Coding practice Problems(11/11/2024)

Q 1) 0-1 Knapsack Problem

```
import java.util.*;

class Knapsack_Problem {

   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));

   }

}
```

Time Complexity: O(2^n)

Output:

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                    powershell

PS C:\sde program> javac Knapsack_Problem.java
PS C:\sde program> java Knapsack_Problem
220
PS C:\sde program>
```

Q 2) Floor in sorted array

```
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]);
        }
}
```

Time Complexity: O(n)

Output:

Q 3) Check equal arrays

```
import java.io.*;
import java.util.*;
class check_equal {
   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");
    }
}
```

Time Complexity: O(n)

Output:

Q 4) Palindrome linked list

```
class Node {
    int data;
    Node next;
    Node(int d) {
        data = d;
        next = null;
    }
}
class palindrome {
    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) {
     // Linked list : 1->2->3->2->1
        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");
    }
}
```

Time Complexity: O(n)

Output:

Q 5) Balanced tree check

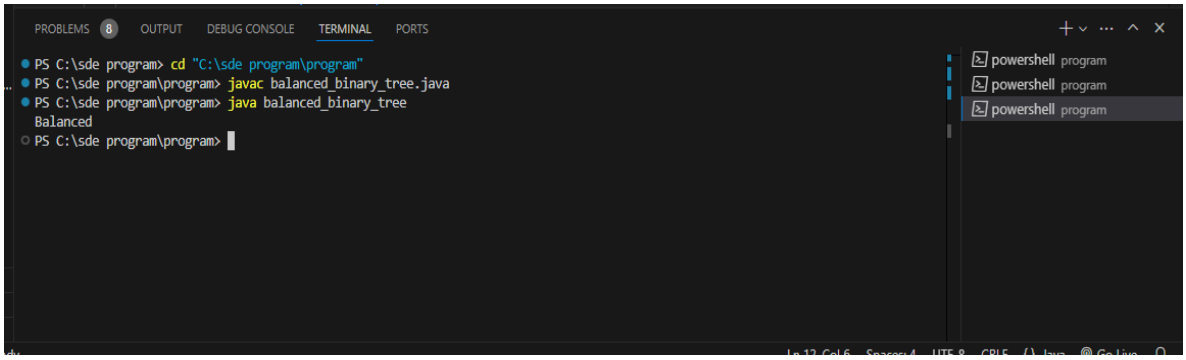
```
import java.io.*;
import java.lang.*;
import java.util.*;
```

```
class Node {
   int key;
   Node left;
   Node right;
   Node(int k)
   {
      key = k;
      left = right = null;
   }
}
class balanced_binary_tree {
   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");
    }
}
```

Time Complexity: O(n)

Output:



Q 6) Triplet sum in array

```
import java.util.Arrays;
public class triplet_sum {
    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);
}}
```

Time Complexity: O(n^2)

Output: