GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 5 Report

Emre YILMAZ
1901042606

1. SYSTEM REQUIREMENTS

In the first part of the assignment,

- We are asked to obtain some formulas such as *total depth*, *max/min node number*, *max/min leaf node number, max/min inner node number* for Complete Binary Tree
- We are asked to calculate the time complexity (best/worst case) while using Complete Binary Search Tree for searching.

In the second part of the assignment,

- We are asked to research about the quadtree structure for two-dimensional point data.
- Then, we are given a dataset to add them to the empty quadtree. We are asked to show the nodes during each insertion

In the third part of the assignment,

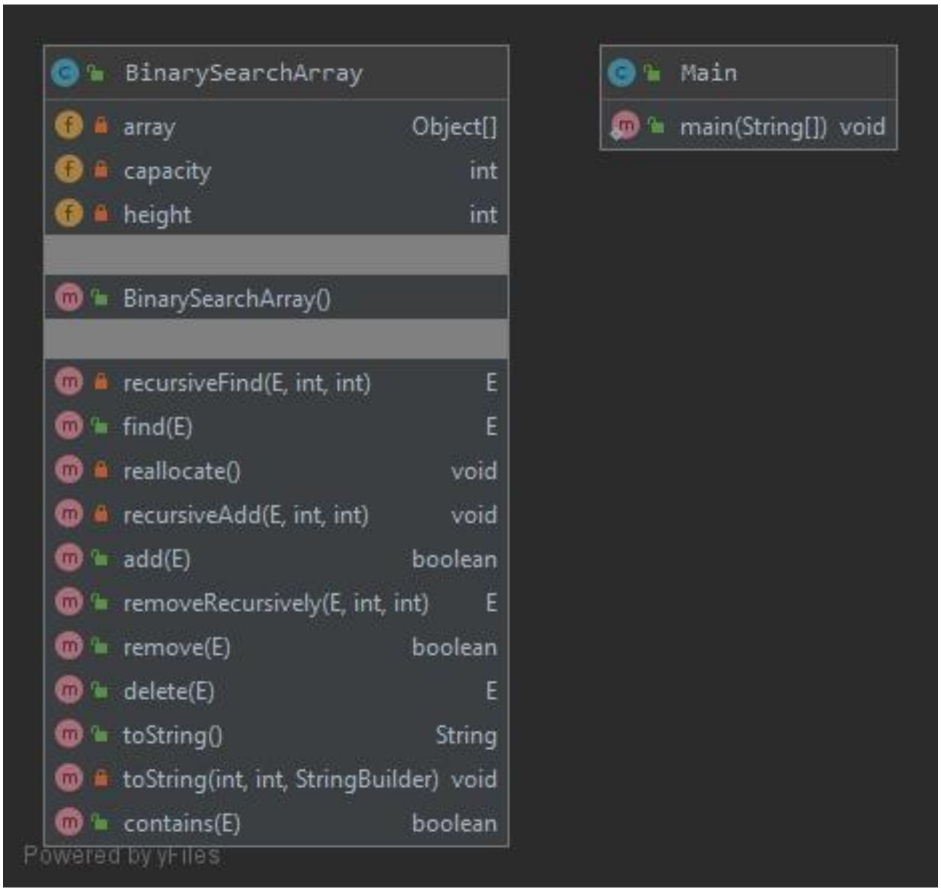
- We are asked to implement a *binary heap structure* that extends the *BinaryTree class* in the textbook.
- The unusual thing about this implementation is that it has to be implemented using *node linked structure* instead of *array structure*. Actually, the BinaryTree class in the textbook does not help at all for this implementation. The main thing here is the convert from array to node structure.

In the fourth part of the assignment,

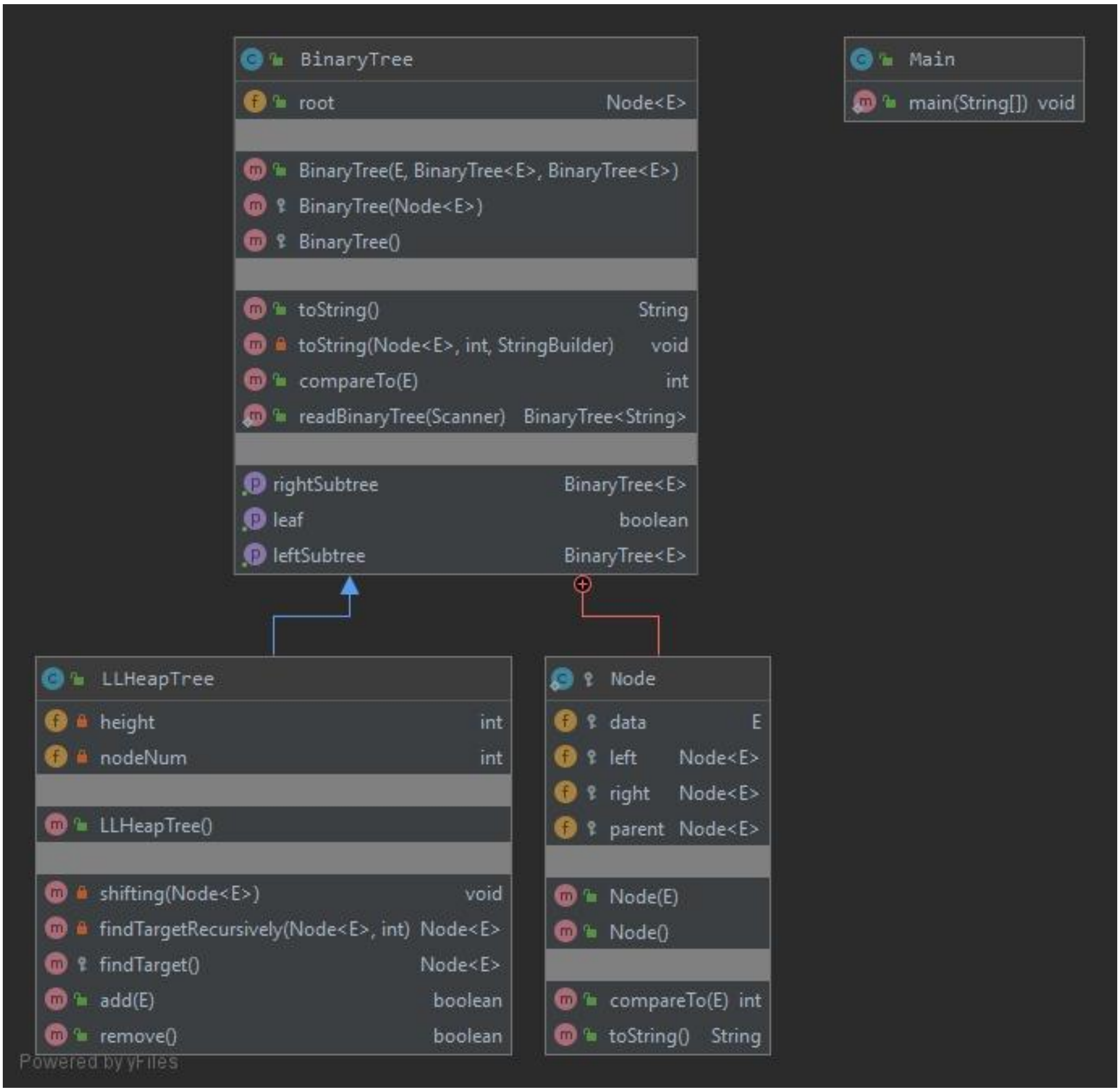
- We are asked to implement a *binary search tree* that implements *SearchTree interface* in the textbook.
- As in the previous question, we were asked to change the classic node linked structure implementation in this question. We are asked to use array instead of node linked structure for binary search tree

2. **CLASS DIAGRAMS**

Array Structured Binary Search Tree:



<u>Node Linked Structure Binary Heap Tree:</u>



3. PROBLEM SOLUTION APPROACH

- A and C part of the **first question** is solved by drawing various binary complete trees. Then, it is noted the necessary data of the tree such as *node number*, *inner node number*, *leaf number* etc. Then, using this data, trying to get some formulas that is asked.
- Then, using this data got from the trees, the formulas requested in the questions were tried to be got. Since the root node is in every tree, the formulas emerged simply when the root was subtracted from the data. Moreover, it was not difficult to get the formulas, as we think height focused.

- In the **Question-3:** firstly, the *BinaryTree class* and its methods in the textbook is written as a parent class. It is made some changes to make the *BinaryClass* more convenient for the Node Linked Structure. Generic type <E> is marked as *Comperable.* Also, an extra *Node<E>* data field '*parent'* is added to the *Node inner class*. It is going to make things easier when traversing/searching the tree. The other methods in the *BinaryTree class* are never used in the *Node Linked Structure* class.

- There are *add*, *remove*, *find* and *shifting* functions in the *Node Linked Structure* Heap B*inary tree* class. Main principle of all these functions except the *shifting* function is recursion.

- While *adding* a node to the tree, firstly, we must check whether it is in the tree. If it is not, we call the recursive add method. After that, the new node will be added to the end of the tree. Then, if it is smaller than its parent, it will be swapped with its parent. This process will be done until the swapped node is not smaller than its parent.

- While *removing* a node, it is going to be removed from the root. Then, the nodes remaining will be shifted towards the upper side. If there are 2 children of the node that will be shifted and if we are not at the deepest level we have to choose the one on the left. If we are at the deepest level, we have to choose the one on the right.

- In the **Question-4**: all the methods *add*, *contains*, *delete* and *remove* are implemented using recursion. In these functions, the data field *height* and *capacity* are used. When we are going to a new level in the tree, the capacity and height will be increased.
- In the recursions, the main thing is *2*position+1* is the left child, *2*position+2* is the right child of a node.
- The implementation of the *add* function was simple. We just had to find the correct node by using recursion.
- The problem in the implementation in this class is *remove* function. After removing a node from the array, the nodes under that parent have to be shifted towards upper side.

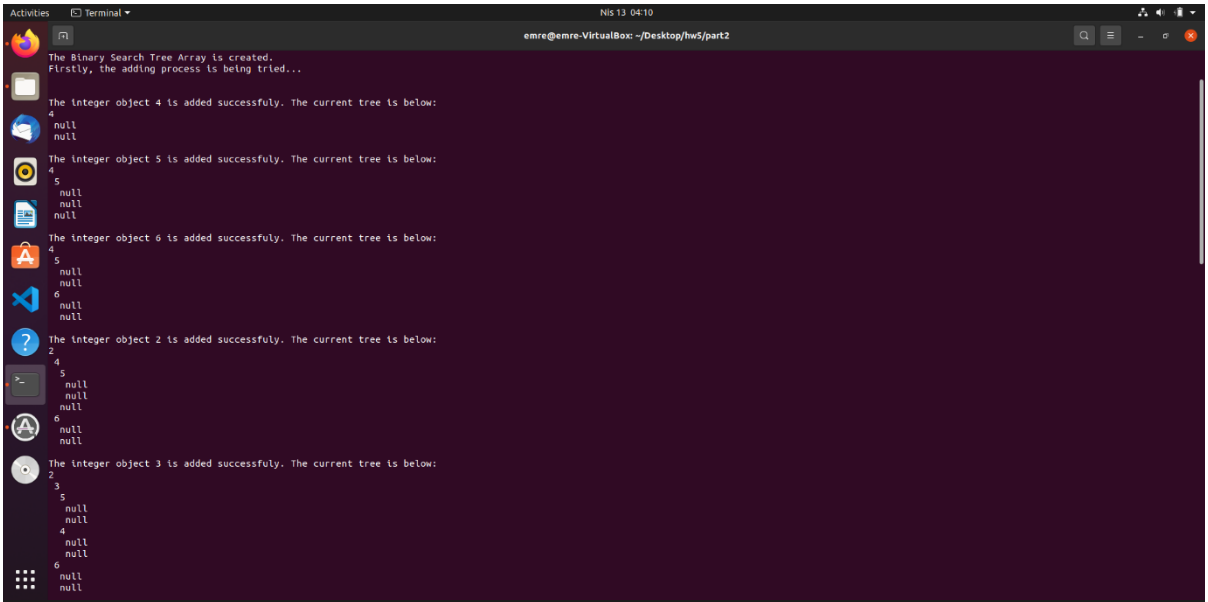
4. TEST CASES

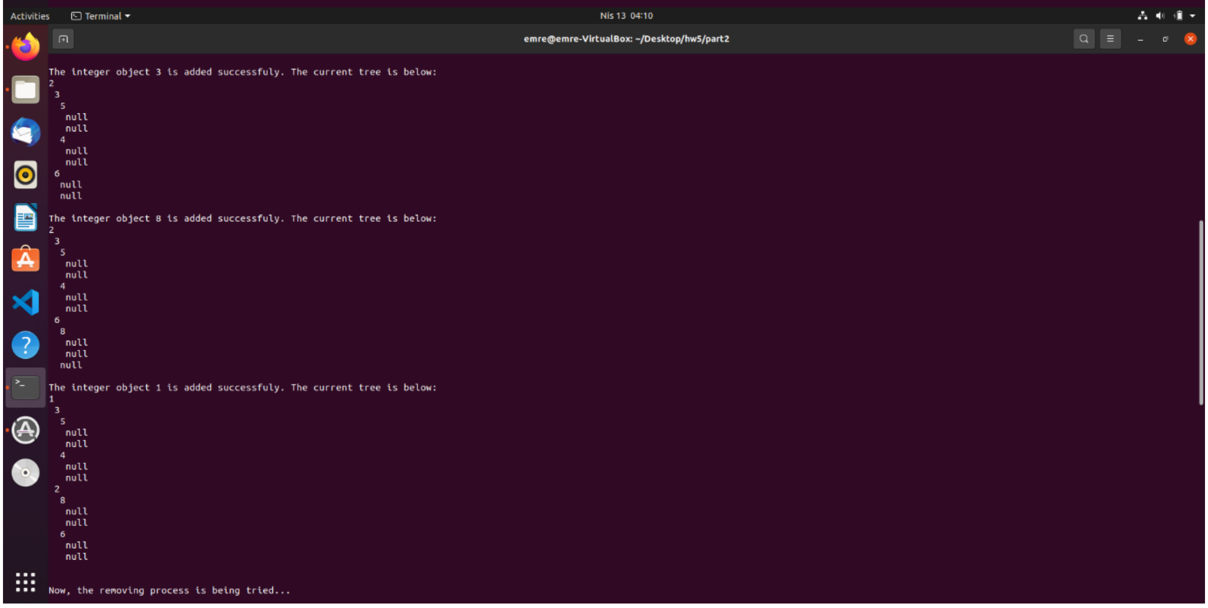
For binary search tree array implementation, additions to various nodes were tested. continuous right insertion or continuous left insertion were tested. Additions to the right and left of a specific node are tested. In the same way, removals from leaves or roots were made. Inner nodes were also removed.

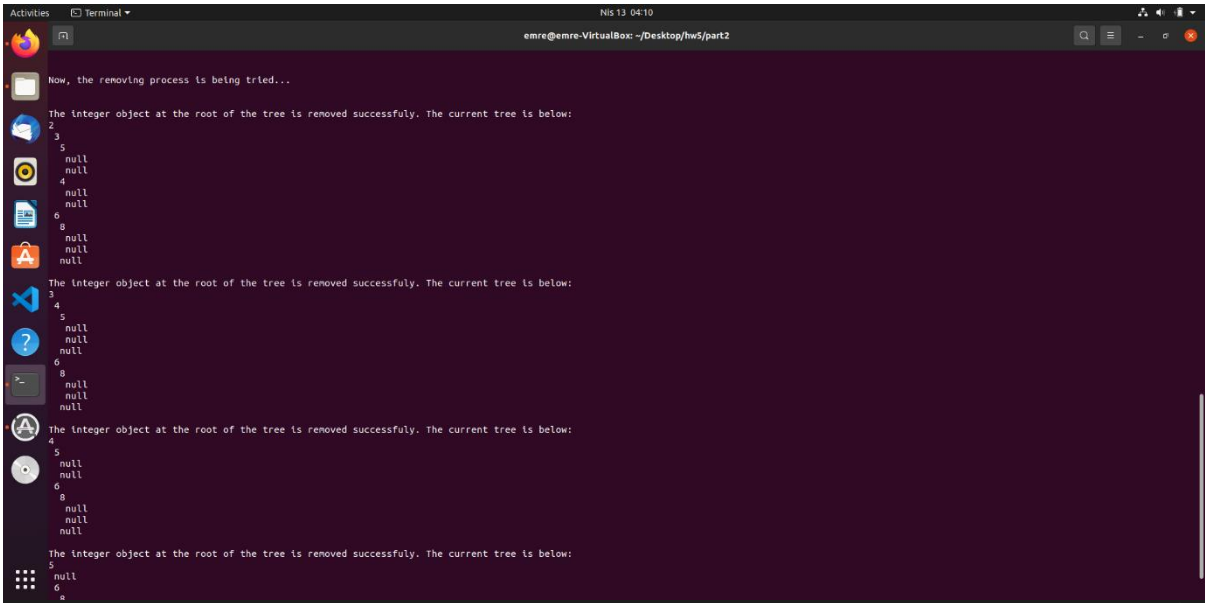
For heap implementation various cases have been tested in the heap. Remove operation from different nodes and adding operation to coincide with different nodes were tested.

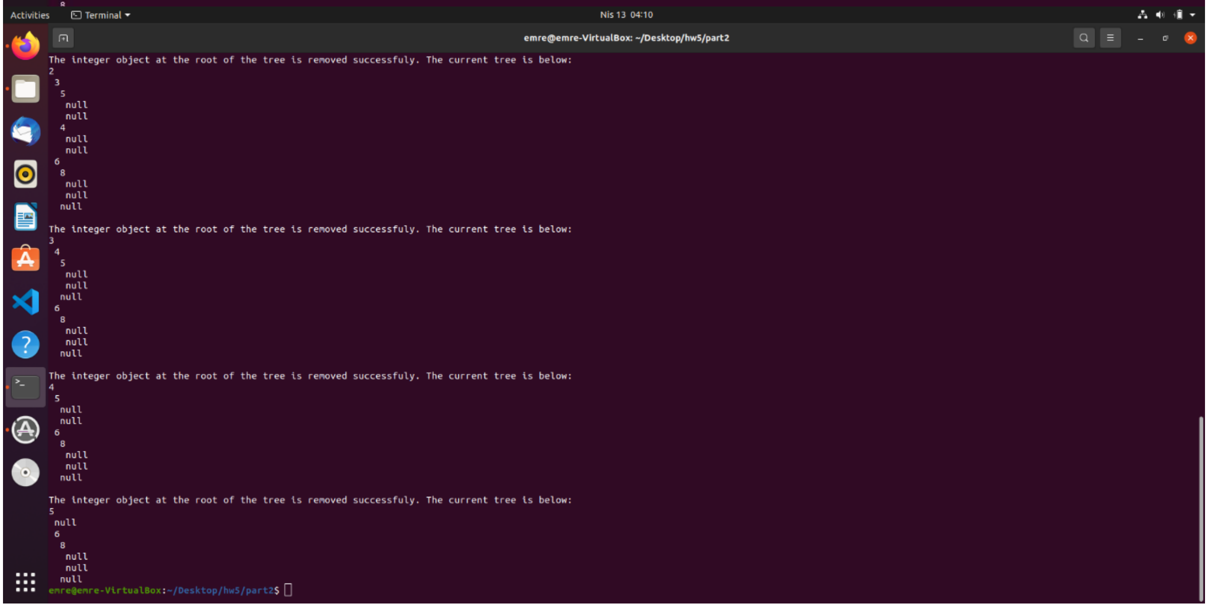
5. RUNNING AND RESULTS

Linked Node Heap Structure Test ->

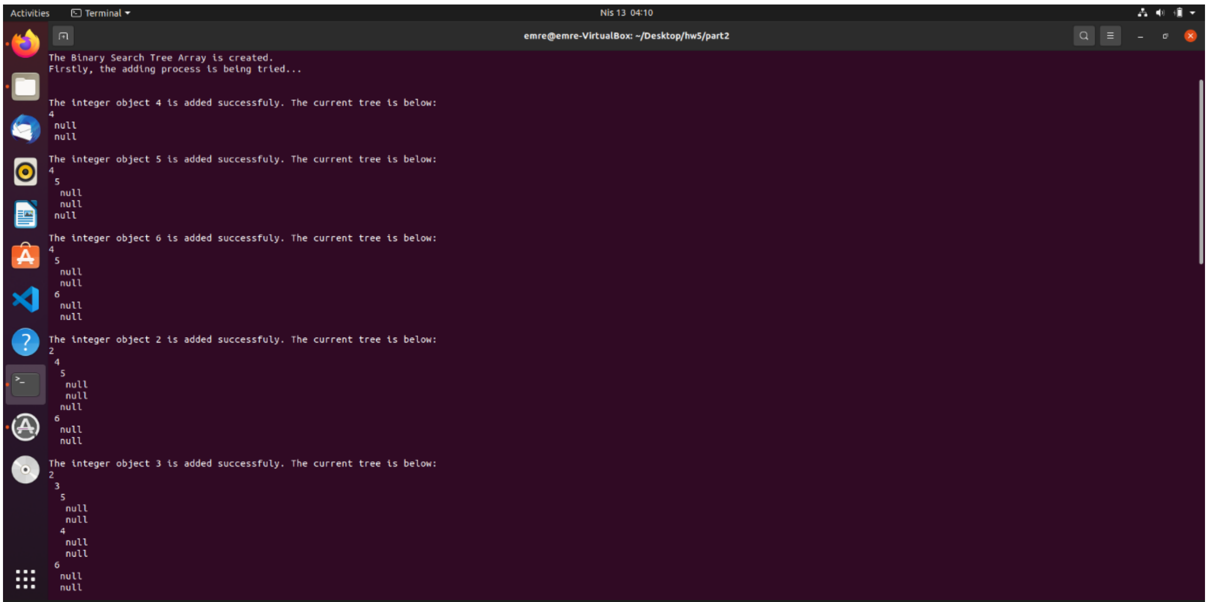


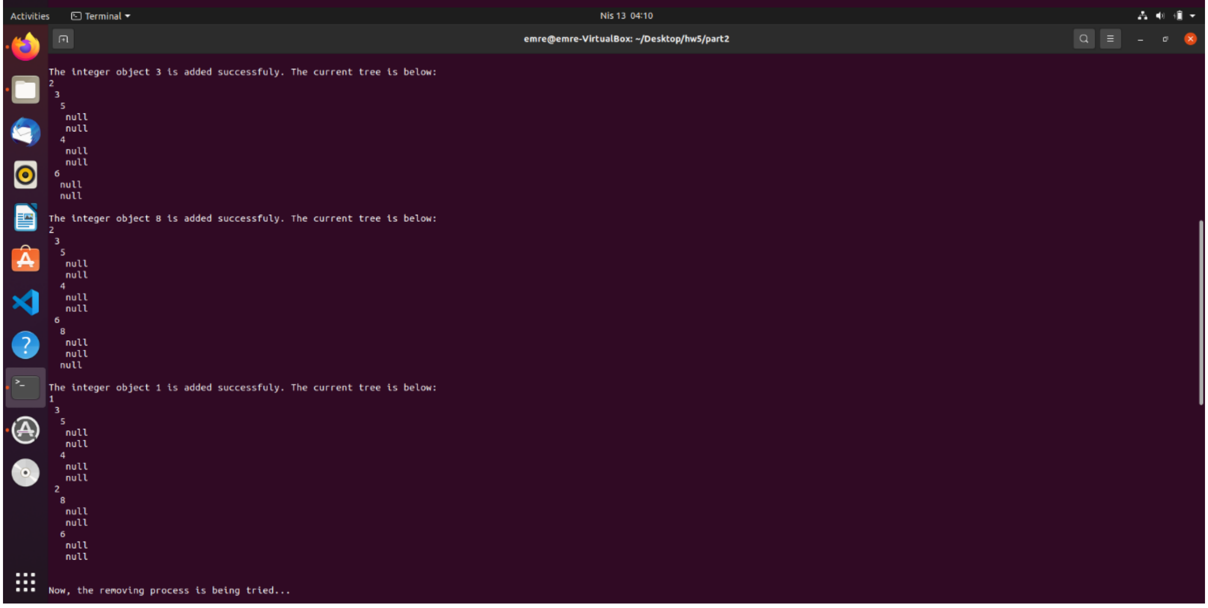


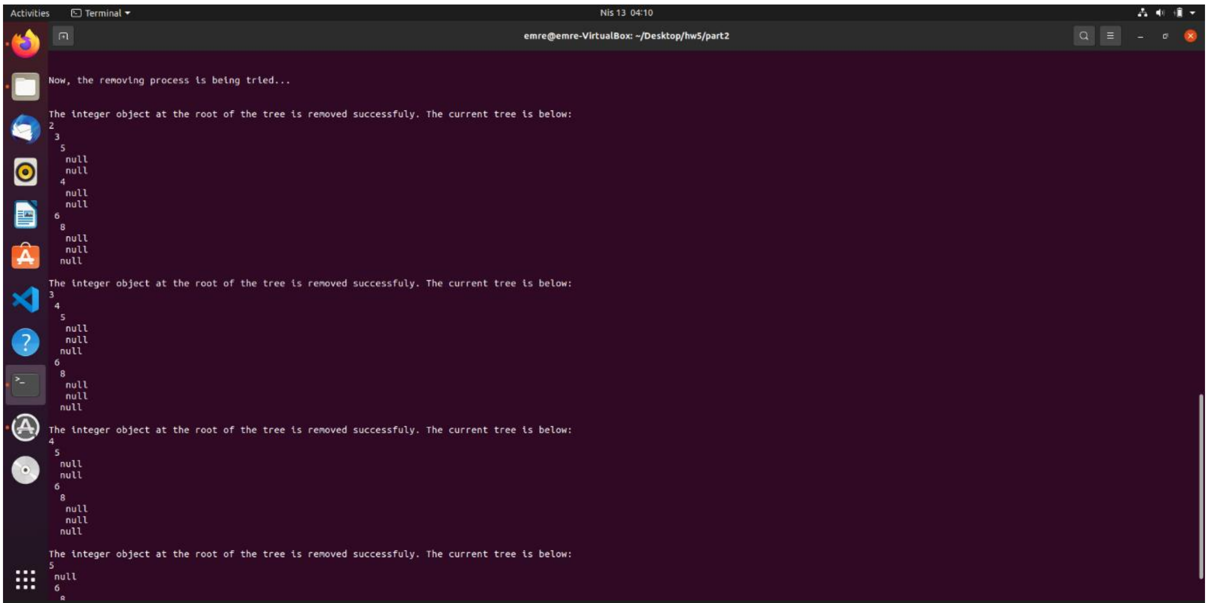

Activities
Terminal
Nis 13 04:10
emre@emre-VirtualBox: ~/Desktop/hw5/part2
Now, the removing process is being tried...
The integer object at the root of the tree is removed successfuly. The current tree is below:
2
 3
  5
   null
   null
  4
   null
   null
 6
  8
   null
   null
  null
The integer object at the root of the tree is removed successfuly. The current tree is below:
3
 4
  5
   null
   null
  null
 6
  8
   null
   null
  null
The integer object at the root of the tree is removed successfuly. The current tree is below:
4
 5
  null
  null
 6
  8
   null
   null
  null
The integer object at the root of the tree is removed successfuly. The current tree is below:
5
 null
 6

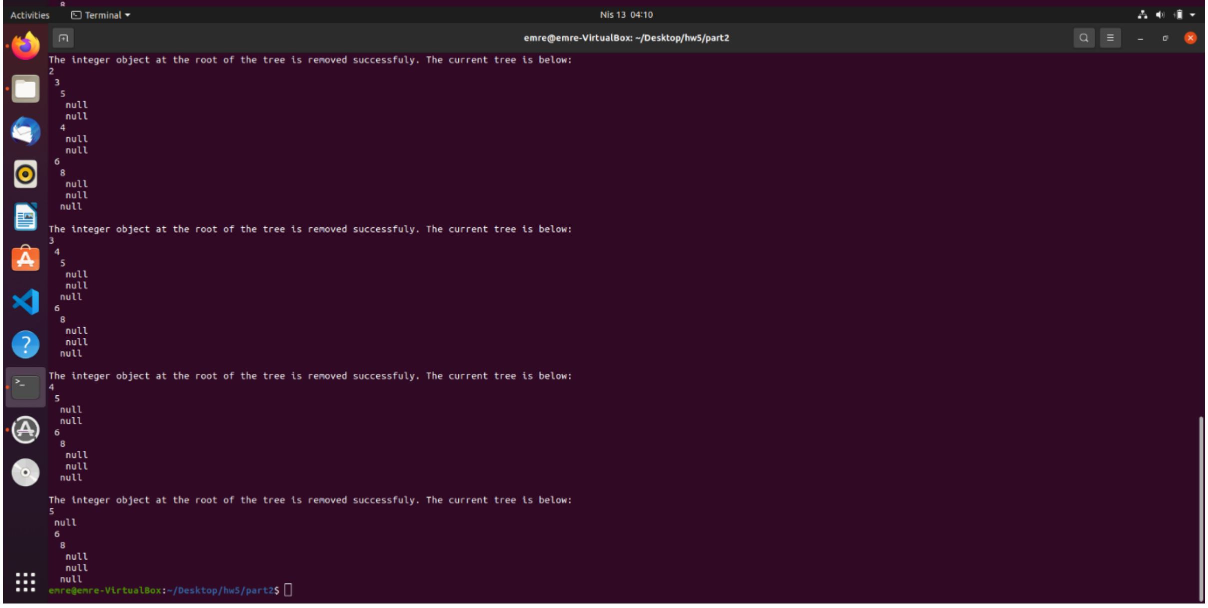

Activities
Terminal
Nis 13 04:10
emre@emre-VirtualBox: ~/Desktop/hw5/part2
The integer object at the root of the tree is removed successfuly. The current tree is below:
2
 3
  5
   null
   null
  4
   null
   null
 6
  8
   null
   null
  null
The integer object at the root of the tree is removed successfuly. The current tree is below:
3
 4
  5
   null
   null
  null
 6
  8
   null
   null
  null
The integer object at the root of the tree is removed successfuly. The current tree is below:
4
 5
  null
  null
 6
  8
   null
   null
  null
The integer object at the root of the tree is removed successfuly. The current tree is below:
5
 null
 6
  8
   null
   null
  null
emre@emre-VirtualBox:~/Desktop/hw5/part2$

Array Structure Binary Search Tree Test ->






Activities
Terminal
Nis 13 04:10
emre@emre-VirtualBox: ~/Desktop/hw5/part2
Now, the removing process is being tried...
The integer object at the root of the tree is removed successfuly. The current tree is below:
2
3
5
null
null
4
null
null
6
8
null
null
null
The integer object at the root of the tree is removed successfuly. The current tree is below:
3
4
5
null
null
null
6
8
null
null
null
The integer object at the root of the tree is removed successfuly. The current tree is below:
4
5
null
null
6
8
null
null
null
The integer object at the root of the tree is removed successfuly. The current tree is below:
5
null
6


Activities
Terminal
Nis 13 04:10
emre@emre-VirtualBox: ~/Desktop/hw5/part2
The integer object at the root of the tree is removed successfuly. The current tree is below:
2
3
5
null
null
4
null
null
6
8
null
null
null
The integer object at the root of the tree is removed successfuly. The current tree is below:
3
4
5
null
null
null
6
8
null
null
null
The integer object at the root of the tree is removed successfuly. The current tree is below:
4
5
null
null
6
8
null
null
null
The integer object at the root of the tree is removed successfuly. The current tree is below:
5
null
6
8
null
null
null
emre@emre-VirtualBox:~/Desktop/hw5/part2$

The analyse time complexity of the methods in the Array Structure Binary Search Tree:

- Contains(E object) = $O(\log n)$
- Find(E object) = $O(\log n)$
- Reallocate() = Amortised $\Theta(1)$
- Add(E object) = $O(\log n)$
- Remove(E object) = $O(\log^2 n)$
- Delete(E object) = $O(\log^2 n)$
- toString() = $\Theta(\log n)$

The analyse time complexity of the methods in the Node Linked Structure Heap:

- Shifting() = $O(\log n)$
- findTarget(E object) = $O(\log n)$
- add(E object) = $\Theta(1)$
- Remove() = $O(\log n)$

1-)

a-) * A complete binary tree is a binary tree in which all the levels are completely filled except possibly lowest one, which is filled left to right.

Maximum total depth in a complete binary tree =>

- With $h=2$, Total depth $=5$
- With $h=3$, Total depth $=17$
- With $h=4$, Total depth $=49$
- With $h=5$, Total depth $=129$

✓ → The formula obtained from data above $=(h-1).2^h+1$

Minimum total depth in a complete binary tree =>

- With $h=2$, Total depth $=3$
- With $h=3$, Total depth $=8$
- With $h=4$, Total depth $=21$
- With $h=5$, Total depth $=54$

→ The formula obtained from data above $= max(h-1)+h$

✓ (Minimum total depth at h height tree = Maximum total depth at $(h-1)$ height tree + h)

b-) The traversal in a binary search tree is the same as using a sorted list for binary search. We go by halving the tree.

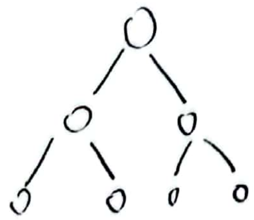
* Worst Case => The node that is being searched does not exist. We will go to the end. We will make $\log_2 n$ comparison. $=\Theta(\log n)$

* Best Case => We will get the node by accessing only the root. We will make 1 comparison $=\Theta(1)$

c-) * A complete binary tree is binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left to right.

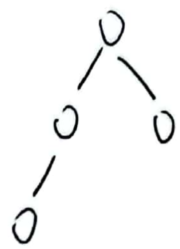
* Maximum node number $= 2^h - 1$
Minimum node number $= 2^{h-1}$
* Maximum leaf node number $= 2^{(h-1)}$
Minimum leaf node number $= 2^{(h-2)} - 1$, (if $h == 1$, $= 1$)
* Maximum inner node number $= 2^{(h-1)} - 1$
Minimum inner node number $= h$

Max Cases =)



(Example)

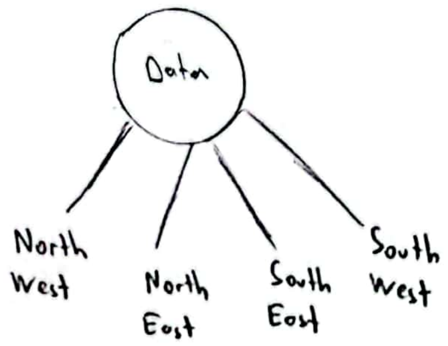
Min Cases =)



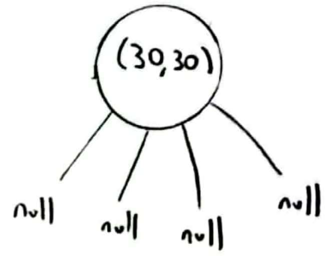
(Example)

2-)

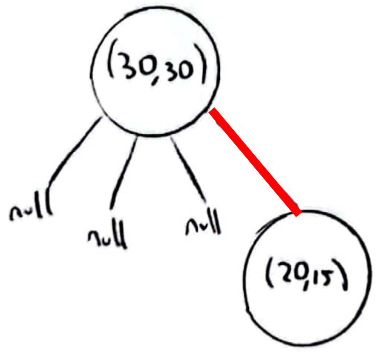
* The data is represented in the Cartesian coordinate plane. The tree is designed as below.



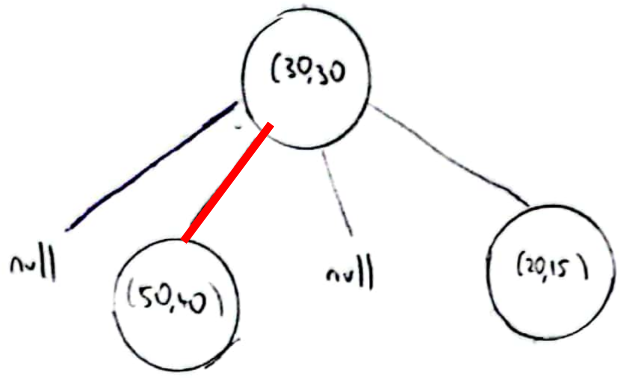
→ First Insertion



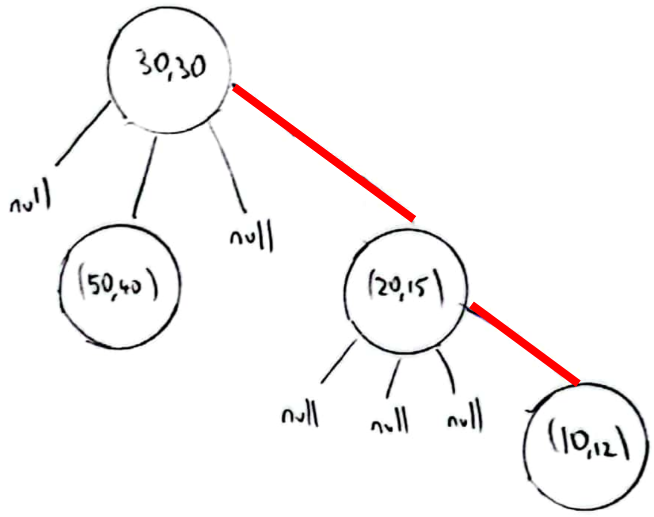
→ Second Insertion



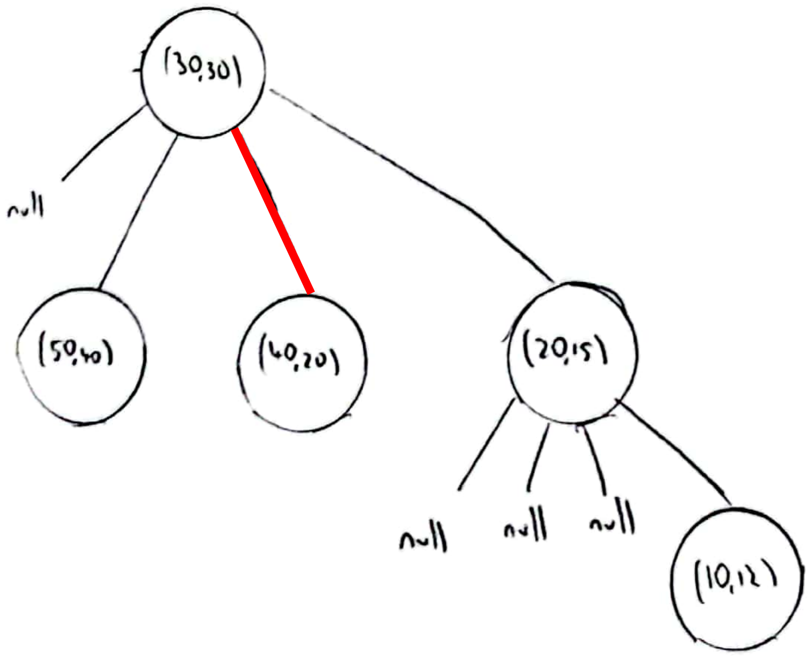
→ Third Insertion



→ Fourth Insertion

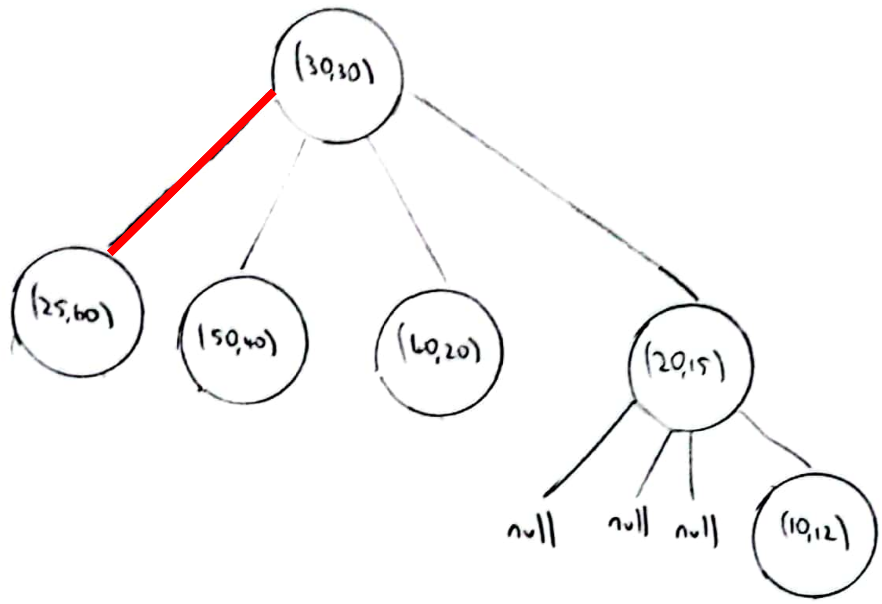


→ Fifth Insertion



Scanned with CamScanner

→ Sixth Insertion



→ The last Insertion