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

Emircan Demirel
1901042674

1. SYSTEM REQUIREMENTS

Functional Requirements:

<u>Question 3:</u> Our binary heap should satisfy the structural property of being a complete tree besides the heap order property.

Node class must keep three different nodes: parent, left and right; and key value of the node.

If the element to be inserted has already in the binary heap, program must increase key value of old node.

The user can merge two different heaps.

<u>Question 4:</u> Binary Search Tree has to be represented in array structure.

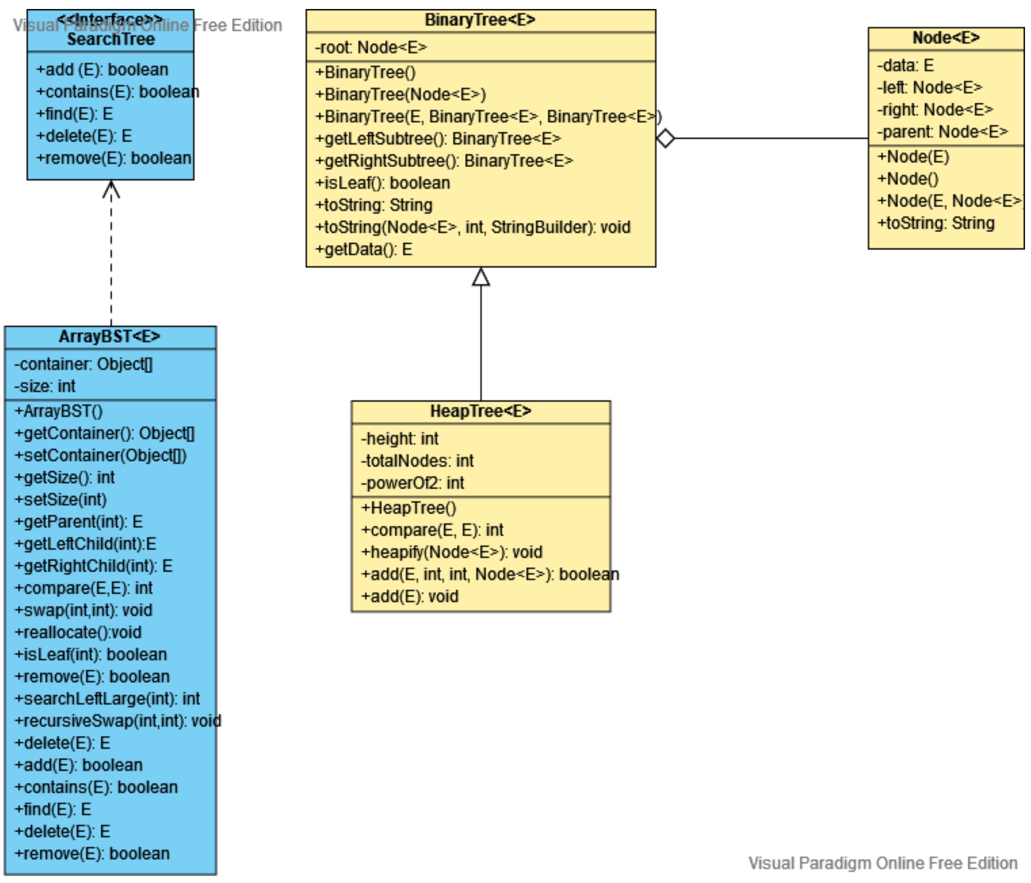
Our class implementation should implement all methods in SearchTree interface.

Non-Functional Requirements:

Java-SE13 or higher should be installed and running properly.

Program must handle errors.

2. CLASS DIAGRAMS



3. PROBLEM SOLUTION APPROACH

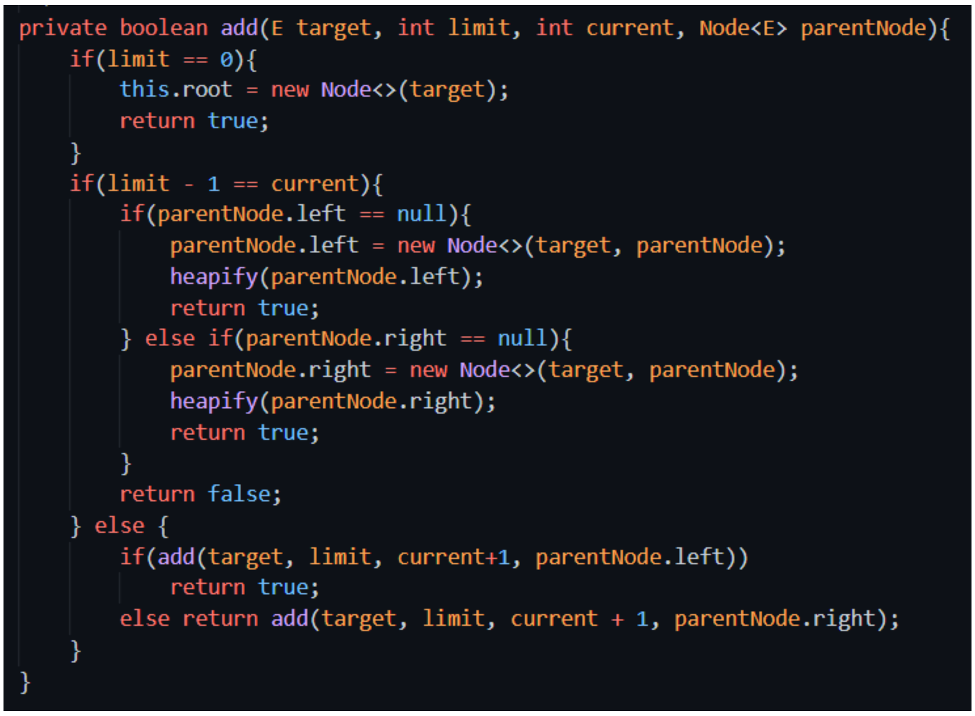
In Question 3, we have to create binary heap structure like binary search tree. To make this with tree structure I create HeapTree class which is extended by BinaryTree class use node link structure. This extended class has three nodes as parent, left and right. To implement add method in complete binary method I used three data field in HeapTree class: height, totalNodes and powerOf2. Thanks to these data fields, my recursive add method can insert new elements to the tree from left to right at each level. totalNodes counts number of elements at each level then if it is equal to powerOf2 counter, height value will be incremented.

In Question 4, we have to implement a binary search tree by using array and comparator. To initialize elements to the array I used subtree and parent formulas (for instance: left: 2*i + 1, right:2*(i+1)). After some comparison, elements are initialized to the array according to the binary search tree order.

I couldn’t implement removal method for both of two questions.

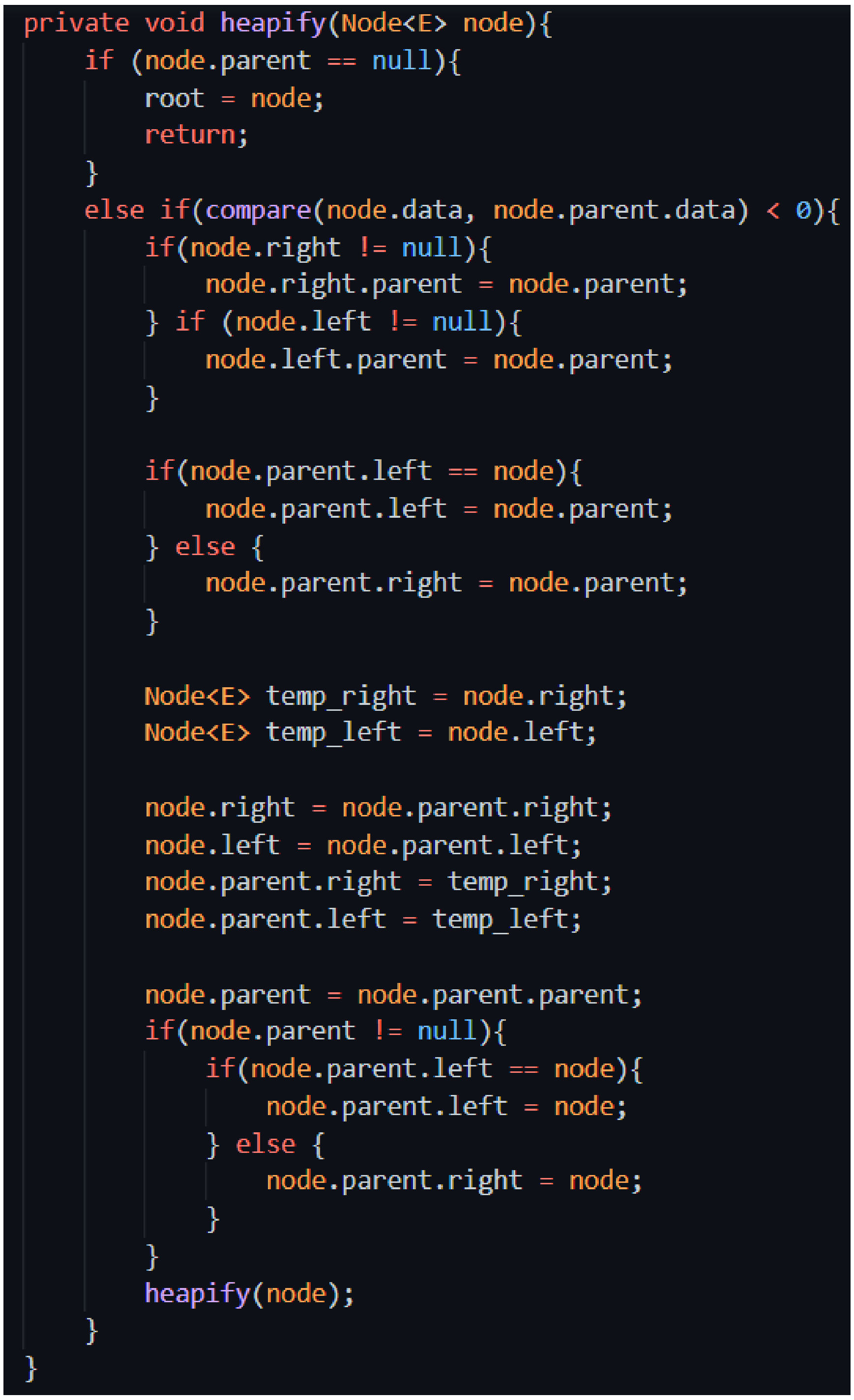
4. ANALYZES

Q3:



```
private boolean add(E target, int limit, int current, Node<E> parentNode){
    if(limit == 0){
        this.root = new Node<>(target);
        return true;
    }
    if(limit - 1 == current){
        if(parentNode.left == null){
            parentNode.left = new Node<>(target, parentNode);
            heapify(parentNode.left);
            return true;
        } else if(parentNode.right == null){
            parentNode.right = new Node<>(target, parentNode);
            heapify(parentNode.right);
            return true;
        }
        return false;
    } else {
        if(add(target, limit, current+1, parentNode.left))
            return true;
        else return add(target, limit, current + 1, parentNode.right);
    }
}
```

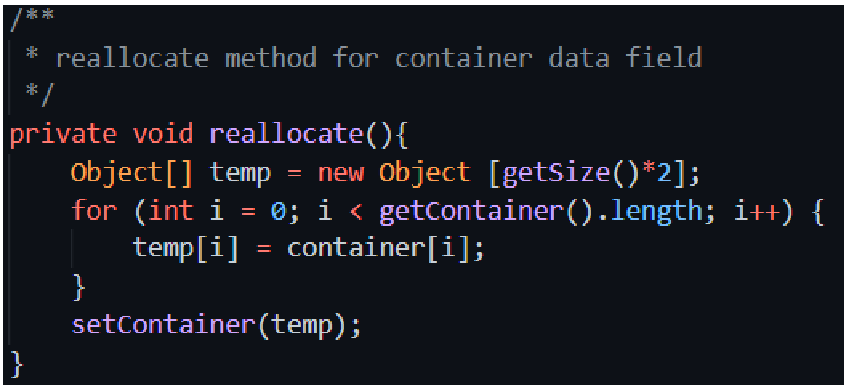
add method time complexity: O(n)



```java
private void heapify(Node<E> node){
    if (node.parent == null){
        root = node;
        return;
    }
    else if(compare(node.data, node.parent.data) < 0){
        if(node.right != null){
            node.right.parent = node.parent;
        } if (node.left != null){
            node.left.parent = node.parent;
        }

        if(node.parent.left == node){
            node.parent.left = node.parent;
        } else {
            node.parent.right = node.parent;
        }

        Node<E> temp_right = node.right;
        Node<E> temp_left = node.left;

        node.right = node.parent.right;
        node.left = node.parent.left;
        node.parent.right = temp_right;
        node.parent.left = temp_left;

        node.parent = node.parent.parent;
        if(node.parent != null){
            if(node.parent.left == node){
                node.parent.left = node;
            } else {
                node.parent.right = node;
            }
        }
        heapify(node);
    }
}
```

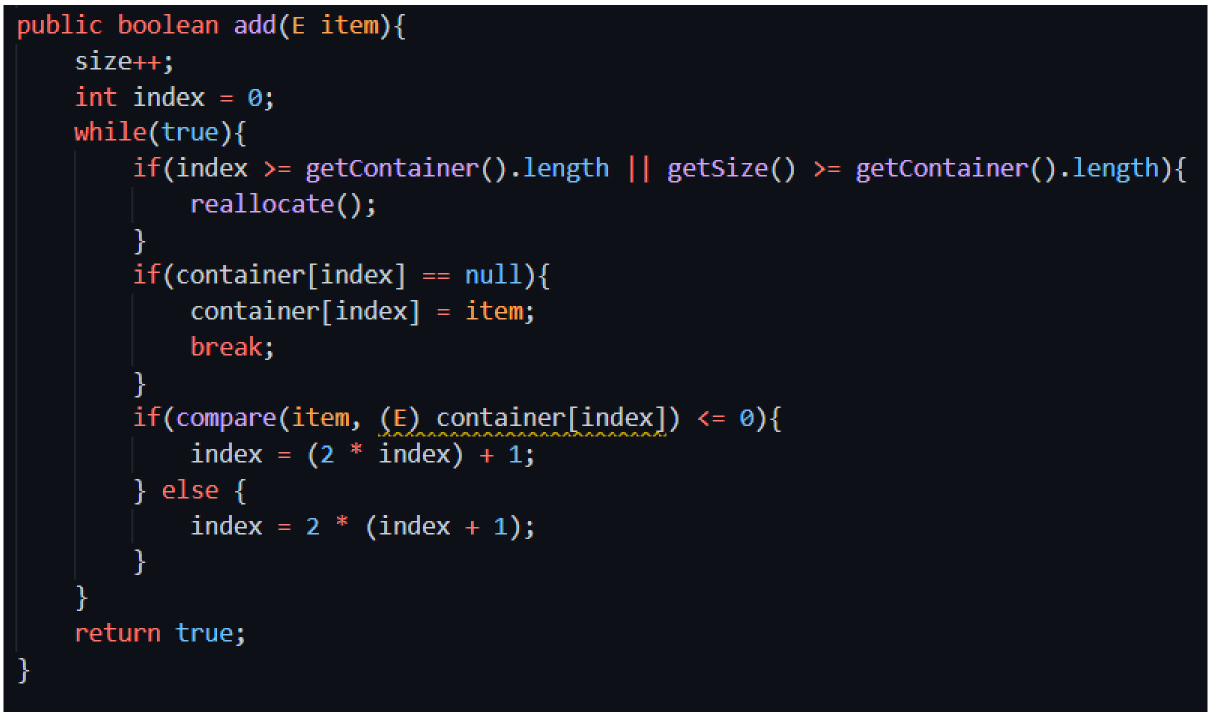
heapify method complexity: best case θ(1); worst case θ(h);

Q4:



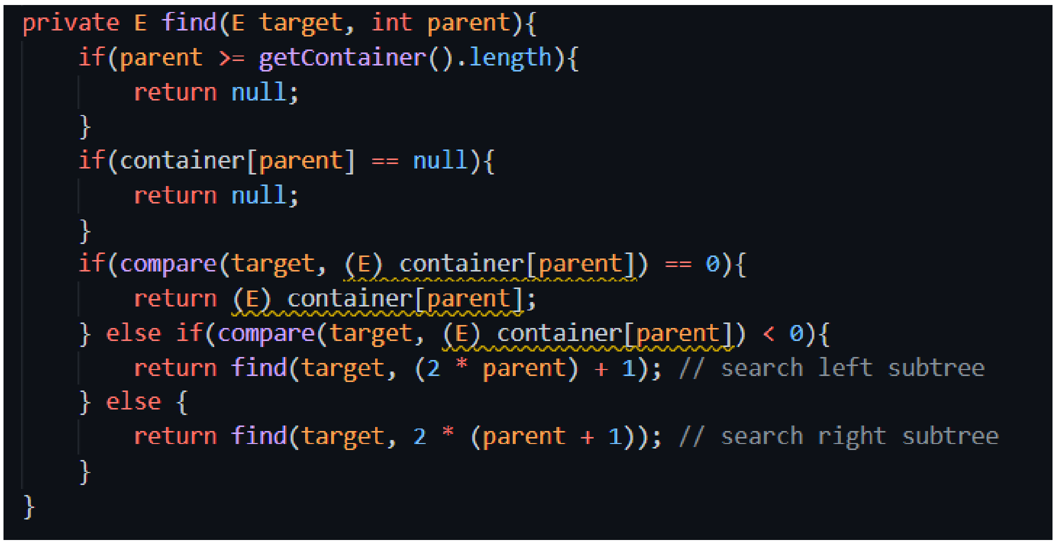
```
/**
 * reallocate method for container data field
 */
private void reallocate(){
    Object[] temp = new Object [getSize()*2];
    for (int i = 0; i < getContainer().length; i++) {
        temp[i] = container[i];
    }
    setContainer(temp);
}
```

Reallocate method: θ(n)



```
public boolean add(E item){
    size++;
    int index = 0;
    while(true){
        if(index >= getContainer().length || getSize() >= getContainer().length){
            reallocate();
        }
        if(container[index] == null){
            container[index] = item;
            break;
        }
        if(compare(item, (E) container[index]) <= 0){
            index = (2 * index) + 1;
        } else {
            index = 2 * (index + 1);
        }
    }
    return true;
}
```

Add method: best case θ(1); worst case θ(n * log n)



```
private E find(E target, int parent){
    if(parent >= getContainer().length){
        return null;
    }
    if(container[parent] == null){
        return null;
    }
    if(compare(target, (E) container[parent]) == 0){
        return (E) container[parent];
    } else if(compare(target, (E) container[parent]) < 0){
        return find(target, (2 * parent) + 1); // search left subtree
    } else {
        return find(target, 2 * (parent + 1)); // search right subtree
    }
}
```

Find and Contains Methods: O(h)

5. TEST CASES

Q3-)

1- Compile -> test add method by Using String

Q4-)

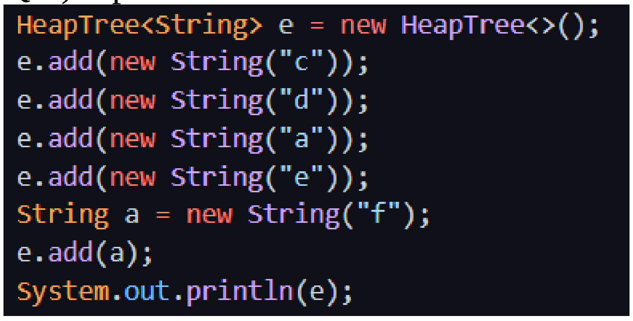
1- Compile -> test add method by using String

2- Compile -> test find method

3- Compile -> test contains method

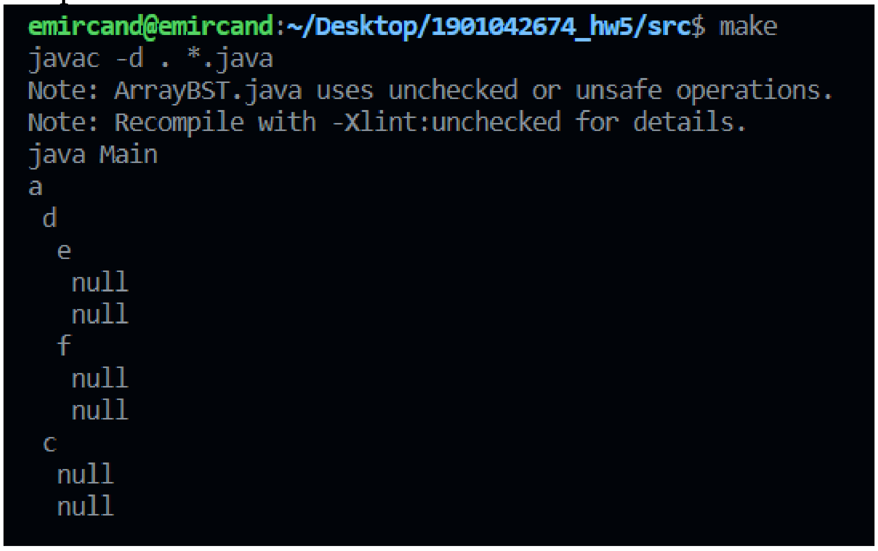
6. RUNNING AND RESULTS

Q3-) inputs



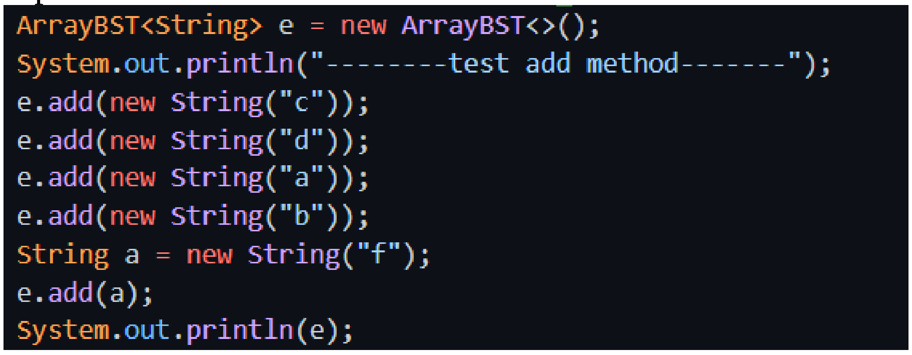
```
HeapTree<String> e = new HeapTree<>();
e.add(new String("c"));
e.add(new String("d"));
e.add(new String("a"));
e.add(new String("e"));
String a = new String("f");
e.add(a);
System.out.println(e);
```

Output



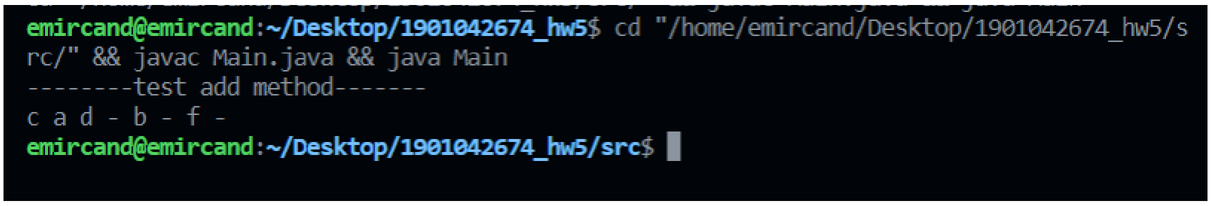
```
emircand@emircand:~/Desktop/1901042674_hw5/src$ make
javac -d . *.java
Note: ArrayBST.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
java Main
a
 d
  e
   null
   null
  f
   null
   null
 c
  null
  null
```

Q4-)
Inputs



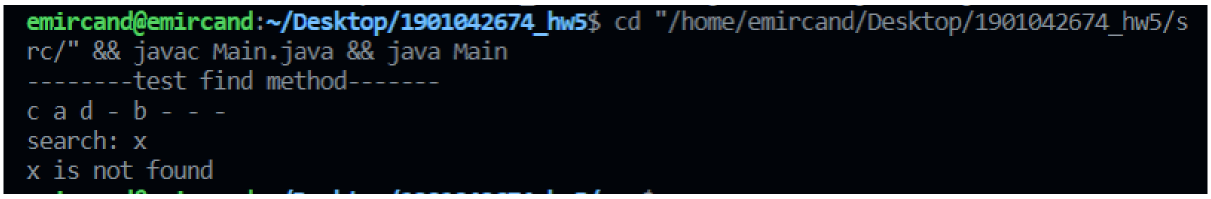
```java
ArrayBST<String> e = new ArrayBST<>();
System.out.println("--------test add method-------");
e.add(new String("c"));
e.add(new String("d"));
e.add(new String("a"));
e.add(new String("b"));
String a = new String("f");
e.add(a);
System.out.println(e);
```

1.

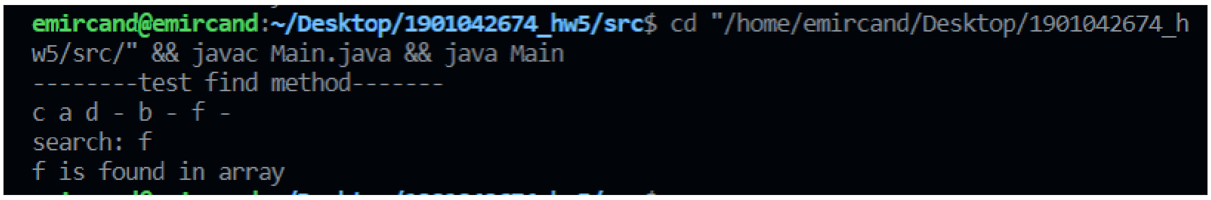


```
emircand@emircand:~/Desktop/1901042674_hw5$ cd "/home/emircand/Desktop/1901042674_hw5/s
rc/" && javac Main.java && java Main
--------test add method-------
c a d - b - f -
emircand@emircand:~/Desktop/1901042674_hw5/src$
```

2.

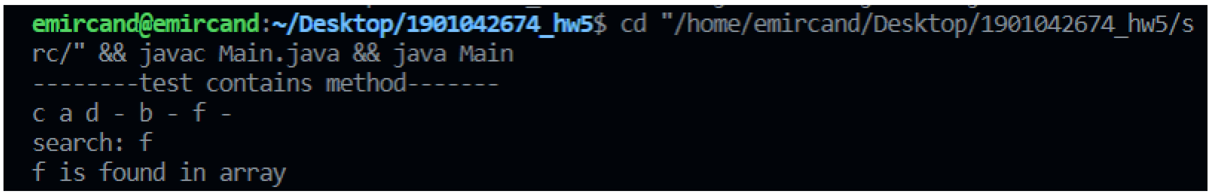


```
emircand@emircand:~/Desktop/1901042674_hw5$ cd "/home/emircand/Desktop/1901042674_hw5/s
rc/" && javac Main.java && java Main
--------test find method-------
c a d - b - - -
search: x
x is not found
```

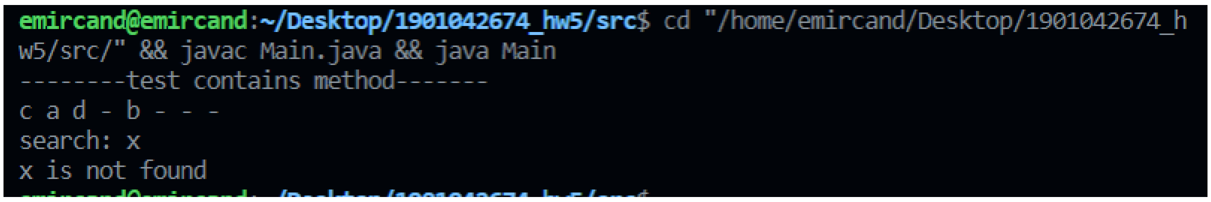


```
emircand@emircand:~/Desktop/1901042674_hw5/src$ cd "/home/emircand/Desktop/1901042674_h
w5/src/" && javac Main.java && java Main
--------test find method-------
c a d - b - f -
search: f
f is found in array
```

3.



```
emircand@emircand:~/Desktop/1901042674_hw5$ cd "/home/emircand/Desktop/1901042674_hw5/s
rc/" && javac Main.java && java Main
--------test contains method-------
c a d - b - f -
search: f
f is found in array
```



```
emircand@emircand:~/Desktop/1901042674_hw5/src$ cd "/home/emircand/Desktop/1901042674_h
w5/src/" && javac Main.java && java Main
--------test contains method-------
c a d - b - - -
search: x
x is not found
```