}  
Pair{key=1, value=1}  
Pair{key=8, value=8}  
  
Process finished with exit code 0
```

A screenshot of an IDE's test runner window. The title bar reads 'Question1MainTest.testMapSpecialValues'. Below the title bar is a progress bar and the text '1 test passed - 2ms'. The left pane shows a tree view with 'Question1MainTest (Q1Folder)' and 'testMapSpecialValues', both marked with green 'OK' icons and '2ms' duration. The right pane shows the test output, which is a list of six pairs: 'Pair{key=0, value=0}', 'Pair{key=1, value=1}', 'Pair{key=2, value=2}', 'Pair{key=3, value=3}', 'Pair{key=4, value=4}', and 'Pair{key=5, value=5}'.

```
Question1MainTest.testMapSpecialValues
1 test passed - 2ms
/home/paypaytr/Documents/jdk1.8.0_131/bin/java ...
Pair{key=0, value=0}
Pair{key=1, value=1}
Pair{key=2, value=2}
Pair{key=3, value=3}
Pair{key=4, value=4}
Pair{key=5, value=5}
```

## 2 Recursive Hashing Set

This part about Question2 in HW5

### 2.1 Pseudocode and Explanation

Empty.

### 2.2 Test Cases

Not done

## 3 Sorting Algorithms

### 3.1 MergeSort with DoubleLinkedList

This part about Question3 in HW5

#### 3.1.1 Pseudocode and Explanation

For this part I implemented basic Node class to work with my specific needs in Double linked list.

We keep head node as value for sort start so its important to keep track of head.

-> printForward will print values using forward travel method

-> printBackward will print values backward travel method. Both starting from given node.

-> halfer method will split double link list into 2 half sized double linked list.

-> merge method will get two linked list as starting head nodes of these linked lists.

I will look if any of them are empty at first so error check always enabled.

So after that comparing two nodes and finding smaller one. Call merge method on smaller one until it merges with second (small merges with bigger)

-> mergeSort will check errors at first. Recur for left and right sides of list. Then use merge methods them to connect. This method will continue until recursion is stopped -> either of them or node.next==null.

### 3.1.2 Average Run Time Analysis

I tried 10 different sized array filled randomly here and assigned DLList.

Sizes are 10,50,100,1000,5000,10 000,5,15,7

Each sized array sent 10 times. For each run all insertions used.

Results :

Size 10(first run so little slow)

```
11038
MERGED DOUBLE LINKED LIST, Sort
1014
```

size 50

```
11038
MERGED DOUBLE LINKED LIST, Sort
733
Insertion Sort
```

size 100

```
430124
MERGED DOUBLE LINKED LIST, Sort
919
```

size 1000

```
MERGED DOUBLE LINKED LIST, Sort
1353
```

size 5000

```
11038
MERGED DOUBLE LINKED LIST, Sort
1420
```

size 10 000

```
11038
MERGED DOUBLE LINKED LIST, Sort
1748
```

size 5

```
MERGED DOUBLE LINKED LIST, Sort
230
```

size 15

```
MERGED DOUBLE LINKED LIST, Sort
235
```

### 3.1.3 Worst-case Performance Analysis

4 Array with size 100,1000,5000,10 000

10 times it run. I tried all sorts and got worst case analysis from all and compared times.

Size 10 000

```
Merged Doubled Linked List Sort
8195
```

size 100(program first run)

```
11038
Merged Doubled Linked List Sort
2207
```

```
Merged Doubled Linked List Sort
2212
```

```
Merged Doubled Linked List Sort
3071
```

size 1000

size 5000

## 3.2 MergeSort

This part about code in course book.

### 3.2.1 Average Run Time Analysis

I tried 10 different sized array filled randomly here and assigned DLList.

Sizes are 10,50,100,1000,5000,10 000,5,15,7

Each sized array sent 10 times. For each run all insertions used.

Results :

```
Merge Sort
26125
Merge Sort
28745
```

```
Merge Sort
1701113
```

```
Merge Sort
4248708
```

```
Merge Sort
393
```

```
Merge Sort
236
```

```
Merge Sort
1838574
```

```
Merge Sort
236
```

### 3.2.2 Worst-case Performance Analysis

4 Array with size 100,1000,5000,10 000  $O(n \log(n))$

10 times it run. I tried all sorts and got worst case analysis from all and compared times.

```
Merge Sort
335114
```

```
Merge Sort
3846569
```

```
Merge Sort
24258019
```

```
Merge Sort
2779762
```

## 3.3 Insertion Sort

### 3.3.1 Average Run Time Analysis

I tried 10 different sized array filled randomly here and assigned DLList.

Sizes are 10,50,100,1000,5000,10 000,5,15,7

Each sized array sent 10 times. For each run all insertions used.

Results :

```
AverageAnalysis
/home/paypaytr/Documents/jdk1.8
Insertion Sort
20146
```

```
Insertion Sort
105567
```

```
Insertion Sort
229155
```

```
Insertion Sort
3878082
```

```
Insertion Sort
3878082
```

```
Insertion Sort
25661128
```

```
Insertion Sort
218
```

```
Insertion Sort
130
```

```
Insertion Sort
475
```

```
Insertion Sort
240
```

### 3.3.2 Worst-case Performance Analysis

4 Array with size 100,1000,5000,10 000 worst =  $O(n^2)$

10 times it run. I tried all sorts and got worst case analysis from all and compared times.

```
Insertion Sort
2371487
```

```
Insertion Sort
21370894
```

```
Insertion Sort
29793738
```

```
Insertion Sort
121642425
```

## 3.4 Quick Sort

### 3.4.1 Average Run Time Analysis

I tried 10 different sized array filled randomly here and assigned DLList.

Sizes are 10,50,100,1000,5000,10 000,5,15,7

Each sized array sent 10 times. For each run all insertions used.

Results :

Quick Sort 26240	Quick Sort 95359	Quick Sort 1255219
Quick Sort 3574795	Quick Sort 34875907	Quick Sort 103445292
Quick Sort 306	Quick Sort 191	Quick Sort 550
Quick Sort 267		

### 3.4.2 Worst-case Performance Analysis

4 Array with size 100,1000,5000,10 000

$O(n^2)$

10 times it run. I tried all sorts and got worst case analysis from all and compared times.

Quick Sort 543809	Quick Sort 32561270
Quick Sort 49184374	Quick Sort 127078422

## 3.5 Heap Sort

### 3.5.1 Average Run Time Analysis

I tried 10 different sized array filled randomly here and assigned DLLList.

Sizes are 10,50,100,1000,5000,10 000,5,15,7

Each sized array sent 10 times. For each run all insertions used.

Results :

Heap Sort 32365	Heap Sort 101174	Heap Sort 277047
Heap Sort 2502715	Heap Sort 2502715	Heap Sort 1791420
Heap Sort 2258045	Heap Sort 358	Heap Sort 201
Heap Sort 770	Heap Sort 342	

### 3.5.2 Worst-case Performance Analysis

4 Array with size 100,1000,5000,10 000 ->  $O(n \log(n))$

10 times it run. I tried all sorts and got worst case analysis from all and compared times.