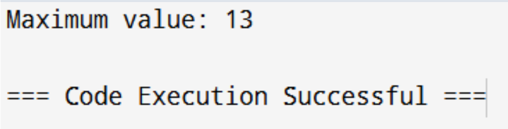
1. 0-1 Knapsack problem:

CODE:

```
import java.util.*;
class Main {
    public static int knapsack(int[] wt, int[] v, int ind, int w, int[][] dp){
        if(ind==0){
            if(wt[0]<=w){
                return v[0];
            }else{
                return 0;
            }
        }
        if(dp[ind][w]!=-1){
            return dp[ind][w];
        }
        int notake=0+knapsack(wt,v,ind-1,w,dp);
        int take=Integer.MIN_VALUE;
        if(wt[ind]<=w){
            take=v[ind]+knapsack(wt,v,ind-1,w-wt[ind],dp);
        }
        dp[ind][w]=Math.max(notake,take);
        return dp[ind][w];
    }
    static int knapsack(int[] wt,int[] v, int n, int w){
        int dp[][]=new int[n][w+1];
        for(int r[]:dp){
            Arrays.fill(r,-1);
        }
        return knapsack(wt,v,n-1,w,dp);
    }
    public static void main(String[] args) {
        int wt[]={1,2,4,5};
        int v[]={5,4,8,6};
        int w=5;
        int n=wt.length;
        System.out.println("Maximum value: " +knapsack(wt,v,n,w));
    }
}
```

OUTPUT:



```
Maximum value: 13

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

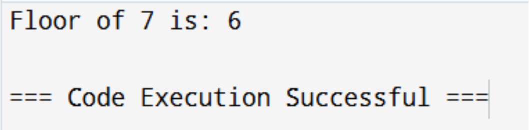
Time Complexity: O(n*w)

2. Floor in sorted array:

CODE:

```
import java.util.*;
class Main {
    public static int floorelement(int[] a, int l, int h, int x){
        if(l>h) return -1;
        if(x>=a[h]) return h;
        int m=(l+h)/2;
        if(a[m]==x) return m;
        if(m>0 && a[m-1]<=x && x<a[m]) return m-1;
        if(x<a[m]) return floorelement(a,l,m-1,x);
        return floorelement(a,m+1,h,x);
    }
    public static void main(String[] args) {
        int a[]={1,2,4,6,10,12,14};
        int x=7;
        int n=a.length;
        int ind=floorelement(a,0,n-1,x);
        if(ind==-1)
            System.out.println("Floor of"+x+"doesn't exist");
        else
            System.out.println("Floor of " +x+ " is: " +a[ind]);
    }
}
```

OUTPUT:



```
Floor of 7 is: 6

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

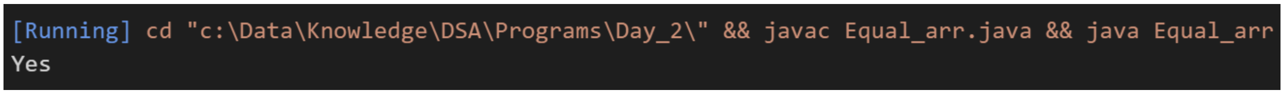
Time Complexity: O(log n)

3. Check equal arrays:

CODE:

```
import java.io.*;
import java.util.*;
class Equal_arr{
    public static boolean equal(int a1[], int a2[]){
        int n=a1.length;
        int m=a2.length;
        if(n!=m) return false;
        Map<Integer, Integer> map=new HashMap<Integer, Integer>();
        int c=0;
        for(int i=0;i<n;i++){
            if(map.get(a1[i])==null) map.put(a1[i],1);
            else{
                c=map.get(a1[i]);
                c++;
                map.put(a1[i],c);
            }
        }
        for(int i=0;i<n;i++){
            if(!map.containsKey(a2[i])) return false;
            if(map.get(a2[i])==0) return false;
            c=map.get(a2[i]);
            --c;
            map.put(a2[i],c);
        }
        return true;
    }
    public static void main(String[] args){
        int a1[]={3,5,2,5,2};
        int a2[]={2,3,5,5,2};
        if(equal(a1,a2)) System.out.println("Yes");
        else System.out.println("No");
    }
}
```

OUTPUT:



```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_2\" && javac Equal_arr.java && java Equal_arr
Yes
```

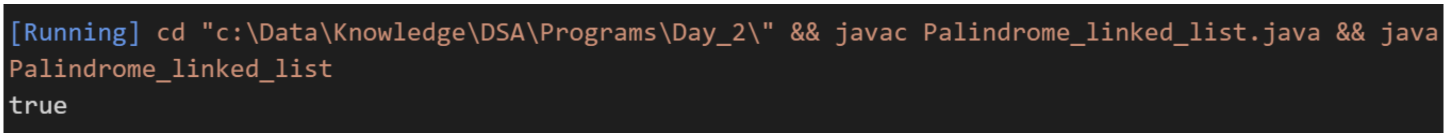
Time Complexity: O(n)

4. Palindrome linked list:

CODE:

```
class Node{
    int data;
    Node next;
    Node(int d){
        data=d;
        next=null;
    }
}
class Palindrome_linked_list{
    static Node reverse(Node head){
        Node prev=null;
        Node curr=head;
        Node next;
        while(curr!=null){
            next=curr.next;
            curr.next=prev;
            prev=curr;
            curr=next;
        }
        return prev;
    }
    static boolean identical(Node n1, Node n2){
        while(n1!=null && n2!=null){
            if(n1.data!=n2.data) return false;
            n1=n1.next;
            n2=n2.next;
        }
        return true;
    }
    static boolean palindrome(Node head){
        if(head==null || head.next==null) return true;
        Node slow=head, fast=head;
        while(fast.next!=null && fast.next.next!=null){
            slow=slow.next;
            fast=fast.next.next;
        }
        Node head2=reverse(slow.next);
        slow.next=null;
        boolean ret=identical(head,head2);
        head2=reverse(head2);
        slow.next=head2;
        return ret;
    }
    public static void main(String[] args){
        Node head=new Node(1);
        head.next=new Node(2);
        head.next.next=new Node(3);
        head.next.next.next=new Node(2);
        head.next.next.next.next=new Node(1);
        boolean res=palindrome(head);
        if(res) System.out.println("true");
        else System.out.println("false");
    }
}
```

OUTPUT:



```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_2\" && javac Palindrome_linked_list.java && java
Palindrome_linked_list
true
```

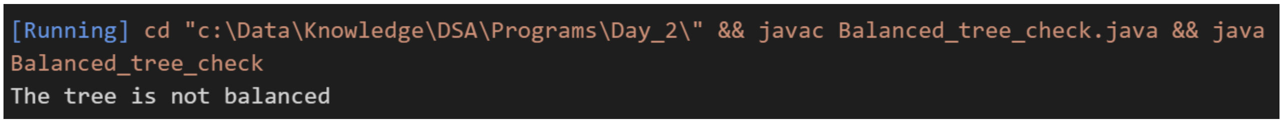
Time Complexity: O(n)

5. Balanced tree check:

CODE:

```
class Node{
    int data;
    Node left;
    Node right;
    Node(int val){
        data=val;
        left=null;
        right=null;
    }
}
class Balanced_tree_check{
    public boolean balance(Node root){
        return dfs(root)!=-1;
    }
    public int dfs(Node root){
        if(root==null) return 0;
        int lh=dfs(root.left);
        if(lh==-1) return -1;
        int rh=dfs(root.right);
        if(rh==-1) return -1;
        if(Math.abs(lh-rh)>1) return -1;
        return Math.max(lh,rh)+1;
    }
    public static void main(String[] args){
        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.right.right=new Node(6);
        root.left.right.right.right=new Node(7);
        Balanced_tree_check checker=new Balanced_tree_check();
        if(checker.balance(root)){
            System.out.println("The tree is balanced");
        }else{
            System.out.println("The tree is not balanced");
        }
    }
}
```

OUTPUT:



```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_2\" && javac Balanced_tree_check.java && java
Balanced_tree_check
The tree is not balanced
```

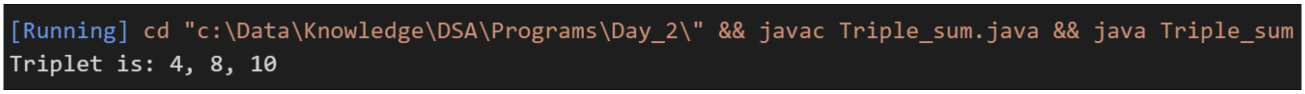
Time Complexity: O(n)

6. Triplet sum in array:

CODE:

```
import java.util.*;
public class Triple_sum{
    static boolean tripletsum(int[] a, int sum){
        int n=a.length;
        for(int i=0;i<n-2;i++){
            Set<Integer> s=new HashSet<>();
            int currsum=sum-a[i];
            for(int j=i+1;j<n;j++){
                int required=currsum-a[j];
                if(s.contains(required)){
                    System.out.println("Triplet is: "+a[i]+", "+a[j]+", "+required);
                    return true;
                }
                s.add(a[j]);
            }
        }
        return false;
    }
    public static void main(String[] args){
        int[] a={1,4,45,6,10,8};
        int sum=22;
        tripletsum(a,sum);
    }
}
```

OUTPUT:



```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Day_2\" && javac Triple_sum.java && java Triple_sum
Triplet is: 4, 8, 10
```

Time Complexity: O(n^2)