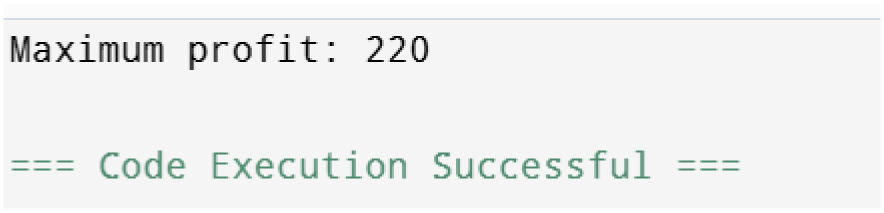
SANJAY S ,22IT093
PRACTICE_SET_2

1. 0-1 knapsack problem

```
public class Main {
    public static void main(String[] args) {
        int[] val = {60, 100, 120};
        int[] wt = {10, 20, 30};
        int capacity = 50;
        int maxProfit = knapSack(capacity, val, wt);
        System.out.println("Maximum profit: " + maxProfit);
    }
    static int knapSack(int capacity, int val[], int wt[]){
        int n=val.length;
        int[] prev=new int[capacity+1];
        for(int i=wt[0];i<=capacity;i++){
            prev[i]=val[0];
        }
         for(int ind=1;ind<n;ind++){
             int[] curr=new int[capacity+1];
            for(int weight=0;weight<=capacity;weight++){
                int take=Integer.MIN_VALUE;
                if(wt[ind]<= weight){
                    take=val[ind]+prev[weight-wt[ind]];
                }
                int notTake=prev[weight];
                curr[weight]=Math.max(take,notTake);
            }
            prev=curr;
        }
        return prev[capacity];
    }
```

}



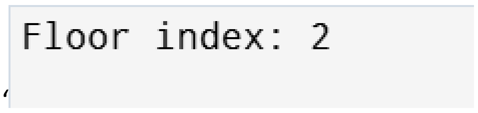
```
Maximum profit: 220

=== Code Execution Successful ===
```

TC: O(n * capacity)

SC: O(n * capacity)

2. Floor in sorted array

```
public class Main {
    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 6, 10};
        int k = 5;
        int floorIndex = findFloor(arr, k);
        System.out.println("Floor index: " + floorIndex);
    }
    static int findFloor(int[] arr, int k) {
        int n=arr.length;
        for(int i=n-1;i>=0;i--){
            if(arr[i]<=k){
                return i;
            }
        }
        return -1;
    }
}
```

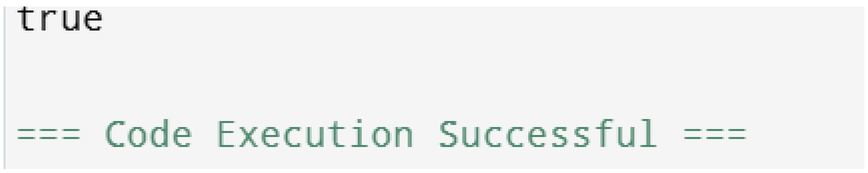


```
Floor index: 2
```

TC:O(n)

SC:O(n)

3. Check equal arrays

```
import java.util.*;
public class Main {
  public static void main(String[] args) {
    int[] arr1 = {1, 2, 3, 4, 5};
    int[] arr2 = {5, 4, 3, 2, 1};

    System.out.println(check(arr1, arr2));
  }
   public static boolean check(int[] arr1, int[] arr2) {
    if (arr1.length != arr2.length) return false;

    HashMap<Integer, Integer> freqMap = new HashMap<>();

    for (int num : arr1) {
      freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
    }

    for (int num : arr2) {
      if (!freqMap.containsKey(num) || freqMap.get(num) == 0) {
        return false;
      }
      freqMap.put(num, freqMap.get(num) - 1);
    }

    return true;
  }
}
```


```
true

=== Code Execution Successful ===
```

TC:O(n)

SC:O(n)

4. Palindrome linked list

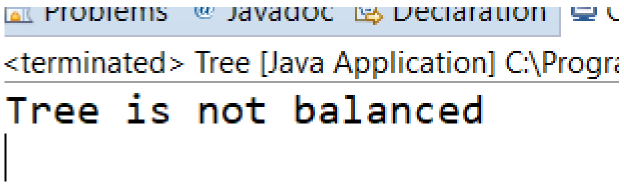
```
class Solution {
    // Function to check whether the list is palindrome.
    public Node reverse(Node node){
        Node prev=null;
        Node crr = node;
        Node next=null;
        while(crr!=null){
            next = crr.next;
            crr.next=prev;
            prev=crr;
            crr=next;
        }
        return prev;
    }
    boolean isPalindrome(Node head) {
        // Your code here
        if(head==null || head.next==null) return true;
        Node fast=head;
        Node slow=head;
        while(fast.next!=null && fast.next.next!=null){
            fast=fast.next.next;
            slow=slow.next;
        }
        Node reverseNode=reverse(slow.next);
        Node  temp1=head;
        Node temp2=reverseNode;
        while(temp2!=null){
            if(temp1.data!=temp2.data){
                return false;
            }
            temp1=temp1.next;
            temp2=temp2.next;
        }
        return true;
    }
}
```

TC:O(n)

SC:O(1)

5. Balanced tree check

```
public class Tree {

    // Function to check whether a binary tree is balanced or not.
    public int fun(Node node) {
        if (node == null) {
            return 0;
        }
        int leftHeight = fun(node.left);
        if (leftHeight == -1) {
            return -1;
        }
        int rightHeight = fun(node.right);
        if (rightHeight == -1) {
            return -1;
        }
        if (Math.abs(leftHeight - rightHeight) > 1) {
            return -1;
        }
        return 1 + Math.max(leftHeight, rightHeight);
    }
    boolean isBalanced(Node root) {
        return fun(root) != -1;
    }

    public static void main(String[] args) {
        Tree tree = new Tree();
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.left.left.left = new Node(6);

        if (tree.isBalanced(root)) {
            System.out.println("Tree is balanced");
        } else {
            System.out.println("Tree is not balanced");
        }
    }
}
```



```
<terminated> Tree [Java Application] C:\Progra
Tree is not balanced
```

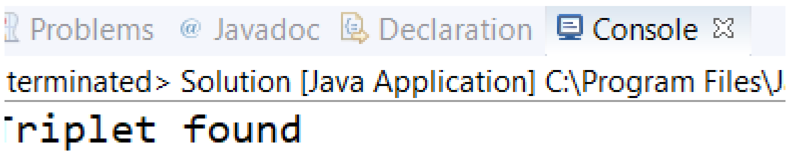
TC:O(n)

SC:O(h)

6. Triplet sum in array

```
import java.util.Arrays;

public class Solution {
    public static boolean find3Numbers(int arr[], int n, int x) {
```

```
    if (n < 3) return false;
    Arrays.sort(arr);
    for (int i = 0; i < n - 2; i++) {
        int j = i + 1;
        int k = n - 1;
        while (j < k) {
            int sum = arr[i] + arr[j] + arr[k];
            if (sum == x) {
                return true;
            } else if (sum < x) {
                j++;
            } else {
                k--;
            }
        }
    }
    return false;
}

public static void main(String[] args) {
    int arr[] = {12, 3, 4, 1, 6, 9};
    int x = 24;
    int n = arr.length;
    if (find3Numbers(arr, n, x)) {
        System.out.println("Triplet found");
    } else {
        System.out.println("No triplet found");
    }
}
```



```
terminated> Solution [Java Application] C:\Program Files\J
riplet found
```

}

TC:O(n^2)

SC:O(1)