DSA PRACTICE PROBLEMS – SET 2

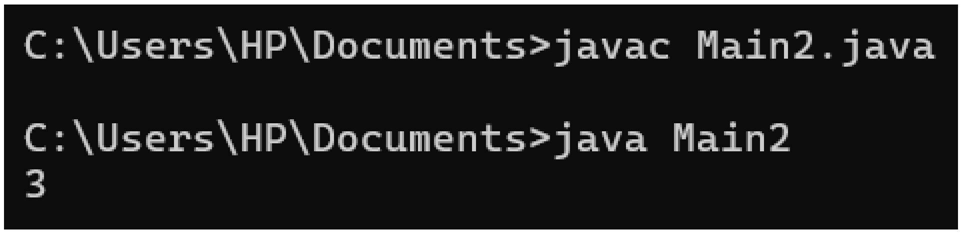
Madhulekha R – AIML - [11/11/2024]

1. **Question:** 0-1 Knapsack problem
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.

Code:

```
import java.util.*;

class Main2{
   public static void main(String args[]) {
        int profit[] = new int[] {1, 2, 3};
        int weight[] = new int[] {4, 5, 1};
        int w = 4;
        int n = profit.length;
        System.out.println(knapSack(w, weight, profit, n));
    }

   static int knapSack(int w, int wt[], int val[], int n) {
        int[][] dp = new int[n + 1][w + 1];
        for (int i = 0; i <= n; i++) {
            for (int j=0; j<=w; j++) {
                if (i==0 || j==0) {
                    dp[i][j] = 0;
                } else if (wt[i - 1] <= j) {
                    dp[i][j] = Math.max(val[i - 1] + dp[i - 1][j - wt[i - 1]], dp[i - 1][j]);
                } else {
                    dp[i][j] = dp[i - 1][j];
                }
            }
        }
        return dp[n][w];
    }
}
```

Output:



```
C:\Users\HP\Documents>javac Main2.java

C:\Users\HP\Documents>java Main2
3
```

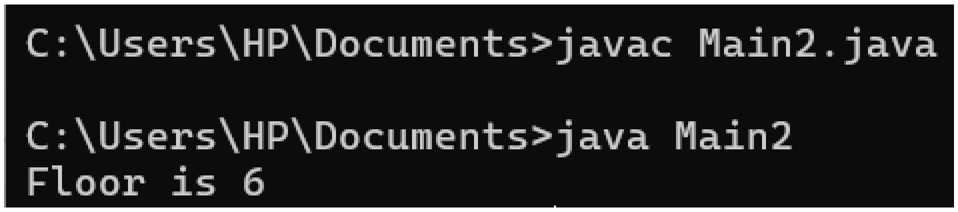
Time Complexity: O(n*w)
Space Complexity: O(n*w)

2. **Question:** Floor in sorted array
   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

   Code:

```
import java.util.*;

class Main2{
    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 doesn't exist in array ");
        else
            System.out.print("Floor is " + arr[index]);
    }
}
```

Output:



```
C:\Users\HP\Documents>javac Main2.java

C:\Users\HP\Documents>java Main2
Floor is 6
```

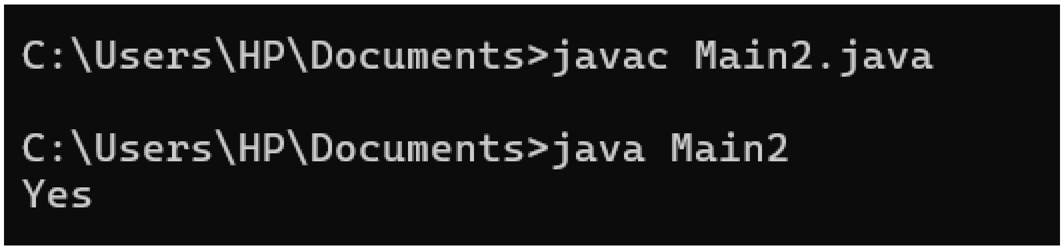
Time Complexity: O(N)
Space Complexity: O(1)

3. **Question:** Check equal arrays
Given two arrays, **arr1** and **arr2** of equal length **N**, the task is to determine if the given arrays are equal or not.

Code:

```
import java.util.*;

class Main2{
    public static boolean areEqual(int arr1[], int arr2[]){
        int n = arr1.length;
        int m = arr2.length;

        if (n!=m)
            return false;

        Arrays.sort(arr1);
        Arrays.sort(arr2);

        for (int i = 0; i < n; i++)
            if (arr1[i] != arr2[i])
                return false;
        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:


```
C:\Users\HP\Documents>javac Main2.java

C:\Users\HP\Documents>java Main2
Yes
```

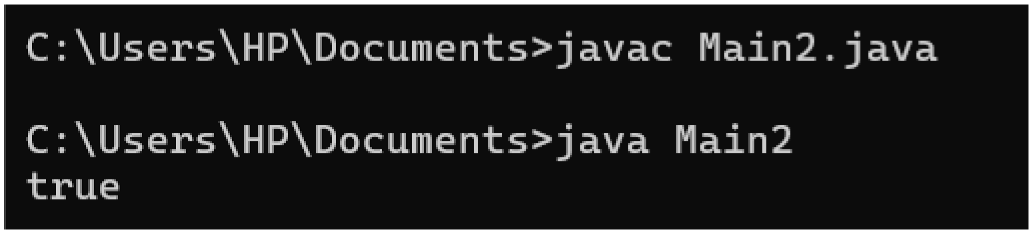
Time Complexity: O(n log n)
Space Complexity: O(n)

4. **Question:** Palindrome linked list
   Given a **singly** linked list. The task is to check if the given linked list is **palindrome** or not.

   Code:

```
import java.util.Stack;

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

class Main2{
    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");
        }
    }

    static boolean isPalindrome(Node head) {
        Node currNode = head;
        Stack<Integer> s = new Stack<>();

        while (currNode != null) {
            s.push(currNode.data);
            currNode = currNode.next;
        }
        while (head != null) {
            int c = s.pop();
            if (head.data != c) {
                return false;
            }
            head = head.next;
        }
        return true;
    }
}
```

Output:



```
C:\Users\HP\Documents>javac Main2.java

C:\Users\HP\Documents>java Main2
true
```

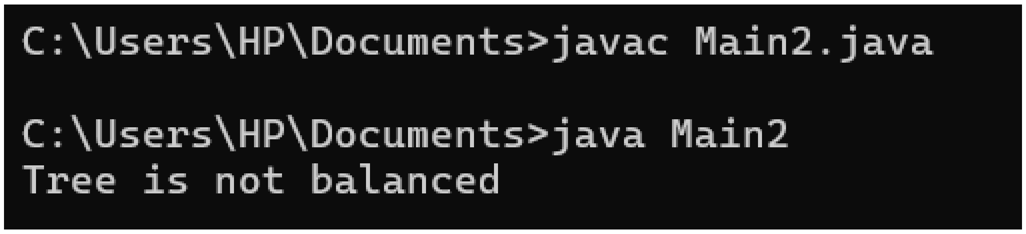
Time Complexity: O(n)
Space Complexity: O(n)

5. **Question:** Balanced tree check

   Code:

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

class Main2{
    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[])
    {
        BinaryTree tree = new BinaryTree();
        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:



```
C:\Users\HP\Documents>javac Main2.java

C:\Users\HP\Documents>java Main2
Tree is not balanced
```

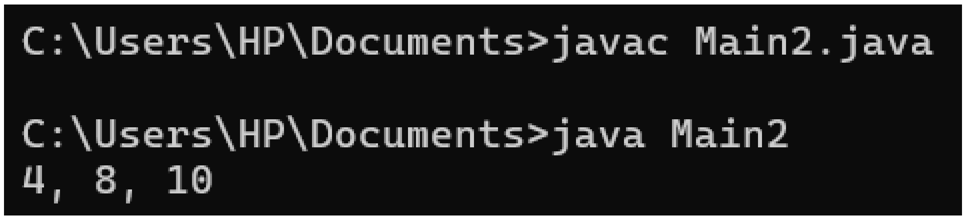
Time Complexity: O(N^2)
Space Complexity: O(N)

6. **Question:** Triplet sum in array
   Given an array **arr[]** of size **n** and an integer **sum**. Find if there's a triplet in the array which sums up to the given integer **sum**.

   Code:

```
import java.util.Arrays;

public class Main2{
    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 currsum = arr[i] + arr[l] + arr[r];
                if (currsum == sum) {
                    System.out.println(arr[i] + ", " + arr[l] + ", " + arr[r]);
                    return true;
                }
                else if (currsum < 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:



```
C:\Users\HP\Documents>javac Main2.java

C:\Users\HP\Documents>java Main2
4, 8, 10
```

Time Complexity: O(n^2)
Space Complexity: O(n)