1.0-1 Knapsack problem

class Knapsack {

    public static int knapSack(int capacity, int[] wt, int[] val, int n) {

        int[][] dp = new int[n + 1][capacity + 1];

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

            for (int w = 0; w <= capacity; w++) {

                if (i == 0 || w == 0)

                    dp[i][w] = 0;

                else if (wt[i - 1] <= w)

                    dp[i][w] = Math.max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);

                else

                    dp[i][w] = dp[i - 1][w];

            }

        }

        return dp[n][capacity];

    }

    public static void main(String[] args) {

        int[] val = { 10, 40, 30, 50 };

        int[] wt = { 5, 4, 6, 3 };

        int capacity = 5;

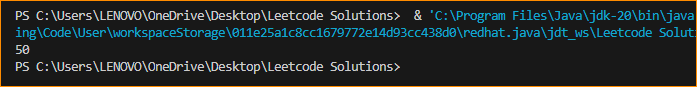
        int n = val.length;

        System.out.println(knapSack(capacity, wt, val, n));

    }

}

Output



Time Complexity

O(n\*w)

2.Check Equal Arrays

import java.util.HashMap;

public class Checkequalarrays {

    public static void main(String[] args) {

        int[] arr1 = { 1, 2, 5, 4, 0 };

        int[] arr2 = { 2, 4, 5, 0, 1 };

        System.out.println(areArraysEqual(arr1, arr2));

        int[] arr3 = { 1, 2, 5 };

        int[] arr4 = { 2, 4, 15 };

        System.out.println(areArraysEqual(arr3, arr4));

    }

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

        if (arr1.length != arr2.length) {

            return false;

        }

        HashMap<Integer, Integer> countMap = new HashMap<>();

        for (int num : arr1) {

            countMap.put(num, countMap.getOrDefault(num, 0) + 1);

        }

        for (int num : arr2) {

            if (!countMap.containsKey(num)) {

                return false;

            }

            countMap.put(num, countMap.get(num) - 1);

            if (countMap.get(num) == 0) {

                countMap.remove(num);

            }

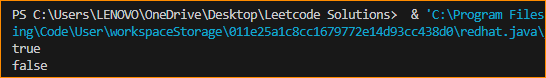
        }

        return countMap.isEmpty();

    }

}

Output



Time Complexity

O(n)

3.Floor sorted array

public class Floorsortedarray {

    public static void main(String[] args) {

        int[] arr1 = { 1, 2, 8, 10, 11, 12, 19 };

        int k1 = 0;

        System.out.println(findFloor(arr1, k1));

        int[] arr2 = { 1, 2, 8, 10, 11, 12, 19 };

        int k2 = 5;

        System.out.println(findFloor(arr2, k2));

        int[] arr3 = { 1, 2, 8 };

        int k3 = 1;

        System.out.println(findFloor(arr3, k3));

    }

    public static int findFloor(int[] arr, int k) {

        int left = 0;

        int right = arr.length - 1;

        int result = -1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            if (arr[mid] <= k) {

                result = mid;

                left = mid + 1;

            } else {

                right = mid - 1;

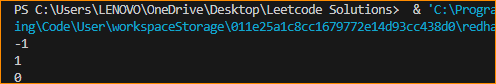
            }

        }

        return result;

    }}

Output



Time Complexity

O(log n)

4.Palindrome Linked List

class ListNode {

    int val;

    ListNode next;

    ListNode(int val) {

        this.val = val;

        this.next = null;

    }

}

public class Palindromelinkedlist {

    public static void main(String[] args) {

        ListNode head1 = new ListNode(1);

        head1.next = new ListNode(2);

        head1.next.next = new ListNode(1);

        head1.next.next.next = new ListNode(1);

        head1.next.next.next.next = new ListNode(2);

        head1.next.next.next.next.next = new ListNode(1);

        System.out.println(isPalindrome(head1));

        ListNode head2 = new ListNode(1);

        head2.next = new ListNode(2);

        head2.next.next = new ListNode(3);

        head2.next.next.next = new ListNode(4);

        System.out.println(isPalindrome(head2));

    }

    public static boolean isPalindrome(ListNode head) {

        if (head == null || head.next == null) {

            return true;

        }

        ListNode slow = head, fast = head;

        while (fast != null && fast.next != null) {

            slow = slow.next;

            fast = fast.next.next;

        }

        ListNode secondHalf = reverseList(slow);

        ListNode secondHalfCopy = secondHalf;

        ListNode firstHalf = head;

        while (secondHalf != null) {

            if (firstHalf.val != secondHalf.val) {

                return false;

            }

            firstHalf = firstHalf.next;

            secondHalf = secondHalf.next;

        }

        reverseList(secondHalfCopy);

        return true;

    }

    private static ListNode reverseList(ListNode head) {

        ListNode prev = null, curr = head;

        while (curr != null) {

            ListNode next = curr.next;

            curr.next = prev;

            prev = curr;

            curr = next;

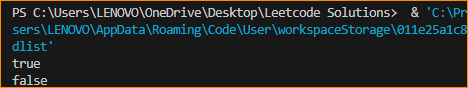
        }

        return prev;

    }

}

Output



Time Complexity

O(n)

5.Balanced tree check

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int item) {

        val = item;

        left = right = null;

    }

}

class Balancedtreecheck {

    private int checkHeight(TreeNode 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 + 1 - rightHeight) > 1) {

            return -1;

        }

        return Math.max(leftHeight, rightHeight) + 1;

    }

    public boolean isBalanced(TreeNode root) {

        return checkHeight(root) != -1;

    }

    public static void main(String[] args) {

        Balancedtreecheck tree = new Balancedtreecheck();

        TreeNode root1 = new TreeNode(1);

        root1.left = new TreeNode(2);

        root1.left.right = new TreeNode(3);

        System.out.println(tree.isBalanced(root1) ? 1 : 0);

        TreeNode root2 = new TreeNode(10);

        root2.left = new TreeNode(20);

        root2.right = new TreeNode(30);

        root2.left.left = new TreeNode(40);

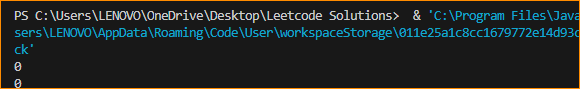
        root2.left.right = new TreeNode(60);

        System.out.println(tree.isBalanced(root2) ? 1 : 0);

    }

}

Output



Time Complexity

O(n)

6.Triplet sum

import java.util.Arrays;

import java.util.ArrayList;

public class Tripletsum {

    public static void main(String[] args) {

        int[] array = { 0, 0, 0 };

        ArrayList<int[]> zeroSumTriplets = findZeroSumTriplets(array);

        System.out.println("Triplet subarrays with sum 0:");

        if (zeroSumTriplets.isEmpty()) {

            System.out.println("[]");

        } else {

            for (int[] triplet : zeroSumTriplets) {

                System.out.println(Arrays.toString(triplet));

            }

        }

    }

    private static ArrayList<int[]> findZeroSumTriplets(int[] array) {

        ArrayList<int[]> zeroSumTriplets = new ArrayList<>();

        Arrays.sort(array);

        for (int i = 0; i < array.length - 2; i++) {

            if (i > 0 && array[i] == array[i - 1]) {

                continue;

            }

            int left = i + 1;

            int right = array.length - 1;

            while (left < right) {

                int sum = array[i] + array[left] + array[right];

                if (sum == 0) {

                    zeroSumTriplets.add(new int[] { array[i], array[left], array[right] });

                    while (left < right && array[left] == array[left + 1])

                        left++;

                    while (left < right && array[right] == array[right - 1])

                        right--;

                    left++;

                    right--;

                } else if (sum < 0) {

                    left++;

                } else {

                    right--;

                }

            }

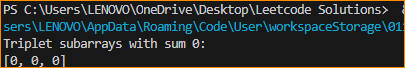
        }

        return zeroSumTriplets;

    }

}

Output



s

Time Complexity

O(n\*\*2)