CSE 222 DATA STRUCTURES AND ALGORITHMS

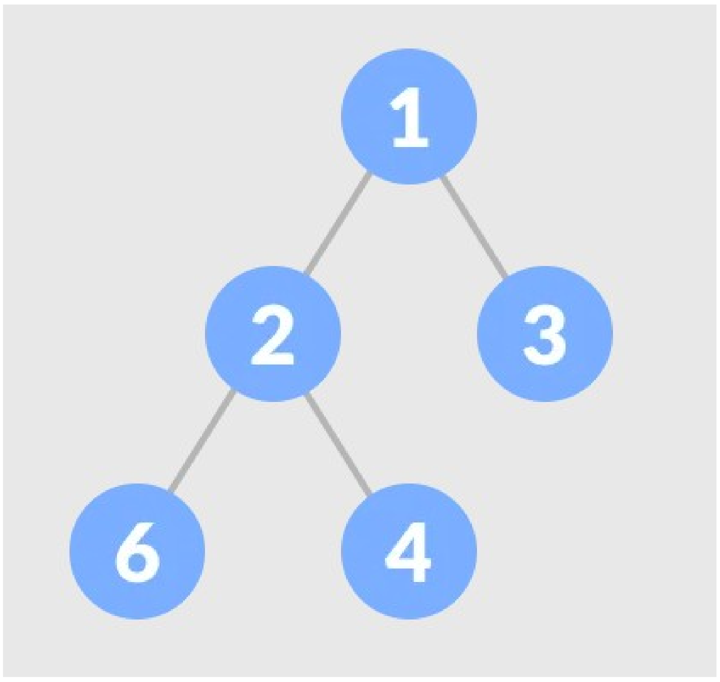
HOMEWORK 5 REPORT

MERVE HORUZ

1801042651

Q1:

a) shows that how to calculate the total depth of the nodes in a complete binary tree of height h. The depth of a node is the number of edges from the node to the tree's root node.
A root node will have a depth of 1. At each layer, multiply depth a node and number of nodes.



→ in node 1, depth is 1.
→ in node 2 and 3, the depth is 2.
→ in node 6 and 4, the depth is 3.
→ so, total depth is 1 + 3*2 + 2*2 = 11

code:

```
int n = 0,  totalDepth= 0;
for(int i = 1, i<=h;  i++, n++)
        totalDepth = totalDepth + Math.pow(2,n) * i
```

b) number of comparison = $\sum_{X \in \{A,\ldots,H\}} pX \cdot C(X)$

X is the variable for every element in complete tree, say C(X)
pX denotes the probability that X is chosen, which is the same for all X, (1/6 for this tree)

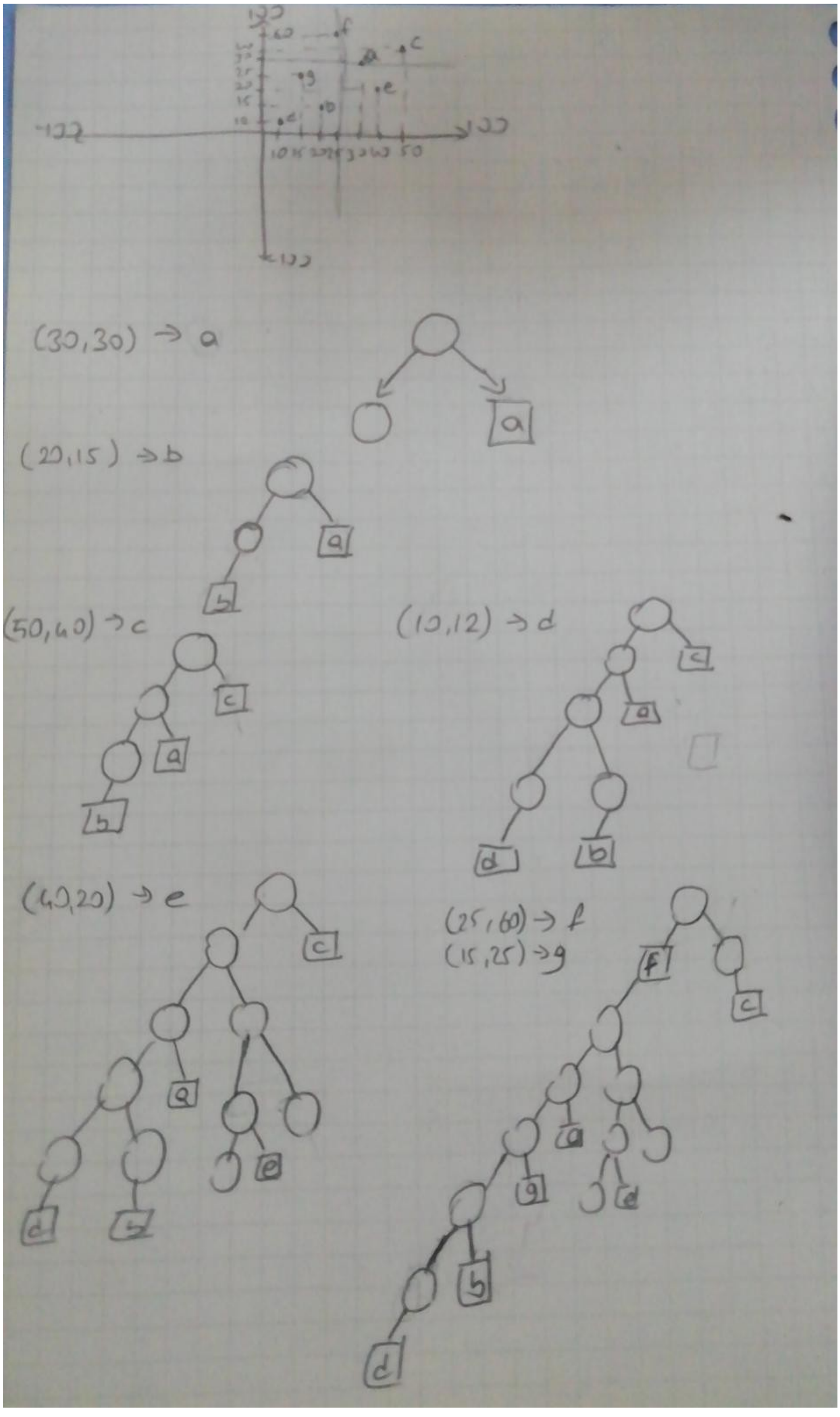
→ 1 node for level 1
→ 2 node for level 2
→ 2 node for level 3
→ number of comparison = 1*1 + 2*2 + 2*3
  = 11/5(number of nodes) = 2.2

c) A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children. There is a restriction in full binary tree. For example you can not draw a tree that has 6 node. (Root node has two children and these children has two nodes or not, so there is node number is 7 or 5 not 6.)

Number of leaves in a full binary tree is one more than the number of internal nodes.

In an n node full binary tree:
→ number of leaves: (n+1)/2
→ number of internal nodes: (n-1)/2

Q2:



1) Detailed system requirements

1.1) Non-functional requirements

openjdk 17.0.1 2021-10-19
OpenJDK Runtime Environment (build 17.0.1+12-Ubuntu-120.04)
OpenJDK 64-Bit Server VM (build 17.0.1+12-Ubuntu-120.04, mixed mode, sharing)

1.2) Functional requirements

→ public boolean add(int item): it takes item that will added as parameter.

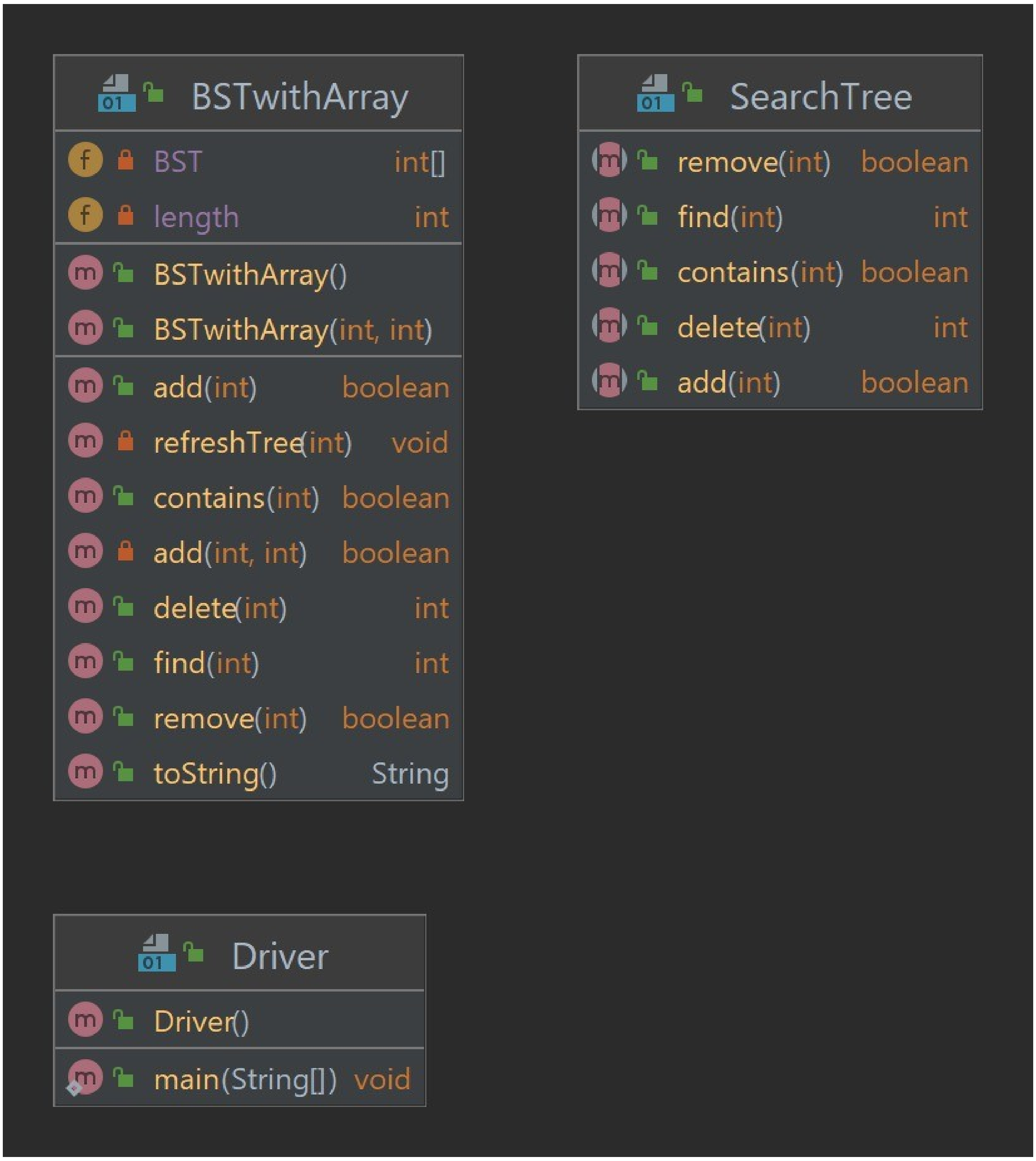
→ public boolean contains(int target): it takes target element to control whether elements is in tree or not. Target that will searched must be in tree, otherwise returns false.

→ public int find(int target): it takes an item as parameter to search. Target that will finded must be in tree otherwise returns -1.

→ public int delete(int target): it takes an item as parameter to delete. Target that will deleted must be in tree, otherwise throws exception.

→ public boolean remove(int target): it takes an item as parameter to remove from tree. Target that will removed must be in tree, otherwise returns false.

2) Class diagram



3) Problem solution approach

The problem is implementing an array based binary search tree class by implementing SearchTree interface.

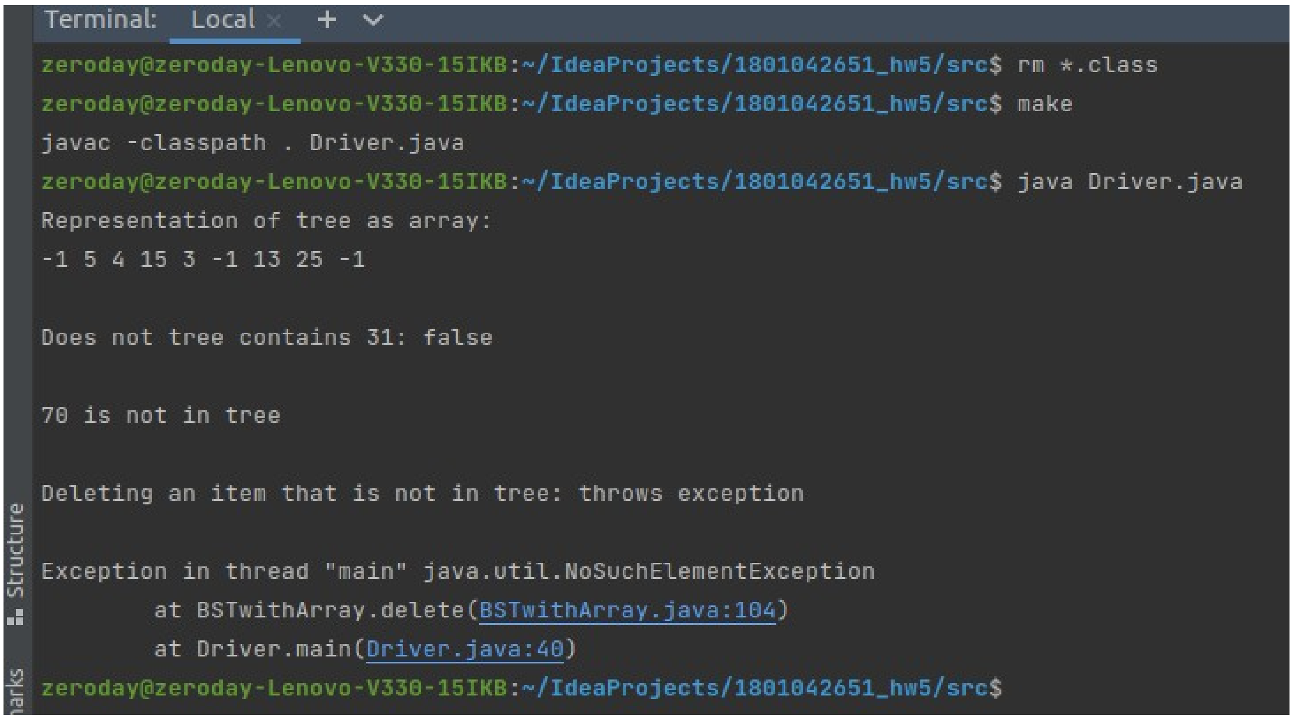
First, SearchTree interface is generated and started to code array based binary search tree.

While adding to array recursive algorithm is used; The root is always stored at index 1 in the array.
if any node is stored at k position then the left child of a node is stored at index 2k and the right child is stored at index 2k + 1 and the parent of a node is stored at floor(k/2) index.

While removing an element from tree, first it checks if tree contains element that will removed. If contains, finds index of this element and refresh the tree such that: If left child of removed index is not empty it replaces left child to parent and sets left child to empty; otherwise right child of removed index is not empty it replaces right child to parent and sets right child to empty.

Hint: -1 in BST array is represents the empty.

4) Test cases

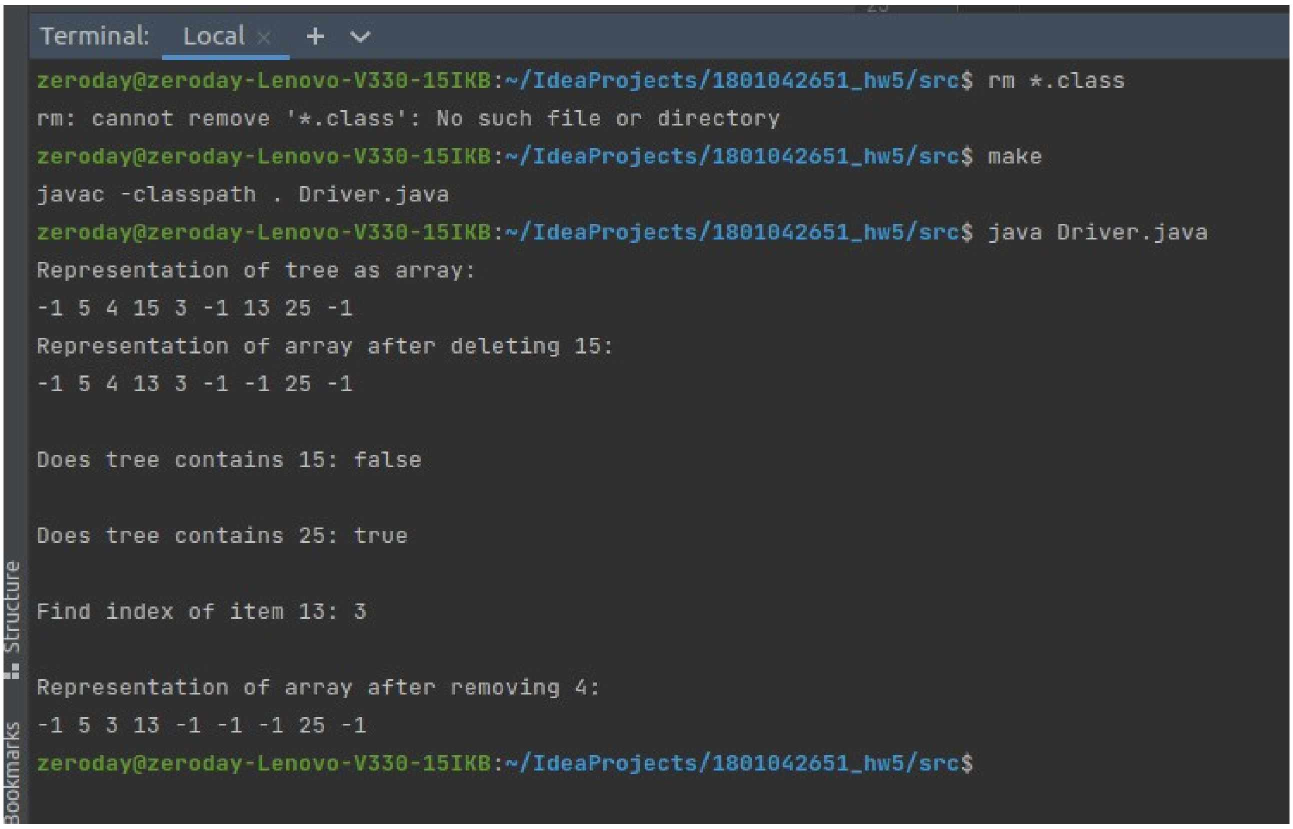


```
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/1801042651_hw5/src$ rm *.class
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/1801042651_hw5/src$ make
javac -classpath . Driver.java
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/1801042651_hw5/src$ java Driver.java
Representation of tree as array:
-1 5 4 15 3 -1 13 25 -1

Does not tree contains 31: false

70 is not in tree

Deleting an item that is not in tree: throws exception

Exception in thread "main" java.util.NoSuchElementException
        at BSTwithArray.delete(BSTwithArray.java:104)
        at Driver.main(Driver.java:40)
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/1801042651_hw5/src$
```

5) Running commands:

→ make
→ java Driver.java
if you would like to run one more time remove .class file
→ rm *.class

6) Running results:



```
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/1801042651_hw5/src$ rm *.class
rm: cannot remove '*.class': No such file or directory
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/1801042651_hw5/src$ make
javac -classpath . Driver.java
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/1801042651_hw5/src$ java Driver.java
Representation of tree as array:
-1 5 4 15 3 -1 13 25 -1
Representation of array after deleting 15:
-1 5 4 13 3 -1 -1 25 -1

Does tree contains 15: false

Does tree contains 25: true

Find index of item 13: 3

Representation of array after removing 4:
-1 5 3 13 -1 -1 -1 25 -1
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/1801042651_hw5/src$
```