GTU Department of Computer Engineering

CSE 222/505 - Spring 2022

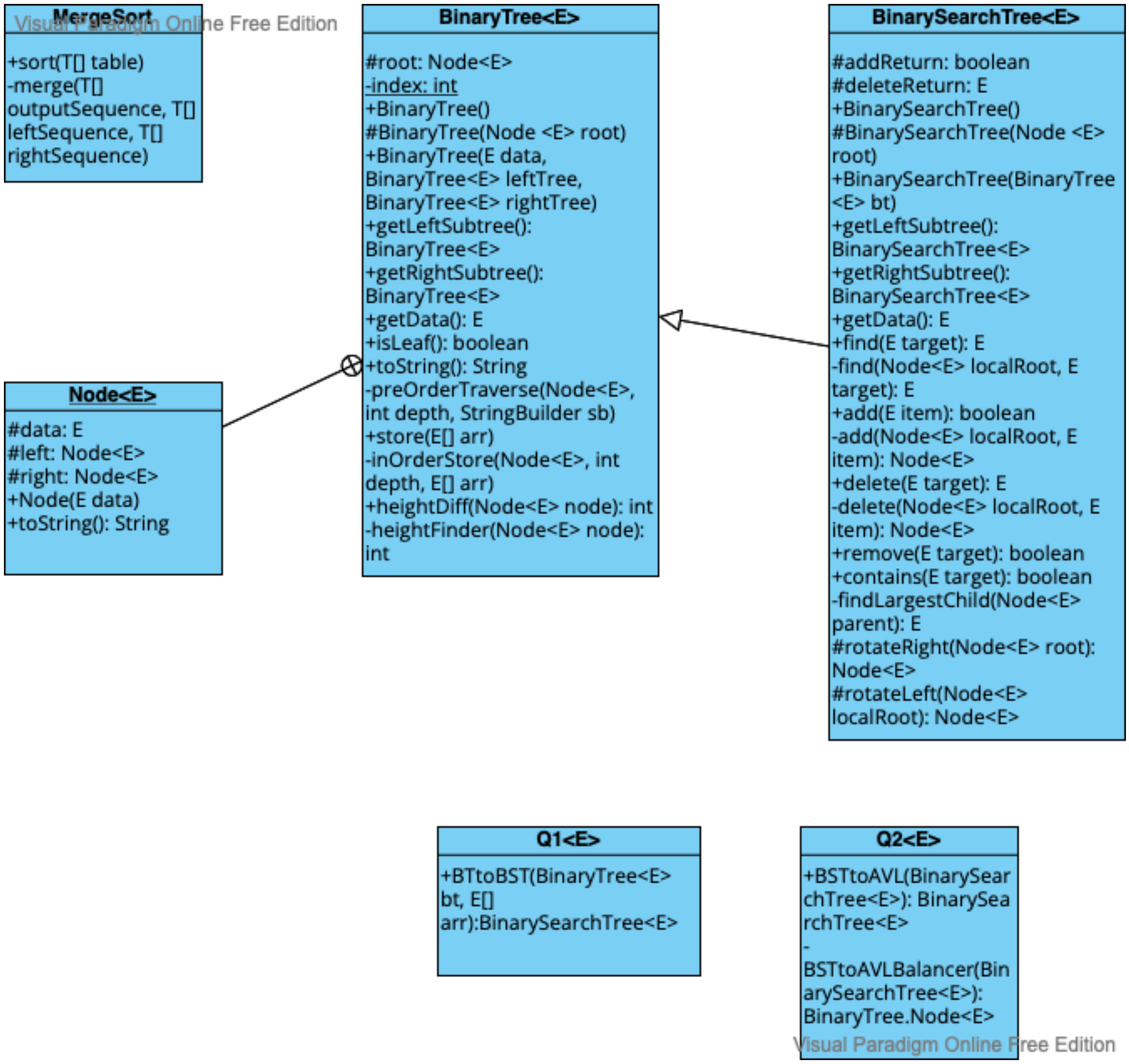
Homework 7 Report

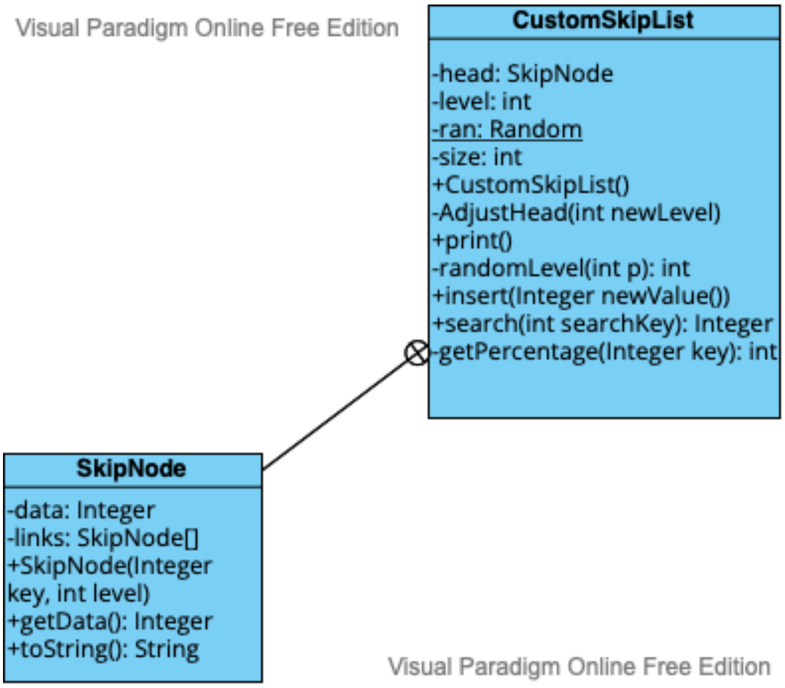
Sena Özbelen
1901042601

1. System Requirements

- The method “BTtoBST” should be called with right parameters. It should be given a binary tree and an array.
- The method “BSTtoAVL” should be called with right parameters. It should be given a binary search tree.

2. Class Diagram


MergeSort
+sort(T[] table)
-merge(T[] outputSequence, T[] leftSequence, T[] rightSequence)
BinaryTree<E>
#root: Node<E>
-index: int
+BinaryTree()
#BinaryTree(Node <E> root)
+BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)
+getLeftSubtree(): BinaryTree<E>
+getRightSubtree(): BinaryTree<E>
+getData(): E
+isLeaf(): boolean
+toString(): String
-preOrderTraverse(Node<E>, int depth, StringBuilder sb)
+store(E[] arr)
-inOrderStore(Node<E>, int depth, E[] arr)
+heightDiff(Node<E> node): int
-heightFinder(Node<E> node): int
BinarySearchTree<E>
#addReturn: boolean
#deleteReturn: E
+BinarySearchTree()
#BinarySearchTree(Node <E> root)
+BinarySearchTree(BinaryTree <E> bt)
+getLeftSubtree(): BinarySearchTree<E>
+getRightSubtree(): BinarySearchTree<E>
+getData(): E
+find(E target): E
-find(Node<E> localRoot, E target): E
+add(E item): boolean
-add(Node<E> localRoot, E item): Node<E>
+delete(E target): E
-delete(Node<E> localRoot, E item): Node<E>
+remove(E target): boolean
+contains(E target): boolean
-findLargestChild(Node<E> parent): E
#rotateRight(Node<E> root): Node<E>
#rotateLeft(Node<E> localRoot): Node<E>
Node<E>
#data: E
#left: Node<E>
#right: Node<E>
+Node(E data)
+toString(): String
Q1<E>
+BTtoBST(BinaryTree<E> bt, E[] arr):BinarySearchTree<E>
Q2<E>
+BSTtoAVL(BinarySearchTree<E>): BinarySearchTree<E>
-BSTtoAVLBalancer(BinarySearchTree<E>): BinaryTree.Node<E>


Visual Paradigm Online Free Edition
CustomSkipList
-head: SkipNode
-level: int
-ran: Random
-size: int
+CustomSkipList()
-AdjustHead(int newLevel)
+print()
-randomLevel(int p): int
+insert(Integer newValue())
+search(int searchKey): Integer
-getPercentage(Integer key): int
SkipNode
-data: Integer
-links: SkipNode[]
+SkipNode(Integer key, int level)
+getData(): Integer
+toString(): String
Visual Paradigm Online Free Edition

3. Problem Solution Approach

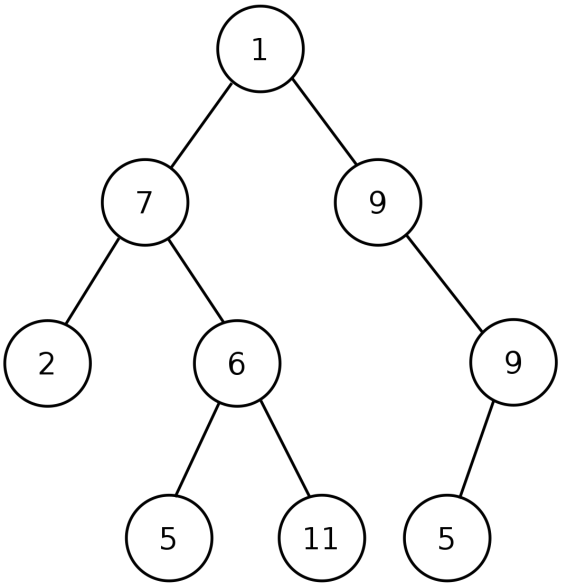
For the first question, I used merge sort to sort the given array first. I implemented a method called “store”. This method calls “inOrderStore”. inOrderStore traverses the given binary tree and assign the items in the array so that it creates a binary search tree. And this tree is returned.

For the second question, I implemented a recursive method. It reaches the very last nodes and checks if it’s balanced, otherwise it rotates the current subtree so that all subtrees become a balanced tree like AVL.

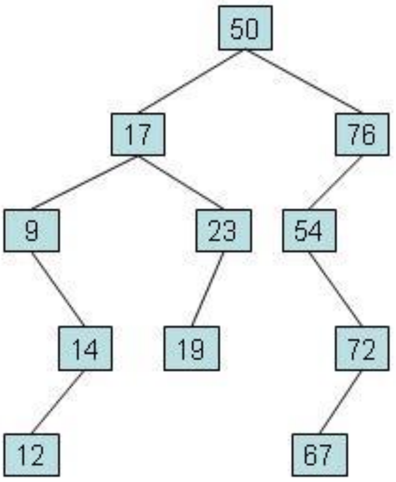
For the third question, I implemented skip list. Skip list has 2 level as default and after adding 10 elements, the program adds a new level to the head. The newly inserted items’ level probability is calculated in getPercentage method. It counts the distance between the new item and the tall items.

4. Test Cases

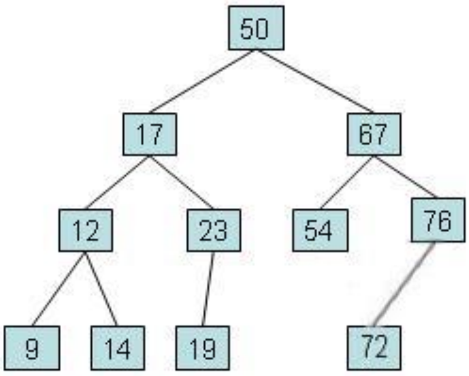
For the first question,
- Create an array with {2,5,1,8,3,11,6,14,4}
- Create a binary tree (see the picture)
- Print the structure of the given binary tree and the expected result



For the second question,
- Create a binary search tree (See the picture)
- Print the unbalanced BST and the expected result



An unbalanced tree

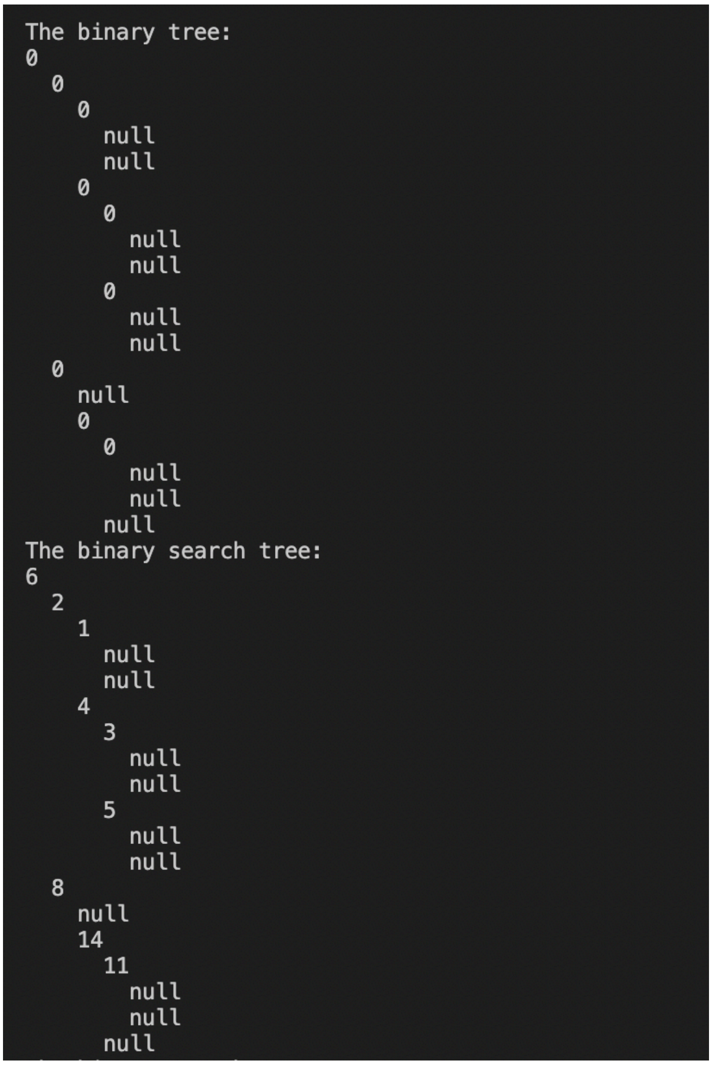


The same tree after
being height-balanced

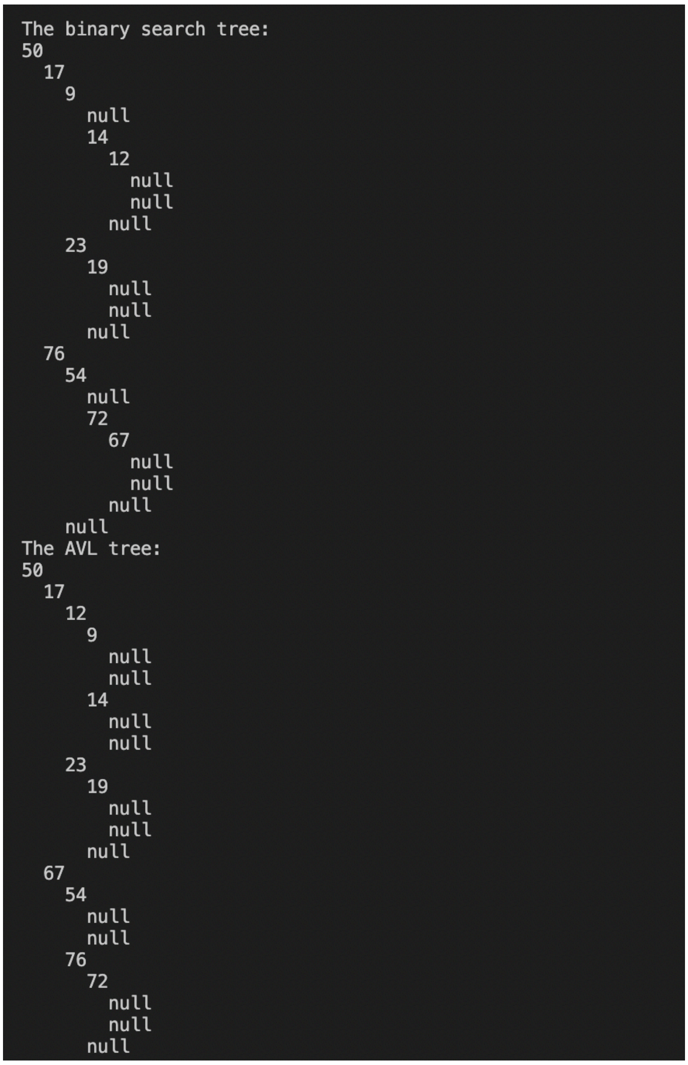
For the third question,

```
//Add first 11 items to see the raise

        sl.insert(50);
        sl.insert(17);
        sl.insert(76);
        sl.insert(9);
        sl.insert(23);
        sl.insert(19);
        sl.insert(14);
        sl.insert(12);
        sl.insert(54);
        sl.insert(72);
        sl.insert(67);

//Add other 10 items to see the raise

        sl.insert(100);
        sl.insert(90);
        sl.insert(70);
        sl.insert(5);
        sl.insert(7);
        sl.insert(88);
        sl.insert(91);
        sl.insert(120);
        sl.insert(45);
        sl.insert(76);
```

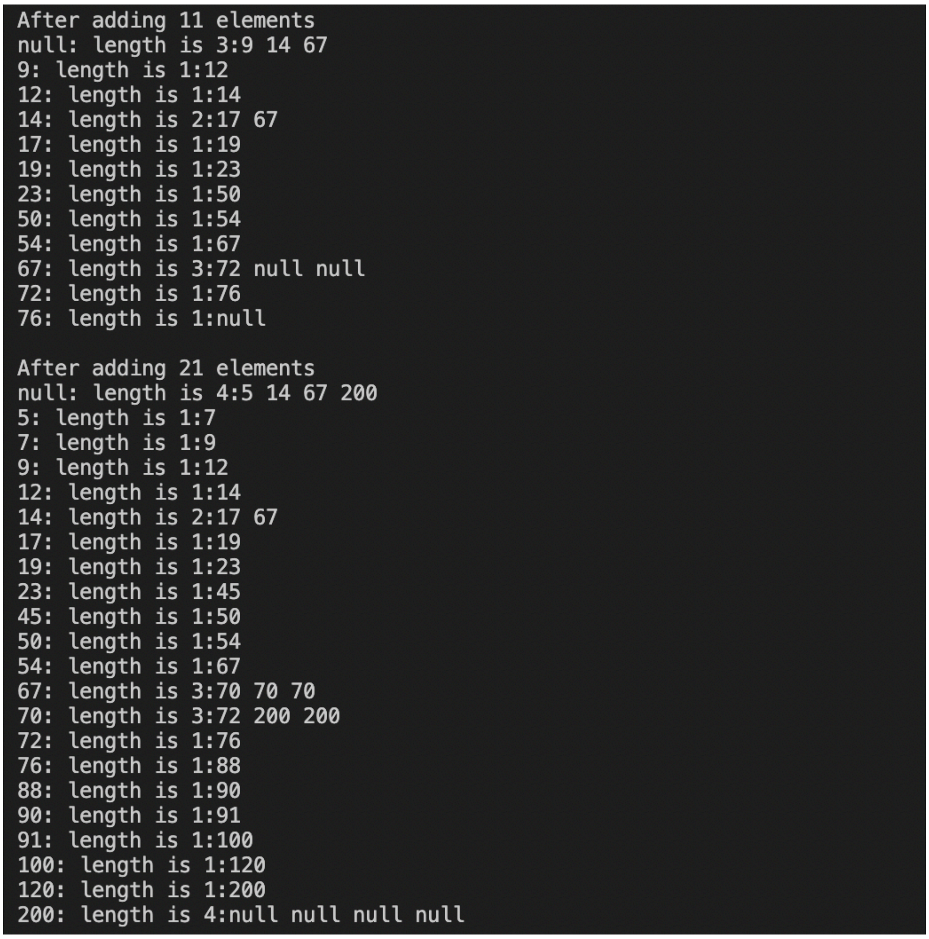
5. Running Command and Results



```
The binary tree:
0
  0
    0
      null
      null
    0
      0
        null
        null
      0
        null
        null
  0
    null
    0
      0
        null
        null
      null
The binary search tree:
6
  2
    1
      null
      null
    4
      3
        null
        null
      5
        null
        null
  8
    null
    14
      11
        null
        null
      null
```



```
The binary search tree:
50
  17
    9
      null
      14
        12
          null
          null
        null
    23
      19
        null
        null
      null
  76
    54
      null
      72
        67
          null
          null
        null
    null
The AVL tree:
50
  17
    12
      9
        null
        null
      14
        null
        null
    23
      19
        null
        null
      null
  67
    54
      null
      null
    76
      72
        null
        null
      null
```



```
After adding 11 elements
null: length is 3:9 14 67
9: length is 1:12
12: length is 1:14
14: length is 2:17 67
17: length is 1:19
19: length is 1:23
23: length is 1:50
50: length is 1:54
54: length is 1:67
67: length is 3:72 null null
72: length is 1:76
76: length is 1:null

After adding 21 elements
null: length is 4:5 14 67 200
5: length is 1:7
7: length is 1:9
9: length is 1:12
12: length is 1:14
14: length is 2:17 67
17: length is 1:19
19: length is 1:23
23: length is 1:45
45: length is 1:50
50: length is 1:54
54: length is 1:67
67: length is 3:70 70 70
70: length is 3:72 200 200
72: length is 1:76
76: length is 1:88
88: length is 1:90
90: length is 1:91
91: length is 1:100
100: length is 1:120
120: length is 1:200
200: length is 4:null null null null
```

Time Complexity

First question:

Merge Sort: O(nlogn)
Store: T(n) = 2T(n/2) + c , T(1) = 1
T(n) = n = O(n)
Total: n + nlogn = nlogn

Second question:

heightFinder: T(n) = 2T(n/2) + c , T(1) = 1
T(n) = n = O(n)
rotateRight, rotateLeft: O(1)