1. 0-1 knapsack problem

Given N items where each item has some weight and profit associated with it and also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible.

Note: The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

Examples:

Input: N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}
Output: 3
Explanation: There are two items which have weight less than or equal to 4. If we select the item with weight 4, the possible profit is 1. And if we select the item with weight 1, the possible profit is 3. So the maximum possible profit is 3. Note that we cannot put both the items with weight 4 and 1 together as the capacity of the bag is 4.
Input: N = 3, W = 3, profit[] = {1, 2, 3}, weight[] = {4, 5, 6}
Output: 0

Solution:

```
import java.io.*;
class Knapsack {
   static int maxVal(int w, int wt[], int val[], int n) {
       if (n == 0 || w == 0) return 0;
       if (wt[n - 1] > w) return maxVal(w, wt, val, n - 1);
       return Math.max(val[n - 1] + maxVal(w - wt[n - 1], wt, val, n - 1), maxVal(w, wt, val, n -
1));
   }

   public static void main(String[] args) {
       int[] wt = {10, 20, 30};
       int[] val = {60, 100, 120};
       int w = 50;
       int n = val.length;
       System.out.println(maxVal(w, wt, val, n));
   }
}
```

Output:

```
ramyabharathi@Ramyas-Air Practiseset2-11_11 % javac Knapsack.java
ramyabharathi@Ramyas-Air Practiseset2-11_11 % java Knapsack
220
```

Time Complexity: O(2^n)

2.Floor in sorted array

Given a sorted array and a value x, the floor of x is the largest element in the array smaller than or equal to x. Write efficient functions to find the floor of x

Examples:

Input: arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 5

Output: 2

Explanation: 2 is the largest element in

arr[] smaller than 5

Input: arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 20

Output: 19

Explanation: 19 is the largest element in

arr[] smaller than 20

Input : arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 0

Output : -1

Explanation: Since floor doesn't exist, output is -1.

Solution:

```
import java.util.*;
class Floorsort {
   static int findFloor(int[] arr, int n, int x) {
       int l = 0, h = n - 1, res = -1;
       while (l <= h) {
           int m = l + (h - l) / 2;
           if (arr[m] == x) return arr[m];
           if (arr[m] < x) {
               res = arr[m];
               l = m + 1;
           } else {
               h = m - 1;
           }
       }
       return res;
   }
   public static void main(String[] args) {
       int[] arr = {1, 2, 8, 10, 10, 12, 19};

       System.out.println(findFloor(arr, arr.length, 5));
       System.out.println(findFloor(arr, arr.length, 20));
       System.out.println(findFloor(arr, arr.length, 0));
   }
}
```

Output:

```
ramyabharathi@Ramyas-Air Practiseset2-11_11 % javac Floorsort.java
ramyabharathi@Ramyas-Air Practiseset2-11_11 % java Floorsort
2
19
-1
```

Time Complexity:O(logn)

3.Check equal arrays

Given two arrays, arr1 and arr2 of equal length N, the task is to determine if the given arrays are equal or not. Two arrays are considered equal if:

- Both arrays contain the same set of elements.
- The arrangements (or permutations) of elements may be different.
- If there are repeated elements, the counts of each element must be the same in both arrays.

Input: arr1[] = {1, 2, 5, 4, 0}, arr2[] = {2, 4, 5, 0, 1}
Output: Yes
Input: arr1[] = {1, 2, 5, 4, 0, 2, 1}, arr2[] = {2, 4, 5, 0, 1, 1, 2}
Output: Yes
Input: arr1[] = {1, 7, 1}, arr2[] = {7, 7, 1}
Output: No

Solution:

```java
import java.util.*;

class Equalarray {
   static boolean checkEqual(int[] arr1, int[] arr2) {
       if (arr1.length != arr2.length) return false;
       Arrays.sort(arr1);
       Arrays.sort(arr2);
       return Arrays.equals(arr1, arr2);
   }

   public static void main(String[] args) {
       int[] arr1a = {1, 2, 5, 4, 0};
       int[] arr2a = {2, 4, 5, 0, 1};
       int[] arr1b = {1, 2, 5, 4, 0, 2, 1};
       int[] arr2b = {2, 4, 5, 0, 1, 1, 2};
       int[] arr1c = {1, 7, 1};
       int[] arr2c = {7, 7, 1};
       System.out.println(checkEqual(arr1a, arr2a) ? "Yes" : "No");
       System.out.println(checkEqual(arr1b, arr2b) ? "Yes" : "No");
       System.out.println(checkEqual(arr1c, arr2c) ? "Yes" : "No");
   }
}
```

Output:

```
ramyabharathi@Ramyas-Air Practiseset2-11_11 % javac Equalarray.java
ramyabharathi@Ramyas-Air Practiseset2-11_11 % java Equalarray
Yes
Yes
No
```

Time Complexity: O(nlogn)

4.Palindrome linked list

Given a singly linked list. The task is to check if the given linked list is palindrome or not.

Input: head: 1->2->1->1->2->1

Output: true

Explanation: The given linked list is 1->2->1->1->2->1 , which is a palindrome and Hence, the output is true.

Input: head: 1->2->3->4

Output: false

Explanation: The given linked list is 1->2->3->4, which is not a palindrome and Hence, the output is false.

Solution:

```
import java.util.*;
class LinkedPalindrome {
   static class Node {
       int data;
       Node next;
       Node(int data) {
           this.data = data;
           this.next = null;
       }
   }

   static boolean isPalindrome(Node head) {
       if (head == null || head.next == null) return true;

       Node slow = head, fast = head, prev = null;
       while (fast != null && fast.next != null) {
           fast = fast.next.next;
           Node next = slow.next;
           slow.next = prev;
           prev = slow;
           slow = next;
       }
       if (fast != null) slow = slow.next;
       while (prev != null && slow != null) {
           if (prev.data != slow.data) return false;
           prev = prev.next;
           slow = slow.next;
       }
       return true;
   }
   public static void main(String[] args) {
       Node head1 = new Node(1);
       head1.next = new Node(2);
       head1.next.next = new Node(1);
       head1.next.next.next = new Node(1);
```

```
        head1.next.next.next.next = new Node(2);
        head1.next.next.next.next.next = new Node(1);
        System.out.println(isPalindrome(head1));

        Node head2 = new Node(1);
        head2.next = new Node(2);
        head2.next.next = new Node(3);
        head2.next.next.next = new Node(4);
        System.out.println(isPalindrome(head2));
    }
}
```

Output:

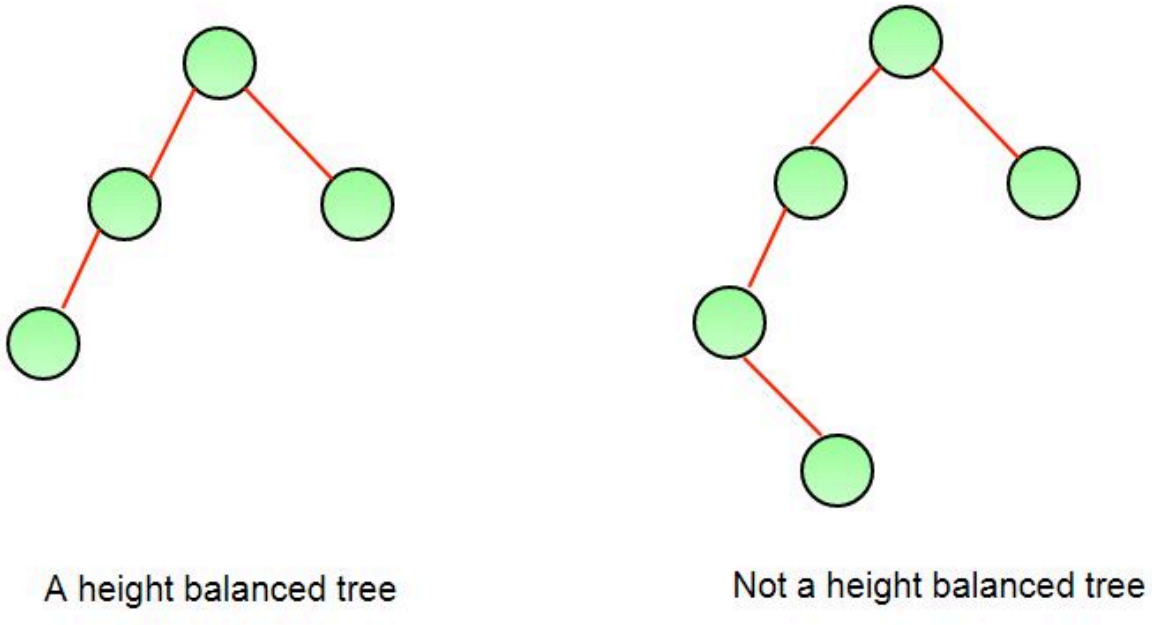
```
ramyabharathi@Ramyas-Air Practiseset2-11_11 % javac LinkedPalindrome.java
ramyabharathi@Ramyas-Air Practiseset2-11_11 % java LinkedPalindrome
true
false
```

Time Complexity: O(n)

5.Balanced tree check

A height balanced binary tree is a binary tree in which the height of the left subtree and right subtree of any node does not differ by more than 1 and both the left and right subtree are also height balanced.

Examples: The tree on the left is a height balanced binary tree. Whereas the tree on the right is not a height balanced tree. Because the left subtree of the root has a height which is 2 more than the height of the right subtree.



Corner Cases : An empty binary tree (Root = NULL) and a Binary Tree with single node are considered balanced.

Solution:

```
import java.util.*;
class BalancedBT {
   static class Node {
       int data;
       Node left, right;
       Node(int data) {
           this.data = data;
           this.left = this.right = null;
       }
   }

   static boolean isBalanced(Node root) {
       return checkHeight(root) != -1;
   }

   static int checkHeight(Node node) {
       if (node == null) return 0;
       int leftHeight = checkHeight(node.left);
       if (leftHeight == -1) return -1;
       int rightHeight = checkHeight(node.right);
       if (rightHeight == -1) return -1;
       if (Math.abs(leftHeight - rightHeight) > 1) return -1;
       return Math.max(leftHeight, rightHeight) + 1;
   }

   public static void main(String[] args) {
       Node root1 = new Node(1);
       root1.left = new Node(2);
       root1.right = new Node(3);
       root1.left.left = new Node(4);
       root1.left.right = new Node(5);
       root1.right.right = new Node(6);
       System.out.println(isBalanced(root1));

       Node root2 = new Node(1);
       root2.left = new Node(2);
       root2.left.left = new Node(3);
       System.out.println(isBalanced(root2));

       Node root3 = new Node(1);
       root3.left = new Node(2);
       root3.right = new Node(3);
       root3.left.left = new Node(4);
       root3.left.right = new Node(5);
```

```
        root3.right.left = new Node(6);
        root3.right.right = new Node(7);
        root3.left.left.left = new Node(8);
        root3.left.left.right = new Node(9);
        root3.left.right.left = new Node(10);
        root3.left.right.right = new Node(11);
        root3.right.left.left = new Node(12);
        root3.right.left.right = new Node(13);
        System.out.println(isBalanced(root3));
    }
}
```

Output:

```
ramyabharathi@Ramyas-Air Practiseset2-11_11 % javac BalancedBT.java
ramyabharathi@Ramyas-Air Practiseset2-11_11 % java BalancedBT
true
false
true
```

Time Complexity:O(n)

6.Triplet sum in array

Given an array arr[] of size n and an integer sum. Find if there's a triplet in the array which sums up to the given integer sum.

Examples:

Input: arr = {12, 3, 4, 1, 6, 9}, sum = 24;
Output: 12, 3, 9
Explanation: There is a triplet (12, 3 and 9) present in the array whose sum is 24.
Input: arr = {1, 2, 3, 4, 5}, sum = 9
Output: 5, 3, 1
Explanation: There is a triplet (5, 3 and 1) present in the array whose sum is 9.

Input: arr = {2, 10, 12, 4, 8}, sum = 9
Output: No Triplet
Explanation: We do not print in this case and return false.

Solution:

```
import java.util.Arrays;
class TripletSum {
    static boolean findTriplet(int[] arr, int n, int sum) {
        Arrays.sort(arr);
        for (int i = 0; i < n - 2; i++) {
            int l = i + 1, r = n - 1;
            while (l < r) {
                int currentSum = arr[i] + arr[l] + arr[r];
                if (currentSum == sum) {
                    System.out.println(arr[i] + ", " + arr[l] + ", " + arr[r]);
```

```
                    return true;
                }
                if (currentSum < sum) l++;
                else r--;
            }
        }
        System.out.println("No Triplet");
        return false;
    }
    public static void main(String[] args) {
        int[] arr1 = {12, 3, 4, 1, 6, 9};
        int sum1 = 24;
        int[] arr2 = {1, 2, 3, 4, 5};
        int sum2 = 9;
        int[] arr3 = {2, 10, 12, 4, 8};
        int sum3 = 9;
        findTriplet(arr2, arr2.length, sum2);
        findTriplet(arr1, arr1.length, sum1);
        findTriplet(arr3, arr3.length, sum3);
    }
}
```

Output:

```
ramyabharathi@Ramyas-Air Practiseset2-11_11 % javac TripletSum.java
ramyabharathi@Ramyas-Air Practiseset2-11_11 % java TripletSum
1, 3, 5
3, 9, 12
No Triplet
```

Time Complexity:O(n^2)