11/11/2024 **DSA(practice -2)**

1.

0-1 Knapsack problem:

Coding:

```
import java.util.*;
public class Knapsack {
public static int knapsack(int[] weights, int[] profits, int capacity) {
int n = profits.length;
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 (weights[i - 1] <= w) {
dp[i][w] = Math.max(profits[i - 1] + dp[i - 1][w - weights[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[] weights = {10, 20, 30};
int[] profits = {60, 100, 120};
int capacity = 50;
int maxProfit = knapsack(weights, profits, capacity);
System.out.println("Maximum profit is: " + maxProfit);
}
}
```

Output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac Knapsack.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java Knapsack
Maximum profit is: 220
```

Time Complexity:o(m*n)

2.Floor sorting array:

Coding:

```
import java.util.*;
public class FloorInSort {
    public static int findFloor(int[] arr, int k) {
        int floorIndex = -1;

        for (int i = 0; i < arr.length; i++) {
            if (arr[i] <= k) {
                floorIndex = i;
            } else {
                break;            }
        }

        return floorIndex;
    }

    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));
    }
}
```

Output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac FloorInSort.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java FloorInSort
-1
1
0
```

3.Check equal arrays:

Coding:

```
import java.util.HashMap;
public class EqualArrays {
    public static boolean areEqual(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) || countMap.get(num) == 0) {
                return false;
            }
            countMap.put(num, countMap.get(num) - 1);
        }

        for (int count : countMap.values()) {
            if (count != 0) {
                return false;
            }
        }
        return true;
    }
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 5, 4, 0};
        int[] arr2 = {2, 4, 5, 0, 1};
        System.out.println(areEqual(arr1, arr2));

        int[] arr3 = {1, 2, 5};
        int[] arr4 = {2, 4, 15};
        System.out.println(areEqual(arr3, arr4));     }
}
```

Output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac EqualArrays.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java EqualArrays
true
false
```

4.Palindrome linked list:

Coding:

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}
public class Palindrome {
    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 firstHalf = head;

        while (secondHalf != null) {
            if (firstHalf.val != secondHalf.val) {
                return false;
            }
            firstHalf = firstHalf.next;
            secondHalf = secondHalf.next;
        }

        return true;
    }
    private static ListNode reverseList(ListNode head) {
        ListNode prev = null, current = head, next;
        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        return prev;
    }
    public static void main(String[] args) {
```

```
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(2);
        head.next.next.next = new ListNode(1);
        System.out.println(isPalindrome(head));
    }
}
```

Output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac Palindrome.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java Palindrome
true
```

5.Balanced tree check:

Coding:

```
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int x) {
        val = x;
        left = right = null;
    }
}
public class BalancedTree {
    public static boolean isBalanced(TreeNode root) {
        return checkHeight(root) != -1;
    }
    private static int checkHeight(TreeNode node) {
        if (node == null) {
            return 0;
        }
        int leftHeight = checkHeight(node.left);
        int rightHeight = checkHeight(node.right);
        if (leftHeight == -1 || rightHeight == -1 || Math.abs(leftHeight - rightHeight) > 1) {
            return -1;
        }
        return Math.max(leftHeight, rightHeight) + 1;
    }
    public static void main(String[] args) {
        TreeNode root1 = new TreeNode(1);
```

```
        root1.left = new TreeNode(2);
        root1.left.left = new TreeNode(3);
        root1.left.left.left = new TreeNode(4);
        System.out.println(isBalanced(root1));

        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(isBalanced(root2));
    }
}
```

Solution:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac BalancedTree.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java BalancedTree
false
true
```

6.Triplet sum in array:

Coding:

```
import java.util.Arrays;
public class Triplet {
    public static int findTriplet(int[] arr, int x) {
        Arrays.sort(arr);
        for (int i = 0; i < arr.length - 2; i++) {
            int j = i + 1;
            int k = arr.length - 1;
            while (j < k) {
                int sum = arr[i] + arr[j] + arr[k];
                if (sum == x) {
                    return 1;
                }
                if (sum < x) {
                    j++;
                }
                else {
                    k--;
                }
            }
```

```
        }
        return 0;
    }
    public static void main(String[] args) {
        int[] arr1 = {1, 4, 45, 6, 10, 8};
        int x1 = 13;
        System.out.println(findTriplet(arr1, x1));

        int[] arr2 = {1, 2, 4, 3, 6, 7};
        int x2 = 10;
        System.out.println(findTriplet(arr2, x2));

        int[] arr3 = {40, 20, 10, 3, 6, 7};
        int x3 = 24;
        System.out.println(findTriplet(arr3, x3));
    }
}
```

Output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac Triplet.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java Triplet
1
1
0
```