1)

a)

$$\sum_{h=1}^{\infty} \frac{(2^{h-1}).h + (2^{h-2}).(h-1)}{2}$$ $\Rightarrow$ it will give us total average depth of not in height of this h.

We can not calculate exact number of total depht because we can not know exact node of complete binary tree without traversal. -

b)

Average nuber of comparisen is equall to average number of nodes, becaus when traversal is continiuing on every node just complete one comparison.

So that

$$\sum_{h=1}^{\infty} \frac{2^{h-1}}{2}$$ this formula gives us.

c) There is no restriction for number of nodes. Because in full binary tree every level doesn't need to be full for going to next level, just every number of leaves of nodes are full or zero.

So if total number of nodes is N and I is internal nodes

formula $N = 2I + 1$

and number of leaves represent with L

$L = I + 1$

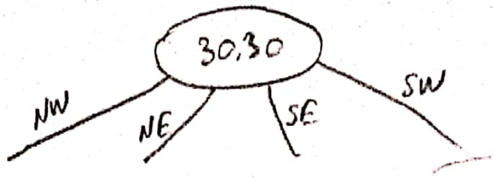
$\underline{\underline{N = 2L - 1}}$

$\underline{\underline{\frac{N+1}{2} = L}}$, $\underline{\underline{\frac{N-1}{2} = I}}$

2)

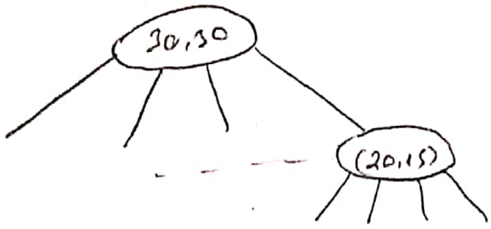
Every point adds to tree with its position in order to the point that is added before of it.

1



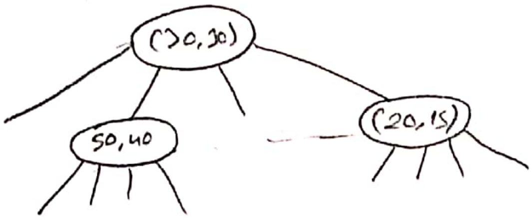
add (30,30)

2



add (20,15)

3

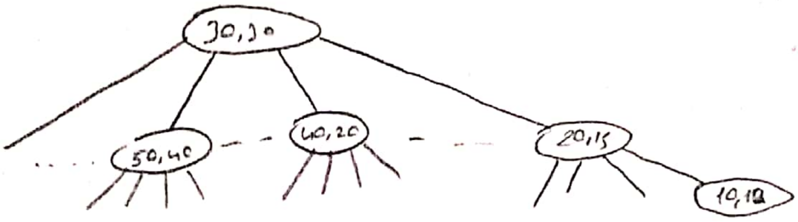


add(50,40)

4

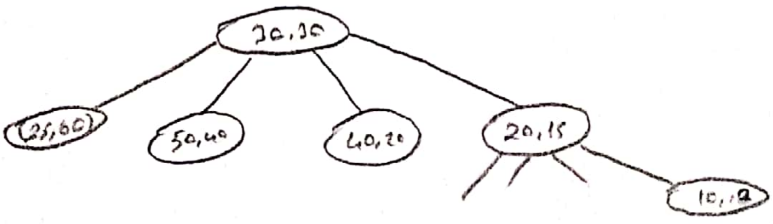
add(10,12)

5



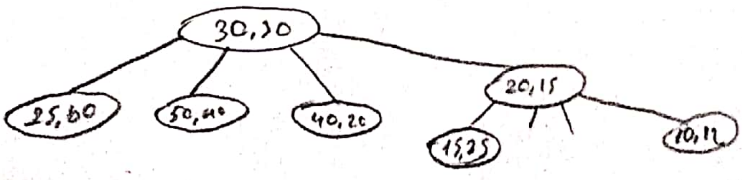
add(40,20)

6



add (25,60)

7



add(15,25)

Question-4

Problem Solution Approach

First I had to code an array class myself from scratch. Because even if all the elements of the array I will use are not full, I need to be able to increase its size. Then, unlike the normal binary search tree function, I had to make some changes to the delete method. Because in a normal tree node structure, when an element was deleted, it was enough to connect the address of the other element. But now, when I deleted an element, I had to reposition the other array elements according to it. So I wrote a helper function (Reordertree) to perform this function.

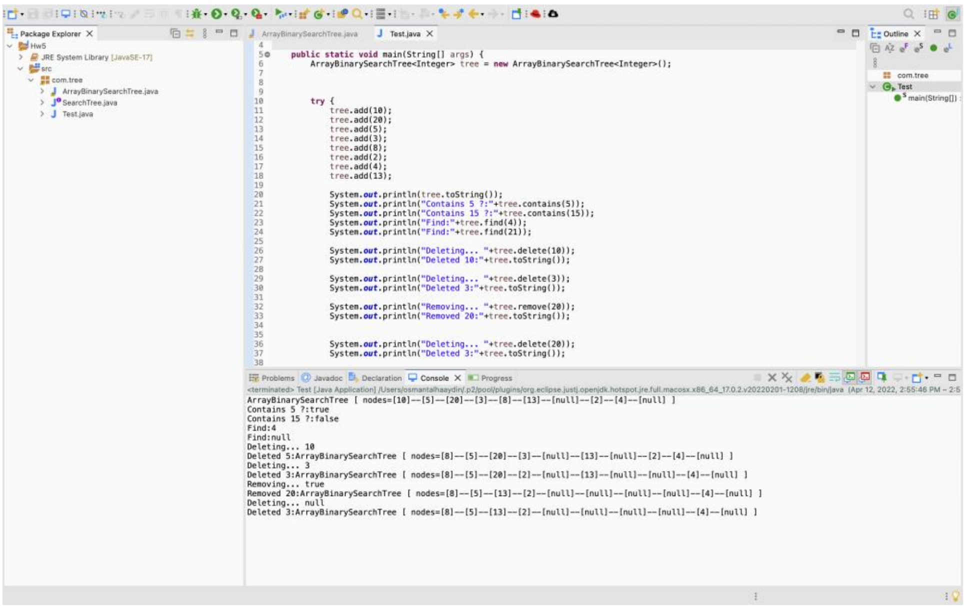
Detailed System Requirement

Requirement ID	Requirement
R01	Java jdk version is 18.

Test Cases

Test ID	Test
T01	Add item to the tree.
T02	Delete item that is in the root.
T03	Delete item that is in the leaf.
T04	Remove item that is in the leaf.
T05	Find item that is already in tree.
T06	Find item that is not already in tree.
T07	Control containing of item that is already in tree.
T08	Control containing of item that is not already in tree.

Running Command and Results



Analyze

- private void add(int index,E item)
    - Best Case:T(n) = Q(1) constant time
    - Worst Case:T(n) = Q(n)
    - Average Case: T(n) = Q(logn)
- Public boolean add(E item)
    - Best Case:T(n) = Q(1) constant time
    - Worst Case:T(n) = Q(n)
    - Average Case: T(n) = Q(logn)
- Private E find(int index,E target)
    - Best Case:T(n) = Q(1) constant time
    - Worst Case:T(n) = Q(n)
    - Average Case: T(n) = Q(logn)
- Public E find(E target)
    - Best Case:T(n) = Q(1) constant time
    - Worst Case:T(n) = Q(n)
    - Average Case: T(n) = Q(logn)
- Public boolean contains(E target)
    - Best Case:T(n) = Q(1) constant time
    - Worst Case:T(n) = Q(n)
    - Average Case: T(n) = Q(logn)
- Private void reorderofTree(int index)
    - Best Case:T(n) = Q(1) constant time
    - Worst Case:T(n) = Q(n^2)
    - Average Case: T(n) = Q(n)
- Private int findLargestChild(int index)
    - Best Case:T(n) = Q(1) constant time
    - Worst Case:T(n) = Q(n)
    - Average Case: T(n) = Q(logn)
- Private E delete(int index,E item)
    - Best Case:T(n) = Q(1) constant time
    - Worst Case:T(n) = Q(n^3)