1. Write a Java program to

a. Search an item through linear search

```
package com.coparray.main;

import java.util.Scanner;

public class LinearSearch {

	public static void main(String[] args) {
		int [] arr= { 15 , 32 , 24, 67 ,49,10};
		Scanner sc= new Scanner(System.in);
		System.out.println("Enter Element to Search");
		int n = sc.nextInt();
	    int position=search( n,arr);
	    if(position>=0)
		  System.out.println("Element Found At Position "+(position+1));
	    else

				System.out.println("element was not found");

	}

	static int search(int element, int a[]) {
		for(int i=0;i<a.length;i++) {
			if(a[i]==element)
				return i;
		}

	   return -1;

	}

}
```

```
<terminated> LinearSearch [Java Application] C
Enter Element to Search
67
Element Found At Position 4
```

b. Create and delete nodes from binary search tree

```
package com.coparray.demo;

class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
```

```
        left = right = null;
    }
}

public class BinarySearchTree {

       Node root;

    BinarySearchTree() {
        root = null;
    }

    void insert(int key) {
        root = insertRec(root, key);
    }

    Node insertRec(Node root, int key) {
        if (root == null) {
            root = new Node(key);
            return root;
        }
        if (key < root.key) {
            root.left = insertRec(root.left, key);
        } else if (key > root.key) {
            root.right = insertRec(root.right, key);
        }
        return root;
    }

    void deleteKey(int key) {
        root = deleteRec(root, key);
    }

    Node deleteRec(Node root, int key) {
        if (root == null) {
            return root;
        }
        if (key < root.key) {
            root.left = deleteRec(root.left, key);
        } else if (key > root.key) {
            root.right = deleteRec(root.right, key);
        } else {
            if (root.left == null) {
                return root.right;
            } else if (root.right == null) {
                return root.left;
            }
            root.key = minValue(root.right);
            root.right = deleteRec(root.right, root.key);
        }
        return root;
    }

    int minValue(Node root) {
        int minv = root.key;
        while (root.left != null) {
            minv = root.left.key;
            root = root.left;
        }
```

```
        return minv;
    }

    void inorder() {
        inorderRec(root);
    }

    void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            System.out.print(root.key + " ");
            inorderRec(root.right);
        }
    }

    public static void main(String[] args) {
        BinarySearchTree tree = new BinarySearchTree();
        tree.insert(50);
        tree.insert(30);
        tree.insert(20);
        tree.insert(40);
        tree.insert(70);
        tree.insert(60);
        tree.insert(80);

        System.out.println("Inorder traversal of the given tree");
        tree.inorder();

        System.out.println("\nDelete 20");
        tree.deleteKey(20);
        System.out.println("Inorder traversal of the modified tree");
        tree.inorder();

        System.out.println("\nDelete 30");
        tree.deleteKey(30);
        System.out.println("Inorder traversal of the modified tree");
        tree.inorder();

        System.out.println("\nDelete 50");
        tree.deleteKey(50);
        System.out.println("Inorder traversal of the modified tree");
    }
}
```

```
<terminated> BinarySearchTree [Java Application] C:\Users\amrit\Desktop
Inorder traversal of the given tree
20 30 40 50 60 70 80
Delete 20
Inorder traversal of the modified tree
30 40 50 60 70 80
Delete 30
Inorder traversal of the modified tree
40 50 60 70 80
```