Date: 11-11-2024

CODING PRACTICE PROBLEMS – DAY 2

1. 0-1 knapsack problem

```java
class Knapsack {
    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));
    }
}
```

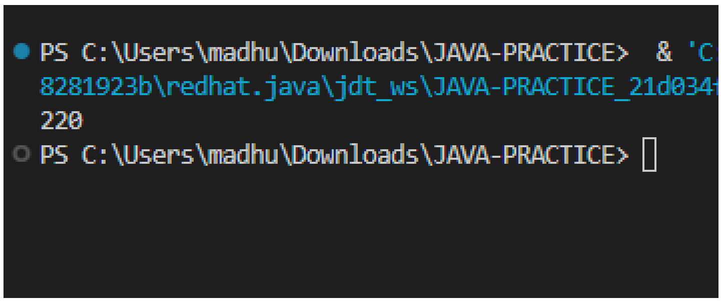
Time Complexity: O(2^N)

Reason: Each item results in two choices: include or exclude.

This leads to 2^n possible subsets of items, making the time complexity exponential.

This exponential growth is typical for naive recursive solutions to combinatorial problems

OUTPUT:



```
● PS C:\Users\madhu\Downloads\JAVA-PRACTICE>  & 'C
  8281923b\redhat.java\jdt_ws\JAVA-PRACTICE_21d034
  220
○ PS C:\Users\madhu\Downloads\JAVA-PRACTICE> 
```

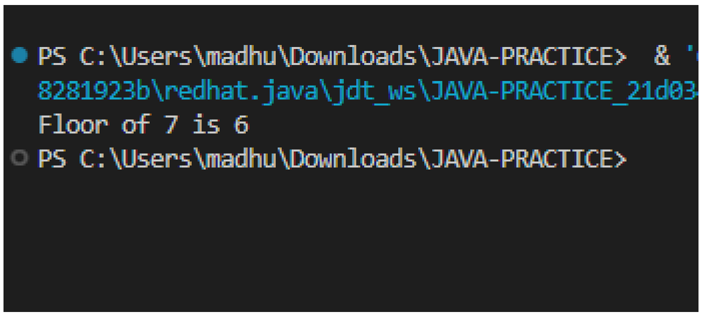
2. Floor in sorted array

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

class SortedArray {
    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)

Reason: The function uses a linear search to find the first element greater than x

OUTPUT:



```
PS C:\Users\madhu\Downloads\JAVA-PRACTICE>  & '
8281923b\redhat.java\jdt_ws\JAVA-PRACTICE_21d03
Floor of 7 is 6
PS C:\Users\madhu\Downloads\JAVA-PRACTICE>
```

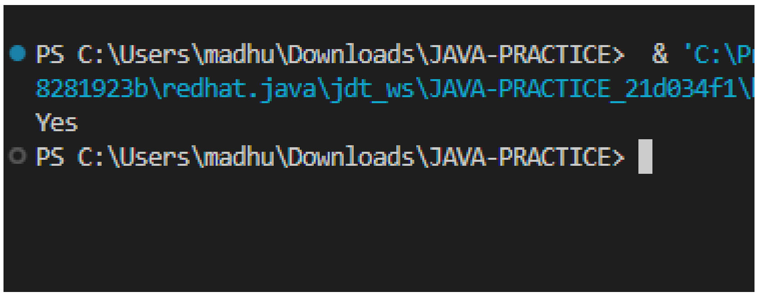
3. Check equal arrays

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

class CheckEqual {
    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 log(N))

Reason: Comparing the elements linearly takes O(N)O(N) time. Therefore, the total time complexity is O(NlogN + N)

OUTPUT:



```
PS C:\Users\madhu\Downloads\JAVA-PRACTICE>  & 'C:\P
8281923b\redhat.java\jdt_ws\JAVA-PRACTICE_21d034f1\
Yes
PS C:\Users\madhu\Downloads\JAVA-PRACTICE>
```

4. Palindrome linked list

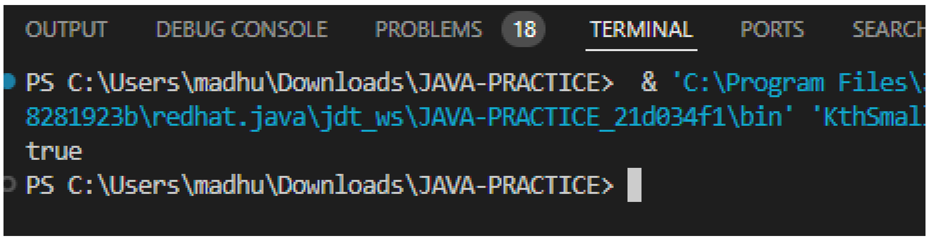
```java
import java.util.Stack;

class Node {
    int data;
    Node next;

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

class KthSmallest {
    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;
    }

    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");
    }
}
```

Time Complexity: O(N)

Reason: First loop: O(n), Second loop: O(n)

Combining these, the total time complexity is **O(n + n) = O(n)**.

OUTPUT:



```
PS C:\Users\madhu\Downloads\JAVA-PRACTICE>  & 'C:\Program Files\
8281923b\redhat.java\jdt_ws\JAVA-PRACTICE_21d034f1\bin' 'KthSmal
true
PS C:\Users\madhu\Downloads\JAVA-PRACTICE> 
```

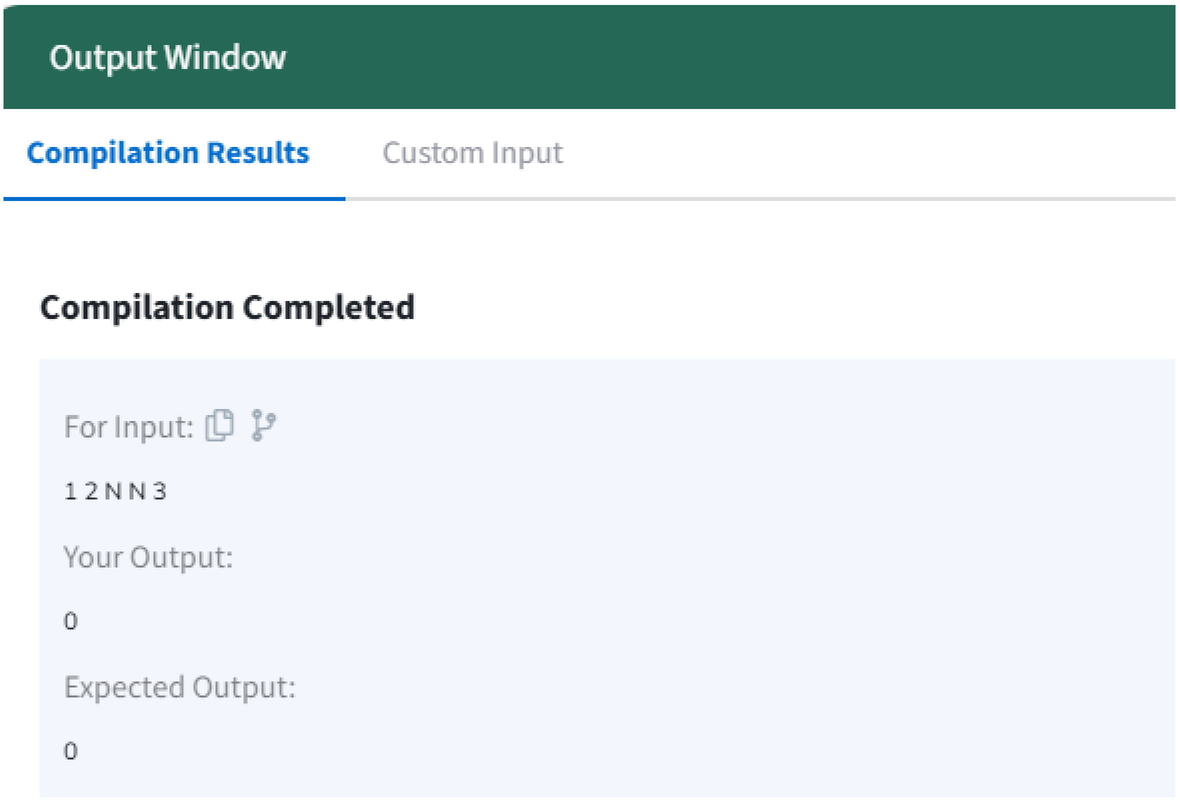
5. Balanced tree check

```java
class Tree
{
    boolean isBalanced(Node root)
    {
        if(root==null)
            return true;
           int lh=h(root.left);
           int rh=h(root.right);
           return (Math.abs(lh-rh)<=1 && isBalanced(root.left) &&
isBalanced(root.right));
    }
    public static int h(Node root)
    {
        if(root==null)
            return 0;
        int max1=1+h(root.left);
        int max2=1+h(root.right);
        return Math.max(max1,max2);
    }
}
```

Time Complexity: O(N^2)

Reason: The time complexity of the isBalanced function for each node is O(n) (due to h calls) multiplied by O(n) nodes

OUTPUT:



Compilation Completed

```
For Input:
1 2 N N 3
Your Output:
0
Expected Output:
0
```

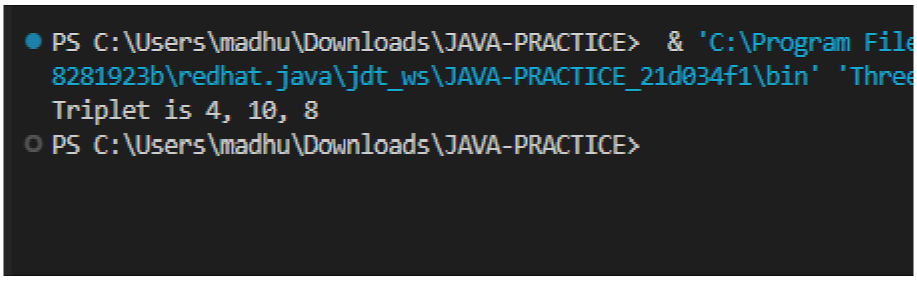
6. Triplet sum in array

```java
import java.util.Arrays;

public class ThreeSum {
    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);
    }
}
```

Time Complexity: O(N^3)

Reason: There are three nested loops traversing the array.

OUTPUT:



```
● PS C:\Users\madhu\Downloads\JAVA-PRACTICE>  & 'C:\Program File
  8281923b\redhat.java\jdt_ws\JAVA-PRACTICE_21d034f1\bin'  'Three
  Triplet is 4, 10, 8
○ PS C:\Users\madhu\Downloads\JAVA-PRACTICE>
```