Coding Practice-2

1. 0 - 1 Knapsack Problem

You are given the weights and values of items, and you need to put these items in a knapsack of capacity **capacity** to achieve the maximum total value in the knapsack. Each item is available in only one quantity.

In other words, you are given two integer arrays **val[]** and **wt[]**, which represent the values and weights associated with items, respectively. You are also given an integer **capacity**, which represents the knapsack capacity. Your task is to find the maximum sum of values of a subset of val[] such that the sum of the weights of the corresponding subset is less than or equal to **capacity**. You cannot break an item; you must either pick the entire item or leave it (0-1 property).

Examples :

Input: capacity = 4, val[] = [1, 2, 3], wt[] = [4, 5, 1]
Output: 3
Explanation: Choose the last item, which weighs 1 unit and has a value of 3.

Input: capacity = 3, val[] = [1, 2, 3], wt[] = [4, 5, 6]
Output: 0
Explanation: Every item has a weight exceeding the knapsack's capacity (3).

Code:

```
class GfG {
    static int knapSack(int W, int wt[], int val[], int n) {
        if (n == 0 || W == 0)
            return 0;

        if (wt[n - 1] > W)
            return knapSack(W, wt, val, n - 1);

        return Math.max(knapSack(W, wt, val, n - 1),
            val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1));
```

```
    }

    public static void main(String args[]) {
        int profit[] = new int[] { 60, 100, 120 };
        int weight[] = new int[] { 10, 20, 30 };
        int W = 50;
        int n = profit.length;
        System.out.println(knapSack(W, weight, profit, n));
    }
}
```

Output:220

Time complexity: $O(2^n)$

2. Floor in sorted array

Given a sorted array **arr[]** (with unique elements) and an integer **k**, find the index (0-based) of the largest element in arr[] that is less than or equal to k. This element is called the "floor" of k. If such an element does not exist, return -1.

Examples

Input: arr[] = [1, 2, 8, 10, 11, 12, 19], k = 0
Output: -1
Explanation: No element less than 0 is found. So output is -1.

Input: arr[] = [1, 2, 8, 10, 11, 12, 19], k = 5
Output: 1
Explanation: Largest Number less than 5 is 2 , whose index is 1.

Input: arr[] = [1, 2, 8], k = 1
Output: 0
Explanation: Largest Number less than or equal to 1 is 1 , whose index is 0.

Code:

```
import java.io.*;
import java.lang.*;
import java.util.*;

class FloorArray {

	static int floorSearch(int arr[], int n, int x) {
		if (x >= arr[n - 1])
			return n - 1;

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

		for (int i = 1; i < n; i++)
			if (arr[i] > x)
				return (i - 1);

		return -1;
	}

	public static void main(String[] args) {
		int arr[] = { 1, 2, 4, 6, 10, 12, 14 };
		int n = arr.length;
		int x = 7;
		int index = floorSearch(arr, n - 1, x);
		if (index == -1)
			System.out.print( x );
		else
			System.out.print( arr[index]);
	}
```

}

Output:6

Time Complexity:O(n)

3. Check equal arrays

Given two arrays **arr1** and **arr2** of equal size, the task is to find whether the given arrays are equal. Two arrays are said to be equal if both contain the same set of elements, arrangements (or permutations) of elements may be different though.
Note: If there are repetitions, then counts of repeated elements must also be the same for two arrays to be equal.

Examples:

Input: arr1[] = [1, 2, 5, 4, 0], arr2[] = [2, 4, 5, 0, 1]
Output: true
Explanation: Both the array can be rearranged to [0,1,2,4,5]

Input: arr1[] = [1, 2, 5], arr2[] = [2, 4, 15]
Output: false
Explanation: arr1[] and arr2[] have only one common value.

Code:

```
class Equal {
   static boolean check(int[] arr1, int[] arr2) {
      if(arr1.length!=arr2.length){
         return false;
      }
      for(int i=0;i<arr1.length-1;i++){
         for(int j=0;j<arr2.length;j++){
            if(arr1[i]==arr2[j]){
               return true;
            }
         }
      }
```

```
        return false;
    }
    public static void main(String[] args){
        int arr1[] = { 3, 5, 2, 5, 2 };
        int arr2[] = { 2, 3, 5, 5, 2 };


        if (check(arr1, arr2))
            System.out.println("true");
        else
            System.out.println("false");
    }


    }
```

Output: true

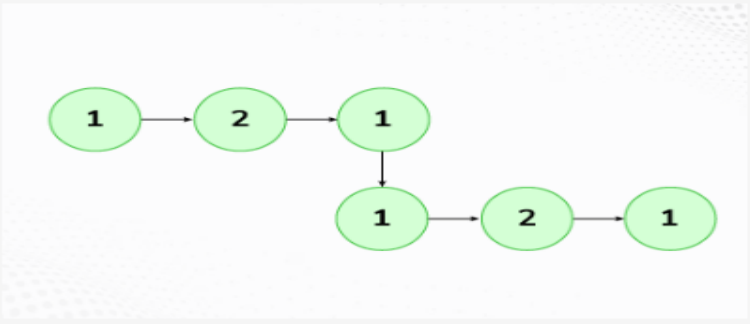
Time complexity: $O(n^{2)}$

4.Palindrome linked list

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

Examples:

Input: LinkedList: 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.



Code:

```
class Node {
    int data;
    Node next;
    Node(int d) {
        data = d;
        next = null;
    }
}

class Pal {

    static Node reverseList(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 isIdentical(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 isPalindrome(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 = reverseList(slow.next);
    slow.next = null;

    boolean ret = isIdentical(head, head2);

    head2 = reverseList(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 result = isPalindrome(head);

        if (result)
            System.out.println("true");
        else
            System.out.println("false");
    }
}
```

Output:true

Time complexity:O(n)

5. Balanced tree check

Given a binary tree, find if it is height balanced or not. A tree is height balanced if difference between heights of left and right subtrees is **not more than one** for all nodes of tree.

Examples:

Input:

```
      1
    /
   2
    \
     3
```

Output: 0

Explanation: The max difference in height of left subtree and right subtree is 2, which is greater than 1. Hence unbalanced

Code:

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

class BalancedTree {
    Node root;

    boolean isBalanced(Node node) {
        int lh;
        int rh;

        if (node == null)
            return true;

        lh = height(node.left);
        rh = height(node.right);

        if (Math.abs(lh - rh) <= 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[]) {
        BalancedTree tree = new BalancedTree();
        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("Tree is balanced");
        else
            System.out.println("Tree is not balanced");
    }
}
```

Output: Tree is not Balanced

Time complexity:$O(n^{2)}$

6. Triplet sum in array

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

Examples

Input:n = 6, x = 13, arr[] = [1,4,45,6,10,8]
Output: 1
Explanation: The triplet {1, 4, 8} in the array sums up to 13.

Input: n = 6, x = 10, arr[] = [1,2,4,3,6,7]
Output: 1
Explanation: Triplets {1,3,6} & {1,2,7} in the array sum to 10.

Input: n = 6, x = 24, arr[] = [40,20,10,3,6,7]
Output: 0
Explanation: There is no triplet with sum 24.

Code:

```
import java.util.HashSet;
import java.util.Set;

public class GfG {
    static boolean find3Numbers(int[] arr, int sum) {
        int n = arr.length;

        for (int i = 0; i < n - 2; i++) {
            Set<Integer> s = new HashSet<>();
            int curr_sum = sum - arr[i];

            for (int j = i + 1; j < n; j++) {
                int required_value = curr_sum - arr[j];

                if (s.contains(required_value)) {
                    System.out.println("Triplet is " + arr[i] + ", " + arr[j] + ", " + required_value);
                    return true;
                }

                s.add(arr[j]);
```

```
            }
        }
        return false;
    }

    public static void main(String[] args) {
        int[] arr = { 1, 4, 45, 6, 10, 8 };
        int sum = 22;

        find3Numbers(arr, sum);
    }
}
```

Output: Triplet is 4, 8, 10

Time Complexity:O($n^{2)}$