0-1 KnapSack Problem

```
import java.util.Scanner;
class Knapsack {

static int knapSack(int capacity, int[] weights, int[] values, int n) {
    if (n == 0 || capacity == 0)
      return 0;
    if (weights[n - 1] > capacity)
      return knapSack(capacity, weights, values, n - 1);
    else
      return Math.max(knapSack(capacity, weights, values, n - 1),
            values[n - 1] + knapSack(capacity - weights[n - 1], weights, values, n - 1));
  }

  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter the number of items: ");
    int n = sc.nextInt();

    int[] values = new int[n];
    int[] weights = new int[n];

    System.out.println("Enter the values");
    for (int i = 0; i < n; i++) {
      values[i] = sc.nextInt();
    }
```

```
        System.out.println("Enter the values");
        for (int i = 0; i < n; i++) {
            weights[i] = sc.nextInt();
        }

        System.out.print("Enter the capacity of the knapsack: ");
        int capacity = sc.nextInt();

        System.out.println( knapSack(capacity, weights, values, n));
    }
}
```

```
Enter the number of items: 3
Enter the values
1
2
3
Enter the Weights
4
5
1
Enter the capacity of the knapsack: 4
3
```

Time Complexity : O(2^n)

Space Complexity:O(n)

2)Floor in Sorted Array

```
import java.util.*;
public class FloorArray{
	public static void main(String args[]){
		Scanner sc = new Scanner(System.in);
		int n = sc.nextInt();
		int[] arr = new int[n];
		for(int i=0;i<n;i++){
			arr[i] = sc.nextInt();
		}
		int x = sc.nextInt();
		System.out.println(array(n,arr,x));
	}

	public static int array(int n,int[] arr,int x){



		if(arr[0]>x) return -1;
		if(arr[n-1]<x) return arr[n-1];

		for(int i=1;i<arr.length;i++){
			if(arr[i]>x){
				return i-1;
			}
		}
		return -1;
	}
}
```

```
Enter the no.of values
6
Enter the arr values
1
2
8
8
10
12
Enter the X value
9
8
```

Time Complexity : O(n)

Space Complexity:O(n)

3)Equals Array

```
import java.util.*;
public class EqualArray{

    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the n values");
        int n = sc.nextInt();

        int[] arr1 = new int[n];
        for(int i=0;i<n;i++){
            arr1[i] = sc.nextInt();
        }
        System.out.println("Enter the m values");
```

```
            int m = sc.nextInt();

            int[] arr2 = new int[m];
            for(int i=0;i<m;i++){
                arr2[i] = sc.nextInt();
            }
            System.out.println(Array(arr1,arr2));

        }

        public static Boolean Array(int[] arr1,int[] arr2){

            Arrays.sort(arr1);
            Arrays.sort(arr2);
            if(arr1.length!=arr2.length) return false;
            return Arrays.equals(arr1,arr2);
        }
}
```

```
Enter the n values
3
1
2
1
Enter the m values
3
1
2
1
true
```

Time Complexity : O(n)

Space Complexity : O(n)

4)Palindrome Linked list

```
import java.util.*;


class Node {
   int val;
   Node next;


   Node() {}
   Node(int val) { this.val = val; }
   Node(int val, Node next) { this.val = val; this.next = next; }
}


public class LinkedList {


   public static Node createLinkedlist(int n, Scanner sc) {
      Node head = null;
      Node current = null;


      for (int i = 0; i < n; i++) {
         int a = sc.nextInt();
         if (head == null) {
            head = new Node(a);
            current = head;
         } else {
            current.next = new Node(a);
            current = current.next;
         }
      }
```

```
        return head;
    }


    public static boolean palindrome(Node head){
        Stack<Integer>a = new Stack<>();
        Node cur = head;
        while(head!=null){
            a.push(head.val);
            head = head.next;
        }
        while(cur!=null){
            if(a.isEmpty()|| cur.val!=a.peek()){
                return false;
            }
            cur = cur.next;
            a.pop();
        }
        return true;
}


    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of nodes: ");
        int n = sc.nextInt();
```

```
        Node head = createLinkedlist(n, sc);

        System.out.println(palindrome(head));

        sc.close();
    }
}
```

```
Enter the number of nodes: 4
1
2
2
1
true
```

Time Complexity : O(n)

Space Complexity: O(n)

5)Balanced Tree Check

```
class Node {
    int data;
    Node left, right;
    Node(int d) {
        data = d;
        left = right = null;
    }
}

class BinaryTree {
    Node root;
```

```
boolean isBalanced(Node node) {
   int l;
   int r;

   if (node == null)
      return true;

   l = height(node.left);
   r = height(node.right);

   if (Math.abs(l - r) <= 1 && isBalanced(node.left) && isBalanced(node.right))
      return true;

   return false;
}

int height(Node node) {
   if (node == null)
      return 0;

   return 1 + Math.max(height(node.left), height(node.right));
}

public static void main(String args[]) {
   BinaryTree tree = new BinaryTree();
   tree.root = new Node(1);
   tree.root.left = new Node(2);
```

```
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.left.left.left = new Node(8);

        if (tree.isBalanced(tree.root))
            System.out.println("Balanced");
        else
            System.out.println("Not Balanced");
    }
}
```

```
Not Balanced
```

Time Complexity : O(n^2)

Space Complexity : O(n)

6)Triplet Sum

```
import java.util.Arrays;
import java.util.Scanner;

public class Triplet {
    static boolean find3Numbers(int[] arr, int sum) {
        int n = arr.length;
        Arrays.sort(arr);
        for (int i = 0; i < n - 2; i++) {
            int l = i + 1;
            int r = n - 1;
            while (l < r) {
```

```
            int curr = arr[i] + arr[l] + arr[r];
            if (curr == sum) {

                return true;
            } else if (curr < sum) {
                l++;
            } else {
                r--;
            }
        }
    }
    return false;
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();

    int[] arr = new int[n];

    for (int i = 0; i < n; i++) {
        arr[i] = sc.nextInt();
    }

    int sum = sc.nextInt();
```

```
        System.out.println(find3Numbers(arr, sum));


    }
}
```

```
5
1
2
3
4
5
6
true
```

Time Complexity : O(n^2)

Space Complexity:O(n)