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 spesificed 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

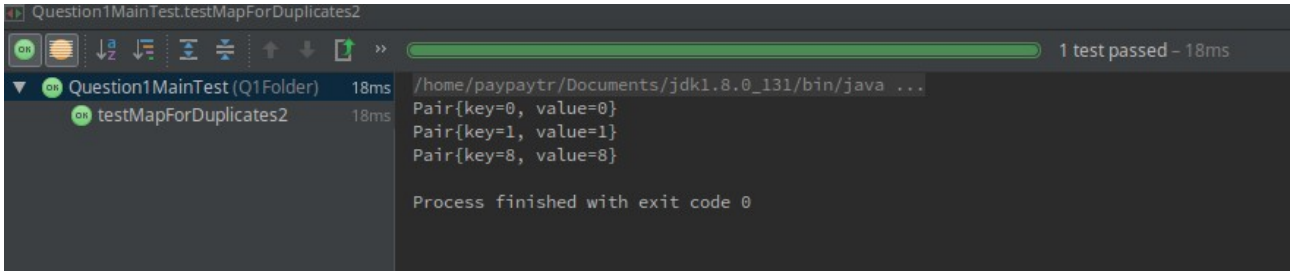
This class implements methods in interface.It also has a inner (protected) class called Pair
to keep objects in structure.

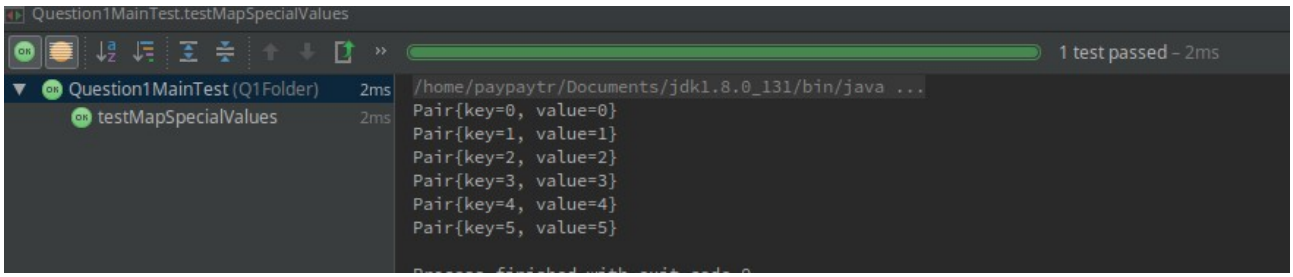
Pair Inner Class has basic getKey,getValue ,constructor and toString methods that expected from this kind of structure.

->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





2 Recursive Hashing Set

This part about Question2 in HW5

2.1 Pseudocode and Explanation

Empty.

2.2 Test Cases

Not done

3 Sorting Algortihms

3.1 MergeSort with DoubleLinkedList

This part about Question3 in HW5

3.1.1 Pseudocode and Explanation

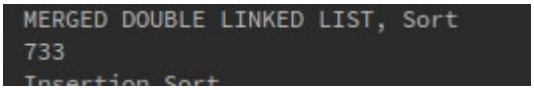
For this part I implemented basic Node class to work with my spesific needs in Double linked list.
We keep head node as value for sort start so its important to keep track of head.
-> printForward will print values using forward travel method
->printBackward will print values backward travel method. 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 recurrsion 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)

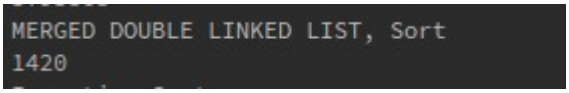
size 50



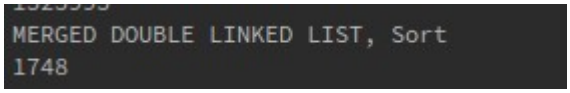
size 100

size 1000

size 5000



size 10 000



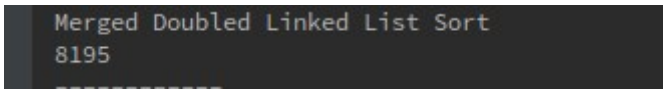
size 15

size 5

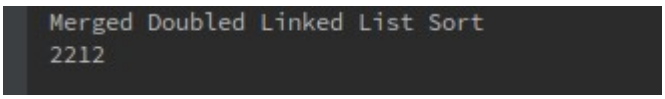
3.1.3 Worst-case Performance Analysis

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.

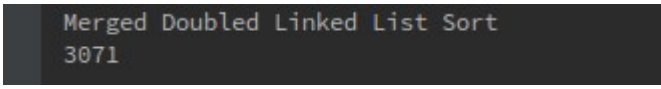
Size 10 000



size 100(program first run)



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 :

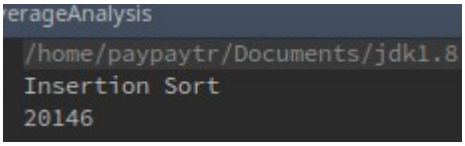
3.2.2 Worst-case Performance Analysis

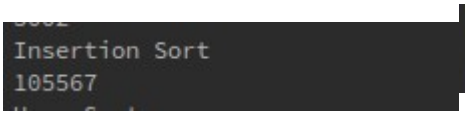
4 Array with size 100,1000,5000,10 000 O(n log (n))
10 times it run. I tried all sorts and got worst case analysis from all and compared times.

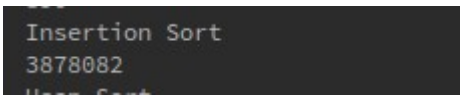
3.3 Insertion Sort

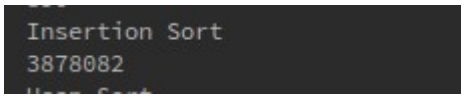
3.3.1 Average Run Time Analysis

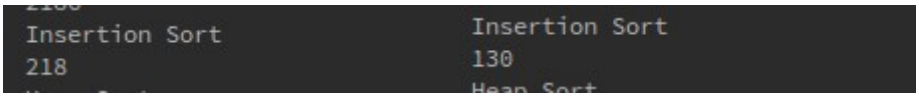
I tried 10 different sized array filled randomly here and assigned DLList.
Sizes are 10,50,100,1000,5000,10 000,5,15,7
Each sized array sent 10 times. For each run all insertions used.
Results :

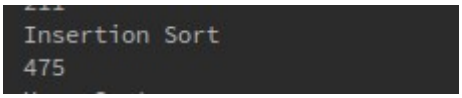






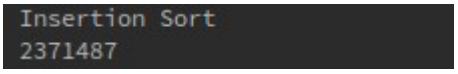






3.3.2 Worst-case Performance Analysis

4 Array with size 100,1000,5000,10 000 worst = o(n2)
10 times it run. I tried all sorts and got worst case analysis from all and compared times.

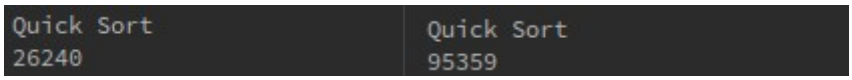


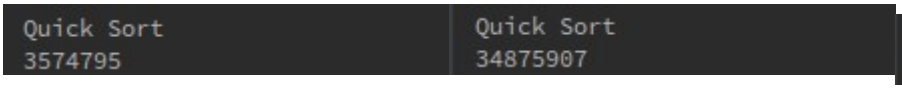
3.4 Quick Sort

3.4.1 Average Run Time Analysis

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

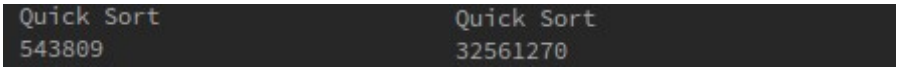
Sizes are 10,50,100,1000,5000,10 000,5,15,7
Each sized array sent 10 times. For each run all insertions used.
Results :





3.4.2 Worst-case Performance Analysis

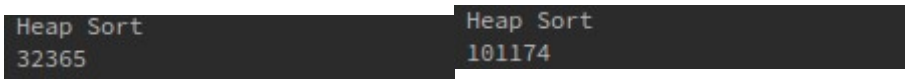
4 Array with size 100,1000,5000,10 000
O (n2)
10 times it run. I tried all sorts and got worst case analysis from all and compared times.

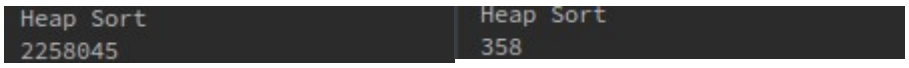


3.5 Heap Sort

3.5.1 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 :





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.