GTU Department of Computer Engineering

CSE 222/505 - Spring 2022

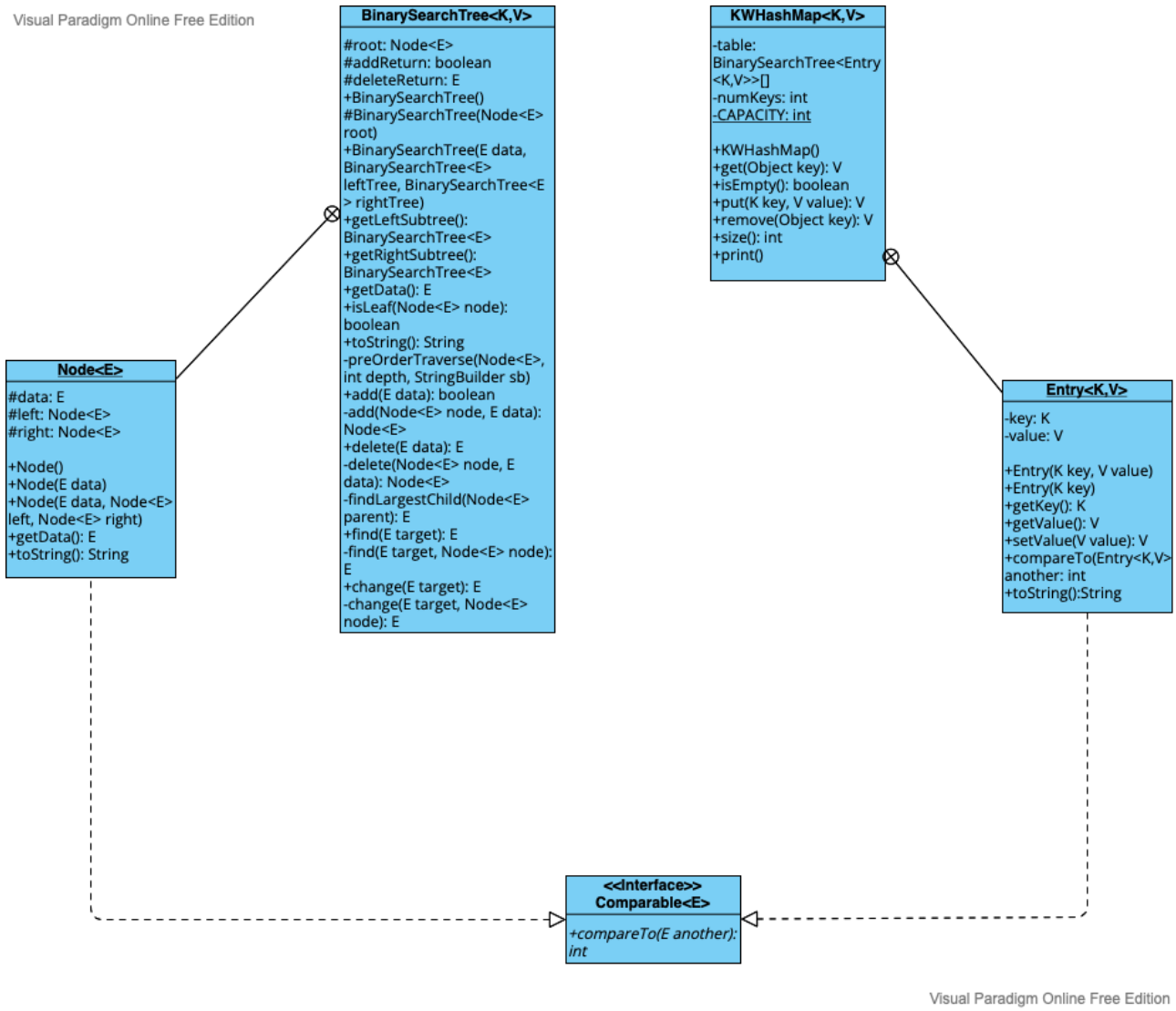
Homework 6 Report

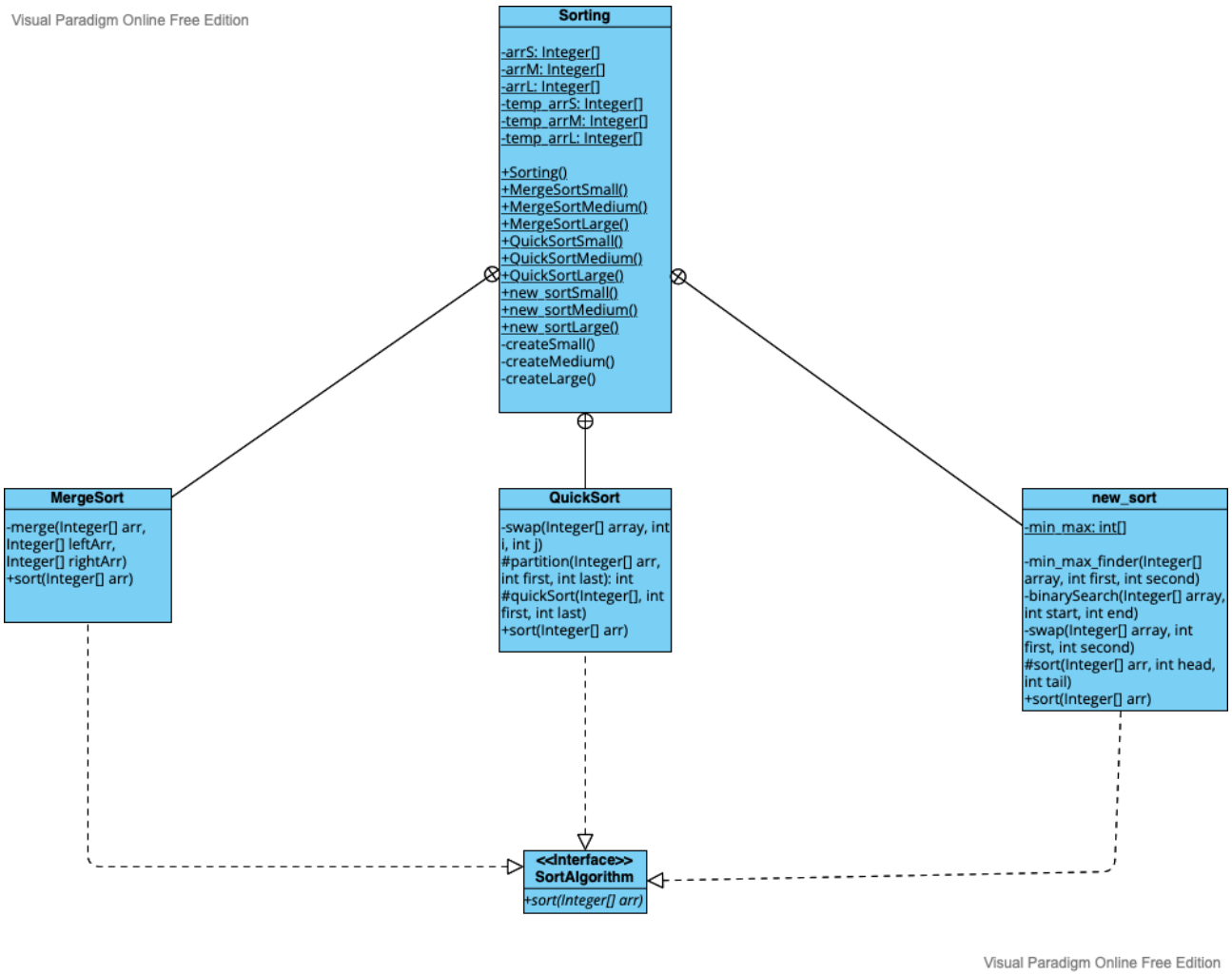
Sena Özbelen
1901042601

1. System Requirements

- KWHashMap should be initialized with right parameters and generics.
- Sorting class constructor should be called to create arrays.
- Sorting public methods can be called individually since they are static methods.

2. Class Diagram





3. Problem Solution Approach

For KWHashMap, we are asked to implement hash map using chaining and BST. I implemented using BST. For hash map methods, I used the BST methods instead of for loops since it is not possible to use regular loops for a BST.

For the second part of the question,

Coalesced hashing uses the concept of Open Addressing to find first empty place for colliding element from the bottom of the hash table and the concept of Separate Chaining to link the colliding elements to each other through pointers.

Double hashing uses the idea of applying a second hash function to key when a collision occurs. Double hashing can overcome the collision problem but it has larger intervals so this might cause trashing. Also, it is harder to implement.

For the sorting problem, in merge sort, there are two methods to sort the array. Sort method creates two distinct arrays by dividing the main array into two and it calls itself with new arrays recursively. After the very last division, it starts to merge two arrays by comparing.

In quick sort, there are four methods and firstly, partition method is called to divide the main array regarding the pivot which is the first element. By comparing the entries to the value of pivot, the entries is sorted.
In new_sort, we are asked to find min and max values so that they can be swapped. I modified the binary search method. It runs as it traverses the array so that we can find both max and min values at the same time by comparing. The method stores the values in a static integer array.

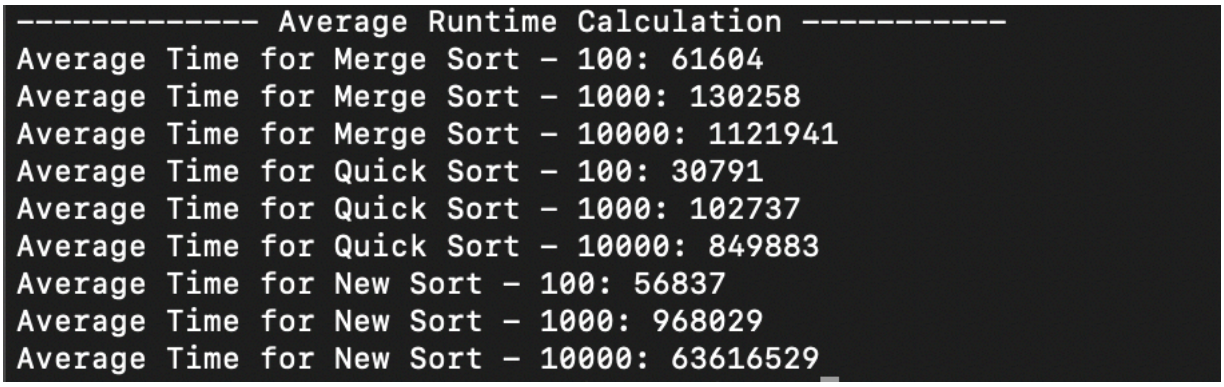
4. Test Cases

For the KWHashMap part,
-Add 3,7
-Add 12,4
-Add 13,5
-Add 25,8
-Add 23,3
-Add 51,6
-Delete 25

For the sorting part,
-Create 100, 1000, 10000-item random integer arrays.
-Perform three sorting algorithm respectively
-Calculate the average for 10 time-running.

5. Running Command and Results

```
Before deleting the key 25
1.level Node data: Key: 3 Value: 7

1.level Node data: Key: 12 Value: 4

1.level Node data: Key: 13 Value: 5

1.level Node data: Key: 23 Value: 3

1.level Node data: Key: 25 Value: 8

1.level Node data: Key: 51 Value: 6

After deleting the key 25
1.level Node data: Key: 3 Value: 7

1.level Node data: Key: 12 Value: 4

1.level Node data: Key: 13 Value: 5

1.level Node data: Key: 23 Value: 3


1.level Node data: Key: 51 Value: 6
```



For a random created sorting problem,
Merge Sort and Quick Sort are performed n*logn time but in this case, quick sort is more efficient than merge sort.
New Sort uses a modified version of binary search. Binary search takes logn time.
Therefore,
$T(n) = 2T(n/2) + c = O(\log n)$ for modified binary search
$T(n) = T(n-2) + O(\log n) = O(2^n)$
We can see the sudden increase between 100,1000 and 10000-item arrays. Therefore, this method is not efficient for high number of entries.