CODING PRACTICE PROBLEM

(Sowmya A) **DATE: 11/11/2024**

1. 0/1 KnapSack Problem

```
class knap_sack {

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

        if (n == 0 || W == 0)
            return 0;

        if (wt[n - 1] > W)
            return knapSack(W, wt, val, n - 1);

        else
            return Math.max(knapSack(W, wt, val, n - 1),
             val[n - 1] + knapSack(W - wt[n-1], wt, val, n-1));
    }

    public static void main(String args[])
    {
        int profit[] = new int[] { 60, 100, 120 };
        int weight[] = new int[] { 10, 20, 30 };
        int W = 50;
        int n = profit.length;
        System.out.println(knapSack(W, weight, profit, n));
    }
}
```

O/P:

```
D:\>javac knapsack.java

D:\>java knap_sack
220
```

Time Complexity: O(2^N)

2. Floor in a sorted array

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

class floorsorted {
        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]);
        }
}
```

O/P:

```
D:\>javac floor.java

D:\>java floorsorted
Floor of 7 is 6
```

Time Complexity: O(N)

3. Check if two arrays are equal

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

class equalarray {

    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");
    }
}
```

O/P:

```
D:\>javac equalarray.java

D:\>java equalarray.java
Yes
```

Time Complexity: O(N*log(N))

4. Palindrome Linked List

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

class linkedlistpalindrome {

    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("false");
    }
}
```

O/P:

```
D:\>javac linkedlistpalindrome.java

D:\>java linkedlistpalindrome
true
```

Time Complexity: O(N)

5. Balanced Binary Tree Or Not

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

class BinaryTree {
    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");
   }
}
```

O/P:

```
D:\>javac BinaryTreeBalance.java

D:\>java BinaryTree
Tree is not balanced
```

Time Complexity: O(N^2)

6. Triple sum in a array

```java
import java.util.Arrays;

public class triplesum {
    static boolean find3Numbers(int[] arr, int sum)
    {
        int n = arr.length;

        for (int i = 0; i < n - 2; i++) {

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

                for (int k = j + 1; k < n; k++) {

                    if (arr[i] + arr[j] + arr[k] == sum) {

                        System.out.println(
                            "Triplet is " + arr[i] + ", "
                            + arr[j] + ", " + arr[k]);
                        return true;
                    }
                }
            }
        }
        return false;
    }


    public static void main(String[] args)
    {
        int[] arr = { 1, 4, 45, 6, 10, 8 };
        int sum = 22;

        find3Numbers(arr, sum);
    }
}
```

O/P:

```
D:\>javac triplesum.java

D:\>java triplesum
Triplet is 4, 10, 8
```

Time Complexity: O(N*3)