**11-11-2024 DSA PRACTICE PROBLEMS SET 2**

**1. 0/1 Knapsack Problem**

**CODE:**

import *java.util.\**;

*public* *class* Knapsack {

*static* *int* knap(*int* *W*, *int* *wt*[], *int* *prof*[], *int* *n*)

    {

*int*[] dp = new *int*[W + 1];

        for (*int* i = 1; i < n + 1; i++) {

            for (*int* w = W; w >= 0; w--) {

                if (wt[i - 1] <= w)

                    dp[w]

                        = Math.max(dp[w], dp[w - wt[i - 1]]

                                              + prof[i - 1]);

            }

        }

        return dp[W];

    }

*public* *static* *void* main(*String*[] *args*)

    {

*int* prof[] = { 60, 100, 120 };

*int* wt[] = { 10, 20, 30 };

*int* W = 50;

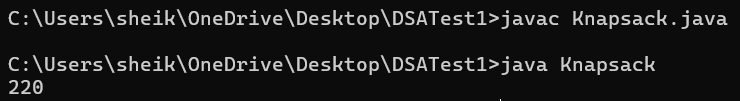
*int* n = prof.length;

        System.out.print(knap(W, wt, prof, n));

    }

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N \* W)**

**2.Floor In A Sorted Array**

**CODE:**

import *java.util.\**;

*class* FloorInSorted {

*public* *static* *void* main(*String*[] *args*) {

*ArrayList*<*Integer*> nums = new *ArrayList*<>();

*Scanner* scan = new Scanner(System.in);

        System.out.print("Enter number of elements: ");

*int* n = scan.nextInt();

        System.out.println("Enter elements in sorted order:");

        for (*int* i = 0; i < n; i++) {

*int* ele = scan.nextInt();

            nums.add(ele);

        }

        System.out.print("Enter target: ");

*int* target = scan.nextInt();

*int* result = floorSorted(nums, target);

        if (result != -1) {

            System.out.println("Index of Floor of Target: " + result);

            System.out.println("Floor Element: " + nums.get(result));

        } else {

            System.out.println("Floor of target not found");

        }

        scan.close();

    }

*public* *static* *int* floorSorted(*ArrayList*<*Integer*> *nums*, *int* *target*) {

*int* low = 0, high = nums.size() - 1;

*int* ans = -1;

        while (low <= high) {

*int* mid = low + (high - low) / 2;

            if (nums.get(mid) <= target) {

                ans = mid;

                low = mid + 1;

            } else {

                high = mid - 1;

            }

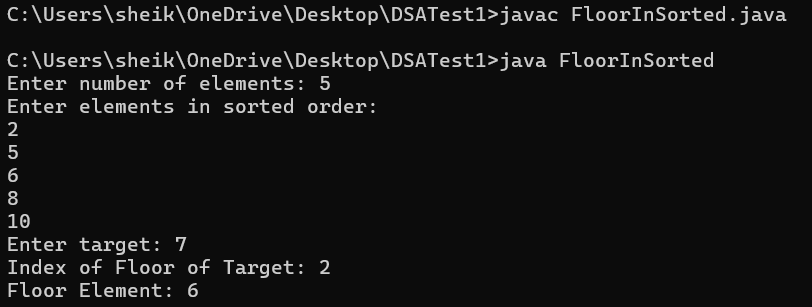
        }

        return ans;

    }

}

**OUTPUT:**



**TIME COMPLEXITY: O(log(N))**

**3. Check Equal Array**

**CODE:**

import *java.util.HashMap*;

import *java.util.Map*;

import *java.util.Scanner*;

*public* *class* EqualArray {

*public* *static* *boolean* areEqual(*int* *arr1*[], *int* *arr2*[]) {

*int* N = arr1.length;

*int* M = arr2.length;

        if (N != M)

            return false;

*Map*<*Integer*, *Integer*> map = new *HashMap*<>();

*int* cnt = 0;

        for (*int* i = 0; i < N; i++) {

            if (map.get(arr1[i]) == null)

                map.put(arr1[i], 1);

            else {

                cnt = map.get(arr1[i]);

                cnt++;

                map.put(arr1[i], cnt);

            }

        }

        for (*int* i = 0; i < N; i++) {

            if (!map.containsKey(arr2[i]) || map.get(arr2[i]) == 0)

                return false;

            cnt = map.get(arr2[i]);

            --cnt;

            map.put(arr2[i], cnt);

        }

        return true;

    }

*public* *static* *void* main(*String*[] *args*) {

*Scanner* scanner = new Scanner(System.in);

        System.out.print("Enter the size of the arrays: ");

*int* n = scanner.nextInt();

*int*[] arr1 = new *int*[n];

*int*[] arr2 = new *int*[n];

        System.out.println("Enter elements of first array:");

        for (*int* i = 0; i < n; i++) {

            arr1[i] = scanner.nextInt();

        }

        System.out.println("Enter elements of second array:");

        for (*int* i = 0; i < n; i++) {

            arr2[i] = scanner.nextInt();

        }

        if (areEqual(arr1, arr2))

            System.out.println("Yes");

        else

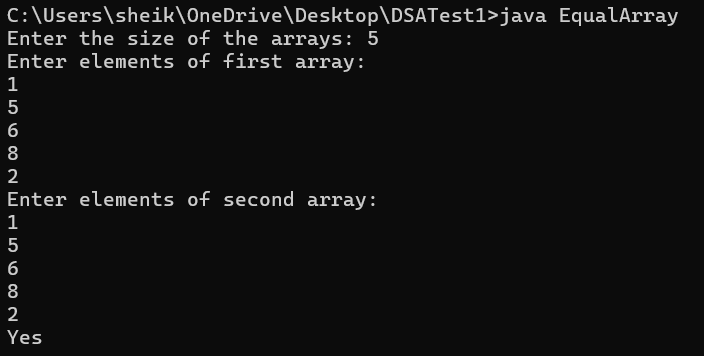
            System.out.println("No");

        scanner.close();

    }

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**4. Palindrome Linked List**

**CODE:**

import java.util.Scanner;

import java.util.Stack;

class Node {

int data;

Node next;

Node(int d) {

data = d;

next = null;

}

}

public class NodePalindrome {

static boolean isPalindrome(Node head) {

Node cur = head;

Stack<Integer> stack = new Stack<>();

while (cur != null) {

stack.push(cur.data);

cur = cur.next;

}

while (head != null) {

int val = stack.pop();

if (head.data != val) {

return false;

}

head = head.next;

}

return true;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of nodes in the linked list: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the linked list:");

Node head = null, tail = null;

for (int i = 0; i < n; i++) {

int data = scanner.nextInt();

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

boolean result = isPalindrome(head);

if (result)

System.out.println("true");

else

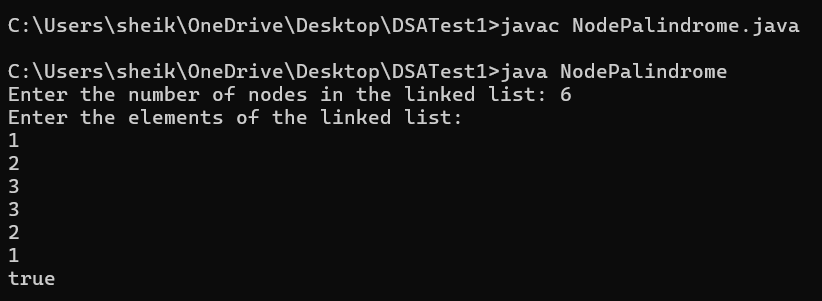
System.out.println("false");

scanner.close();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**5.Balanced Tree Check**

**CODE:**

import *java.lang.Math*;

*class* Node {

*int* data;

*Node* left;

*Node* right;

    Node(*int* *val*) {

        data = val;

        left = null;

        right = null;

    }

}

*class* BinTree {

*public* *boolean* isBalanced(*Node* *root*) {

        return dfsHeight(root) != -1;

    }

*public* *int* dfsHeight(*Node* *root*) {

        if (root == null) return 0;

*int* leftHeight = dfsHeight(root.left);

        if (leftHeight == -1)

            return -1;

*int* rightHeight = dfsHeight(root.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* 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);

*BinTree* solution = new BinTree();

        if (solution.isBalanced(root)) {

            System.out.println("The tree is balanced.");

        } else {

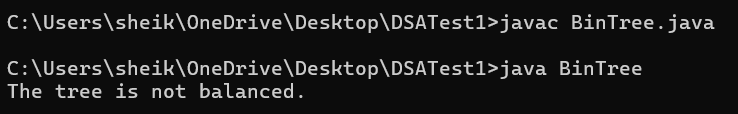
            System.out.println("The tree is not balanced.");

        }

    }

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**6. Triplet Sum In An Array**

**CODE:**

import *java.util.\**;

*public* *class* TripletSum {

*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\_sum = arr[i] + arr[l] + arr[r];

                if (curr\_sum == sum) {

                    System.out.println("Triplet is " + arr[i] + ", " + arr[l] + ", " + arr[r]);

                    return true;

                } else if (curr\_sum < sum) {

                    l++;

                } else {

                    r--;

                }

            }

        }

        return false;

    }

*public* *static* *void* main(*String*[] *args*) {

*Scanner* scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements: ");

*int* n = scanner.nextInt();

*int*[] arr = new *int*[n];

        System.out.println("Enter the elements:");

        for (*int* i = 0; i < n; i++) {

            arr[i] = scanner.nextInt();

        }

        System.out.print("Enter the target sum: ");

*int* sum = scanner.nextInt();

        if (!find3Numbers(arr, sum)) {

            System.out.println("No triplet found");

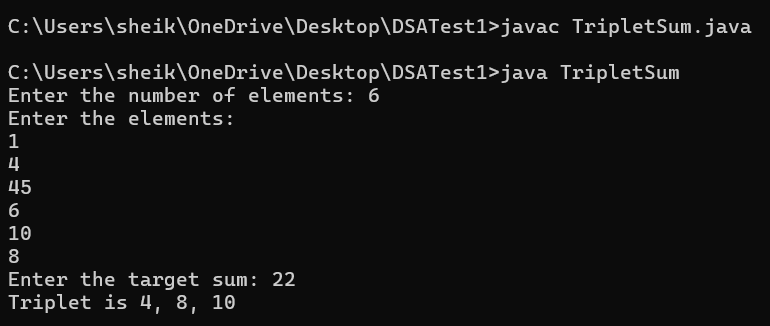
        }

        scanner.close();

    }

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N^2)**