DSA Practice Problems

1. 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.

Solution:

```
import java.util.*;
class Kanp {
    static int knapSack(int W, int wt[], int val[], 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]] + val[i - 1]);
            }
        }
        return dp[W];
    }

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

```
        int profit[] = { 60, 100, 120 };
        int weight[] = { 10, 20, 30 };
        int W = 50;
        int n = profit.length;
        System.out.print("The capacity is :"+knapSack(W, weight, profit, n));
    }
}
```

OUTPUT:

```
  The capacity is : 220
```

Time Complexity : O(n * w)

Space Complexity : O(w)

2.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?

Solution:

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

class Floor {

        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("Floor of " + x
                                    + " doesn't exist in array ");
        else
            System.out.print("Floor of " + x + " is "
                                    + arr[index]);
    }
}
```

OUTPUT:

The floor of 7 is 6

Time Complexity : O(n).

Space Complexity : O(1).

3. 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.

Solution:

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

class Check_array {

    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<Integer, Integer>();
    int count = 0;
    for (int i = 0; i < N; i++) {
        if (map.get(arr1[i]) == null)
            map.put(arr1[i], 1);
        else {
            count = map.get(arr1[i]);
            count++;
            map.put(arr1[i], count);
        }
    }
    for (int i = 0; i < N; i++) {
        if (!map.containsKey(arr2[i]))
            return false;
        if (map.get(arr2[i]) == 0)
            return false;
        count = map.get(arr2[i]);
        --count;
        map.put(arr2[i], count);
    }
    return true;
}
public static void main(String[] args)
```

```
    {
        int arr1[] = { 3, 5, 2, 5, 2 };
        int arr2[] = { 2, 3, 5, 5, 2 };
        if (areEqual(arr1, arr2))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

OUTPUT:

Yes

Time Complexity : O(n)

Space Complexity : O(n)

4. Palindrome LinkedList : Given the head of a singly linked list, return true if it is a Palindrome or false otherwise.

Solution:

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

```
    }
}

class GfG {
    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("alse");
    }
}
```

Output :

True

Time Complexity : O(n)

Space Complexity : O(1)

5. 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

Solution:

```
import java.io.*;
import java.lang.*;
import java.util.*;
class Node {
    int key;
    Node left;
```

```
    Node right;
    Node(int k)
    {
        key = k;
        left = right = null;
    }
}
class GFG {
    public static int isBalanced(Node root)
    {
        if (root == null)
            return 0;
        int lh = isBalanced(root.left);
        if (lh == -1)
            return -1;
        int rh = isBalanced(root.right);
        if (rh == -1)
            return -1;

        if (Math.abs(lh - rh) > 1)
            return -1;
        else
            return Math.max(lh, rh) + 1;
    }

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

```
        Node root = new Node(10);
        root.left = new Node(5);
        root.right = new Node(30);
        root.right.left = new Node(15);
        root.right.right = new Node(20);

        if (isBalanced(root) > 0)
            System.out.print("Balanced");
        else
            System.out.print("Not Balanced");
    }
}
```

Output :

Balanced

Time Complexity : O(n)

Space Complexity : O(h)

6. Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0. Notice that the solution set must not contain duplicate triplets.

Solution:

```
import java.util.Arrays;
```

```
public class GfG {
    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)
    {
        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)

Space Complexity : O(1)