

GTU Department of Computer Engineering

CSE 222/505 - Spring 2022

Homework #6 Report

Serhat SARI
200104004028

1- SYSTEM REQUIREMENTS

a- Non-Functional System Requirements

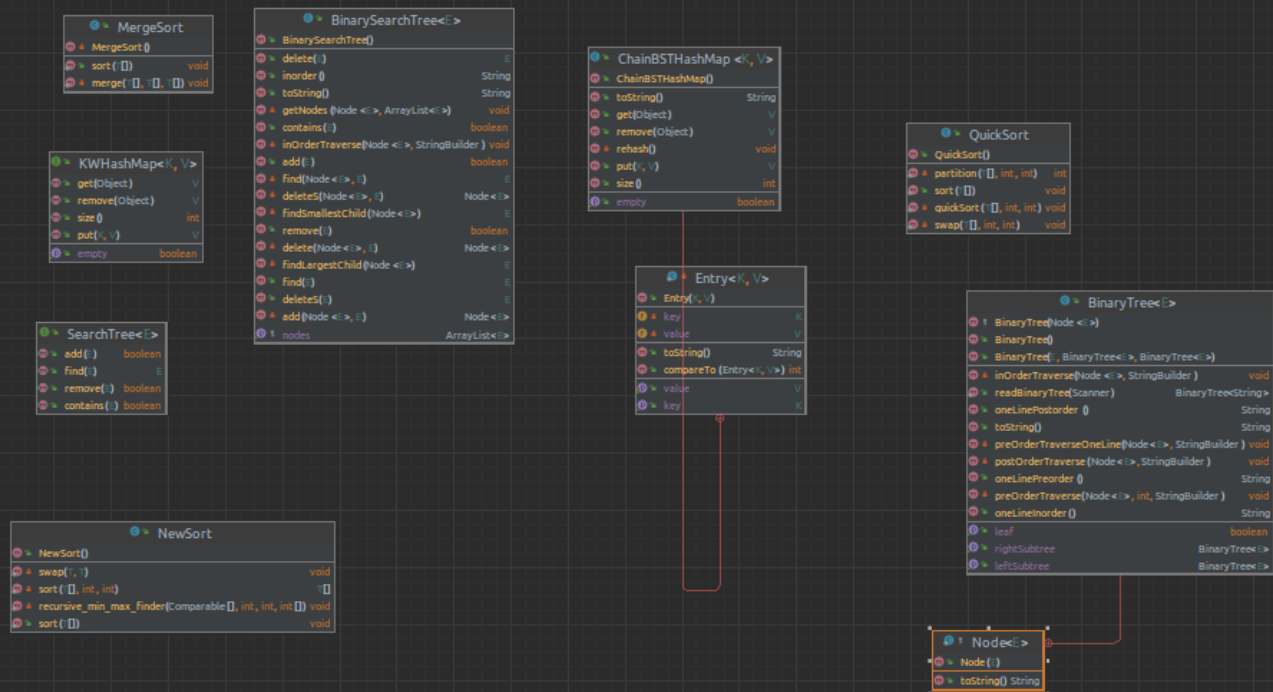
1- Back-end Software : Java 11

2- Software should be able to compile with “javac” on a linux distribution.

b-Functional System Requirements

-1.th Question, you should use Map as it supposed to be.

2- CLASS DIAGRAM



3- Problem Solution Approach

For the first problem, first i have read the book.
Then i started to implement the first one. Q1.1 was easy.
I used BST instead of LinkedList.

For the second problem, i took sorting algorithms from book.
Then i implement the new_sort class which was hard for me to implement.
Because recursive max min finder part was hard.
But eventually i did it.

SORTING ALGORITHM COMPARISON

TESTING SORTING ALGORITHMS

100 random integers

QuickSort: 0.028

MergeSort: 0.032

NewSort: 0.022

1000 random integers

QuickSort: 0.05

MergeSort: 0.113

NewSort: 1.598

10000 random integers

QuickSort: 1.255

MergeSort: 0.771

NewSort: 141.018

Q1. 2.a

Coalesced chaining protects us from the effects of primary and secondary clustering, If the chains are short, this strategy is so efficient.
Deletion from a coalesced hash table is expensive like open addressing way.
Resizing the table is very expensive and should be done so rarely.

Q1.2.b

This technique overcomes the cluster issue.
It is very effective method for resolving collisions.
Double hashing is useful if an application requires a smaller hash table because it can find a empty place in an effective way.

Double hashing can find the next free place faster than the linear probing.
But the computational cost may is higher than linear probing.
Also implementation of double hashing is harder than the others.

TEST CASES



```
1  System.out.println("100 random integers adding to map");
2  double mapTime = 0;
3  long startTime = 0;
4  long endTime = 0;
5  long seconds = 0;
6
7  startTime = System.currentTimeMillis();
8
9  for (int i = 0; i < 1000; i++) {
10     Integer[] table = new Integer[100];
11
12     for (int j = 0; j < table.length; j++) {
13         table[j] = (int) (Math.random() * 100);
14         map.put(table[j], 10);
15     }
16 }
17
```



```
1  System.out.println("1000 random integers adding to map");
2
3  startTime = System.currentTimeMillis();
4
5  for (int i = 0; i < 1000; i++) {
6     Integer[] table = new Integer[1000];
7
8     for (int j = 0; j < table.length; j++) {
9         table[j] = (int) (Math.random() * 100);
10        map.put(table[j], 10);
11    }
12 }
13
```



```
1  System.out.println("1000 random integers adding to map");
2
3  startTime = System.currentTimeMillis();
4
5  for (int i = 0; i < 1000; i++) {
6      Integer[] table = new Integer[10000];
7
8      for (int j = 0; j < table.length; j++) {
9          table[j] = (int) (Math.random() * 100);
10         map.put(table[j], 10);
11     }
12 }
```



```
1  System.out.println("\n\nChainBSTHashMap Get Method");
2  System.out.println("Lets add [1,1], [2,5], [123,3] to the map");
3  map1.put(1, 1);
4  map1.put(2, 5);
5  map1.put(123, 3);
6  System.out.println("Lets print the map");
7  System.out.println(map1);
8  System.out.println("Lets get the value of key 2");
9  System.out.println("Lets print the map");
10 System.out.println(map1.get(2));
11 System.out.println("Lets get the value of key 123");
12 System.out.println("Lets print the map");
13 System.out.println(map1.get(123));
14 System.out.println("Lets get the value of key 1");
15 System.out.println("Lets print the map");
16 System.out.println(map1.get(1));
17 System.out.println("As you can see it works");
```



```
1 System.out.println("\n\nChainBSTHashMap Remove Method");
2 System.out.println("Lets add [1,1], [2,5], [123,3] to the map");
3 System.out.println("Lets print the map");
4 System.out.println(map1);
5 System.out.println("Lets remove the entry of key 2");
6 System.out.println("Lets print the map");
7 map1.remove(2);
8 System.out.println(map1);
9 System.out.println("Lets remove the entry of key 123");
10 System.out.println("Lets print the map");
11 map1.remove(123);
12 System.out.println(map1);
13 System.out.println("Lets remove the entry of key 1");
14 System.out.println("Lets print the map");
15 map1.remove(1);
16 System.out.println(map1);
17 System.out.println("As you can see it works");
18 System.out.println("Lets print the map");
```




```
1  System.out.println("\n\nTESTING SORTING ALGORITHMS\n");
2  System.out.println("100 random integers");
3  double quickSortTime = 0;
4  double mergeSortTime = 0;
5  double newSortTime = 0;
6  long startTime = 0;
7  long endTime = 0;
8  long seconds = 0;
9
10 for (int i = 0; i < 1000; i++) {
11     Integer[] table = new Integer[100];
12
13     for (int j = 0; j < table.length; j++) {
14         table[j] = (int) (Math.random() * 100);
15     }
16     Integer[] table1 = table;
17     Integer[] table2 = table;
18
19     startTime = System.currentTimeMillis();
20     QuickSort.sort(table);
21     endTime = System.currentTimeMillis();
22     seconds = (endTime - startTime);
23     quickSortTime += seconds;
24
25     startTime = System.currentTimeMillis();
26     MergeSort.sort(table1);
27     endTime = System.currentTimeMillis();
28     seconds = (endTime - startTime);
29     mergeSortTime += seconds;
30
31     startTime = System.currentTimeMillis();
32     NewSort.sort(table2);
33     endTime = System.currentTimeMillis();
34     seconds = (endTime - startTime);
35     newSortTime += seconds;
36 }
37
38 System.out.println("QuickSort: " + quickSortTime / 1000);
39 System.out.println("MergeSort: " + mergeSortTime / 1000);
40 System.out.println("NewSort: " + newSortTime / 1000);
41
```



```
1  System.out.println("\n1000 random integers");
2
3  for (int i = 0; i < 1000; i++) {
4      Integer[] table = new Integer[1000];
5
6      for (int j = 0; j < table.length; j++) {
7          table[j] = (int) (Math.random() * 100);
8      }
9      Integer[] table1 = table;
10     Integer[] table2 = table;
11
12     startTime = System.currentTimeMillis();
13     QuickSort.sort(table);
14     endTime = System.currentTimeMillis();
15     seconds = (endTime - startTime);
16     quickSortTime += seconds;
17
18     startTime = System.currentTimeMillis();
19     MergeSort.sort(table1);
20     endTime = System.currentTimeMillis();
21     seconds = (endTime - startTime);
22     mergeSortTime += seconds;
23
24     startTime = System.currentTimeMillis();
25     NewSort.sort(table2);
26     endTime = System.currentTimeMillis();
27     seconds = (endTime - startTime);
28     newSortTime += seconds;
29 }
30
31 System.out.println("QuickSort: " + quickSortTime / 1000);
32 System.out.println("MergeSort: " + mergeSortTime / 1000);
33 System.out.println("NewSort: " + newSortTime / 1000);
```



```
1  System.out.println("\n10000 random integers");
2
3  for (int i = 0; i < 1000; i++) {
4      Integer[] table = new Integer[10000];
5
6      for (int j = 0; j < table.length; j++) {
7          table[j] = (int) (Math.random() * 100);
8      }
9      Integer[] table1 = table;
10     Integer[] table2 = table;
11
12     startTime = System.currentTimeMillis();
13     QuickSort.sort(table);
14     endTime = System.currentTimeMillis();
15     seconds = (endTime - startTime);
16     quickSortTime += seconds;
17
18     startTime = System.currentTimeMillis();
19     MergeSort.sort(table1);
20     endTime = System.currentTimeMillis();
21     seconds = (endTime - startTime);
22     mergeSortTime += seconds;
23
24     startTime = System.currentTimeMillis();
25     NewSort.sort(table2);
26     endTime = System.currentTimeMillis();
27     seconds = (endTime - startTime);
28     newSortTime += seconds;
29 }
30
31 System.out.println("QuickSort: " + quickSortTime / 1000);
32 System.out.println("MergeSort: " + mergeSortTime / 1000);
33 System.out.println("NewSort: " + newSortTime / 1000);
```

RUNNING COMMANDS AND RESULTS

```
100 random integers adding to map
ChainBSTHashMap Put Method: 14 milliseconds
1000 random integers adding to map
ChainBSTHashMap Put Method: 40 milliseconds
1000 random integers adding to map
ChainBSTHashMap Put Method: 327 milliseconds
```

```
ChainBSTHashMap Get Method
Lets add [1,1], [2,5], [123,3] to the map
[(1, 1) (2, 5) (123, 3) ]
Lets get the value of key 2
5
Lets get the value of key 123
3
Lets get the value of key 1
1
As you can see it works
```

```
ChainBSTHashMap Remove Method
Lets add [1,1], [2,5], [123,3] to the map
[(1, 1) (2, 5) (123, 3) ]
Lets remove the entry of key 2
[(1, 1) (123, 3) ]
Lets remove the entry of key 123
[(1, 1) ]
Lets remove the entry of key 1
[]
As you can see it works
```

TESTING SORTING ALGORITHMS

100 random integers

QuickSort: 0.028

MergeSort: 0.032

NewSort: 0.022

1000 random integers

QuickSort: 0.05

MergeSort: 0.113

NewSort: 1.598

10000 random integers

QuickSort: 1.255

MergeSort: 0.771

NewSort: 141.018