

Syed Nadeem G

Information Technology

22IT116

Practise Day 1 (09 – 11 – 2024)

Problems(medium)

1. Maximum Subarray Sum – Kadane"s Algorithm:

Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.

Input: arr[] = {2, 3, -8, 7, -1, 2, 3} Output: 11

Program:

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

```
class Problem1{
```

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

```
    {
```

```
        Scanner sc=new Scanner(System.in);
```

```
        System.out.print("Enter the size of array : ");
```

```
        int n=sc.nextInt();
```

```
        int[] nums=new int[n];
```

```
        System.out.print("Enter the array : ");
```

```
        for(int i=0;i<n;i++)
```

```
        {
```

```
            nums[i]=sc.nextInt();
```

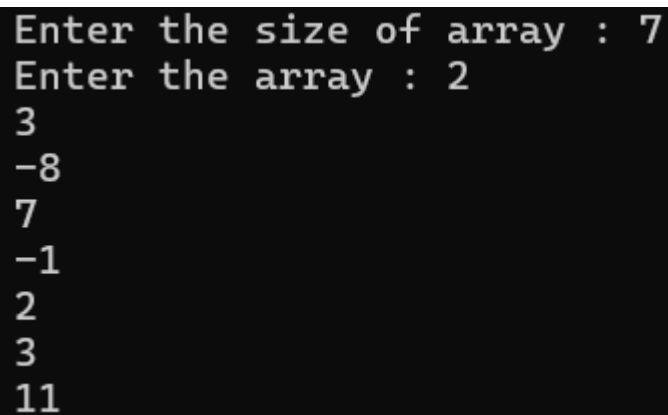
```
        }
```

```

int max_sum = nums[0];
int max = 0;
for (int i : nums) {
    max += i;
    if (max > max_sum) {
        max_sum = max;
    }
    if (max < 0) {
        max = 0;
    }
}
System.out.println(max_sum);
}

```

Output:



```

Enter the size of array : 7
Enter the array : 2
3
-8
7
-1
2
3
11

```

Time Complexity: $O(n)$

2. Maximum Product Subarray

Given an integer array, the task is to find the maximum product of any subarray.

Input: `arr[] = {-2, 6, -3, -10, 0, 2}`

Output: 180

Explanation: The subarray with maximum product is {6, -3, -10} with product = $6 * (-3) * (-10) = 180$

Program:

```
import java.util.Scanner;
```

```
public class Solution{  
    public static void main(String[] args) {  
        Scanner sc=new Scanner(System.in);  
        System.err.println("Enter the size of array : ");  
        int n=sc.nextInt();  
        int[] arr = new int[n];  
        System.out.println("Enter the array: ");  
        for(int i=0;i<n;i++){  
            arr[i]=sc.nextInt();  
        }  
        int pre = 1, suff = 1;  
        int ans = Integer.MIN_VALUE;  
        for (int i = 0; i < n; i++) {  
            if (pre == 0)  
            {  
                pre = 1;  
            }  
            if (suff == 0)  
            {
```

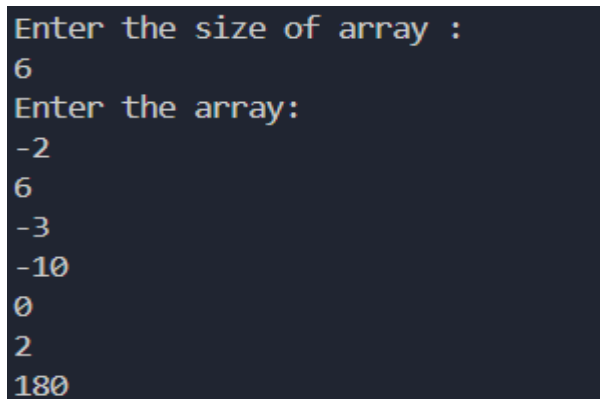
```

        suff = 1;
    }
    pre *= arr[i];
    suff *= arr[n - i - 1];
    ans = Math.max(ans, Math.max(pre, suff));
}
System.out.println(ans);
}

}

```

Output:



```

Enter the size of array :
6
Enter the array:
-2
6
-3
-10
0
2
180

```

Time Complexity: $O(n)$

3. Search in a sorted and rotated Array

Given a sorted and rotated array `arr[]` of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1.

Input : `arr[] = {4, 5, 6, 7, 0, 1, 2}`, `key = 0`

Output : 4

Program:

```
import java.util.Scanner;

class Solution {

    public static int search(int[] arr, int k, int n) {

        int low = 0, high = n - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            if (arr[mid] == k) {

                return mid;

            }

            if (arr[low] <= arr[mid]) {

                if (arr[low] <= k && k <= arr[mid]) {

                    high = mid - 1;

                } else {

                    low = mid + 1;

                }

            } else {

                if (arr[mid] <= k && k <= arr[high]) {

                    low = mid + 1;

                } else {

                    high = mid - 1;

                }

            }

        }

        return -1;

    }

}
```

```
}

public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
    System.out.println("enter the size: ");
    int n=sc.nextInt();
    System.out.println("enter target element: ");
    int k=sc.nextInt();
    int[] arr=new int[n];
    System.out.println("enter the elements of the array: ");
    for(int i=0;i<n;i++)
    {
        arr[i]=sc.nextInt();
    }
    int ans=search(arr,n,k);
    System.out.println(ans);

}
}
```

Output:

```
enter the size:
5
enter target element
0
enter the elements of the array
3 4 0 1 2
2
```

Time Complexity: $O(n)$

4. Container with Most Water

Input: arr = [1, 5, 4, 3]

Output: 6

Explanation: 5 and 3 are distance 2 apart. So the size of the base = 2. Height of container = $\min(5, 3) = 3$. So total area = $3 * 2 = 6$

Program:

```
import java.util.Scanner;
```

```
public class Problem4{
```

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

```
        Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Enter the size of array : ");
```

```
        int n = sc.nextInt();
```

```
        int[] height = new int[n];
```

```
        System.out.println("Enter the array : ");
```

```
        for(int i = 0; i < n; i++)
```

```
        {
```

```
            height[i]=sc.nextInt();
```

```
        }
```

```
        int l = 0;
```

```
        int r = height.length - 1;
```

```
        int area = 0;
```

```
        int ans = 0;
```

```
        while (l < r) {
```

```

        area = (r - l) * Math.min(height[r], height[l]);
        ans = Math.max(ans, area);
        if (height[l] < height[r]) l++;
        else r--;
    }
    System.out.println(ans);
}
}

```

Output:

```

Enter the size of array :
4
Enter the array :
1 5 4 3
6

```

Time Complexity: $O(n)$

5. Find the Factorial of a large number

Input: 100

Output:

```

933262154439441526816992388562667004907159682643816214685929638
952175999932299
156089414639761565182862536979208272237582511852109168640000000
0000000000000000 00

```

Program:

```

import java.math.BigInteger;
import java.util.Scanner;

```



```

public class LargeFactorial {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();
        System.out.println("Factorial of " + n + " is: " + factorial(n));
    }

    public static BigInteger factorial(int n) {
        BigInteger result = BigInteger.ONE;
        for (int i = 2; i <= n; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }
        return result;
    }
}

```

Output:



```

Enter a number: 100
Factorial of 100 is: 9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828625369792082722375825118521091686400000
000000000000000000

```

Time Complexity: $O(n)$

6. Trapping Rainwater Problem states that given an array of n non-negative integers `arr[]` representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain. Input: `arr[] = {3, 0, 1, 0, 4, 0, 2}`

Output: 10

Explanation: The expected rainwater to be trapped is shown in the above image.

Program:

```
import java.util.Scanner;

public class Problem6
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the array size : ");
        int n=sc.nextInt();
        System.out.println("Enter the array : ");
        int[] height=new int[n];
        for(int i=0;i<n;i++)
        {
            height[i]=sc.nextInt();
        }
        int n = height.length;
        int[] prefix = new int[n];
        int[] suffix = new int[n];

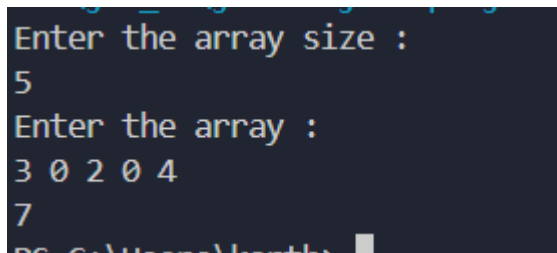
        prefix[0] = height[0];
        for (int i = 1; i < n; i++) {
            prefix[i] = Math.max(height[i], prefix[i - 1]);
        }
```

```

        suffix[n - 1] = height[n - 1];
        for (int i = n - 2; i >= 0; i--) {
            suffix[i] = Math.max(height[i], suffix[i + 1]);
        }
        int sum = 0;
        for (int i = 1; i < n - 1; i++) {
            sum += Math.min(prefix[i], suffix[i]) - height[i];
        }
        System.out.println(sum);
    }
}

```

Output:



```

Enter the array size :
5
Enter the array :
3 0 2 0 4

```

Time Complexity: $O(n)$

7. Chocolate Distribution Problem

Given an array `arr[]` of n integers where `arr[i]` represents the number of chocolates in i th packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized. Input: `arr[] = {7, 3, 2, 4, 9, 12, 56}`, $m = 3$

Output: 2

Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

Program:

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

class Problem7 {
    public static int findMinDifference(int[] arr, int m) {
        // Edge case: If there are fewer packets than students, return -1
        if (m == 0 || arr.length < m) {
            return -1;
        }
        // Sort the array to facilitate finding the smallest difference
        Arrays.sort(arr);
        // Initialize the minimum difference as a large value
        int minDiff = Integer.MAX_VALUE;

        // Use a sliding window to find the minimum difference in each subarray
        // of size m
        for (int i = 0; i <= arr.length - m; i++) {
            int diff = arr[i + m - 1] - arr[i];
            minDiff = Math.min(minDiff, diff);
        }
        return minDiff;
    }

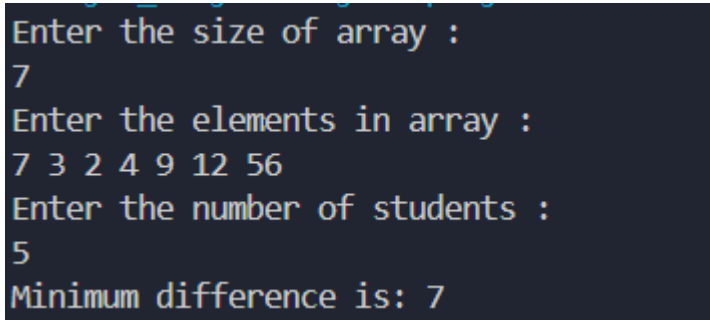
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```

        System.out.println("Enter the size of array : ");
        int n=sc.nextInt();
        System.out.println("Enter the elements in array : ");
        int[] arr=new int[n];
        for(int i=0;i<n;i++)
        {
            arr[i]=sc.nextInt();
        }
        System.out.println("Enter the number of students : ");
        int m=sc.nextInt();
        int result = findMinDifference(arr, m);
        System.out.println("Minimum difference is: " + result);
    }
}

```

Output:



```

Enter the size of array :
7
Enter the elements in array :
7 3 2 4 9 12 56
Enter the number of students :
5
Minimum difference is: 7

```

Time Complexity: $O(n \log n)$

8.MERGE INTERVAL

Given an array of intervals where $\text{intervals}[i] = [\text{start}_i, \text{end}_i]$, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input.*

Input: $\text{arr} = [[1, 3], [2, 4], [6, 8], [9, 10]]$

Output: [[1, 4], [6, 8], [9, 10]]

Explanation: In the given intervals, we have only two overlapping intervals [1, 3] and [2, 4]. Therefore, we will merge these two and return [[1, 4], [6, 8], [9, 10]].

Input: arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]]

Output: [[1, 6], [7, 8]]

Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval [1, 6].

Program:

```
import java.util.Arrays;
import java.util.ArrayList;
import java.util.List;

public class Merge {
    public int[][] merge(int[][] intervals) {

        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

        List<int[]> merged = new ArrayList<>();
        int[] prev = intervals[0];

        for (int i = 1; i < intervals.length; i++) {
            int[] interval = intervals[i];
            if (interval[0] <= prev[1]) {
                prev[1] = Math.max(prev[1], interval[1]);
            } else {
                merged.add(prev);
                prev = interval;
            }
        }
        merged.add(prev);
    }
}
```

```

        // Convert list to array and return
        return merged.toArray(new int[merged.size()][]);
    }

    public static void main(String[] args) {

        int[][] intervals1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        int[][] intervals2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};

        Merge solution = new Merge();

        System.out.println("Input: [[1, 3], [2, 4], [6, 8], [9, 10]]");
        int[][] mergedIntervals1 = solution.merge(intervals1);
        System.out.println("Merged Intervals:");
        for (int[] interval : mergedIntervals1) {
            System.out.println(Arrays.toString(interval));
        }

        System.out.println();

        System.out.println("Input: [[7, 8], [1, 5], [2, 4], [4, 6]]");
        int[][] mergedIntervals2 = solution.merge(intervals2);
        System.out.println("Merged Intervals:");
        for (int[] interval : mergedIntervals2) {
            System.out.println(Arrays.toString(interval));
        }
    }
}

```

Output:

```
Enter the number of intervals: 4
Enter each interval in the format [start end]:
7 8
1 5
2 4
4 6
Merged Intervals:
[1, 6]
[7, 8]
```

Time Complexity: $O(n \log n)$

9. A Boolean Matrix Question

Given a boolean matrix `mat[M][N]` of size $M \times N$, modify it such that if a matrix cell `mat[i][j]` is 1 (or true) then make all the cells of *i*th row and *j*th column as 1.

Input: `{{1, 0}, {0, 0}}`

Output: `{{1, 1} {1, 0}}`

Program:

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

```
public class Problem9{
```

```
    public static void modifyMatrix(int[][] matrix) {
```

```
        int numberOfRows = matrix.length;
```

```
        int numberOfCols = matrix[0].length;
```

```
        int[] rows = new int[numberOfRows];
```

```
        int[] cols = new int[numberOfCols];
```

```
        for (int i = 0; i < numberOfRows; i++) {
```

```
            for (int j = 0; j < numberOfCols; j++) {
```



```
        if (matrix[i][j] == 1) {
            rows[i] = 1;
            cols[j] = 1;
        }
    }
}

for (int i = 0; i < numberOfRows; i++) {
    for (int j = 0; j < numberOfCols; j++) {
        if (rows[i] == 1 || cols[j] == 1) {
            matrix[i][j] = 1;
        }
    }
}
}
```

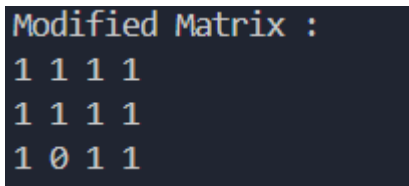
```
public static void printMatrix(int[][] matrix) {
    for (int i = 0; i < matrix.length; i++) {
        for (int j = 0; j < matrix[0].length; j++) {
            System.out.print(matrix[i][j] + " ");
        }
        System.out.println();
    }
}
```

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

```
{1, 0, 0, 1},  
{0, 0, 1, 0},  
{0, 0, 0, 0},  
};
```

```
System.out.println("Modified Matrix :");  
modifyMatrix(matrix);  
printMatrix(matrix);  
}  
}
```

Output:



```
Modified Matrix :  
1 1 1 1  
1 1 1 1  
1 0 1 1  
1 1 1 1
```

Time Complexity: $O(m \times n)$

10. Print a given matrix in spiral form

Given an $m \times n$ matrix, the task is to print all elements of the matrix in spiral form.

Input: matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16 }} Output:
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Program:

```
import java.util.ArrayList;  
import java.util.List;
```

```
public class Spiral {

    public static List<Integer> printSpiral(int[][] mat) {

        List<Integer> ans = new ArrayList<>();

        int n = mat.length;
        int m = mat[0].length;

        int top = 0, left = 0, bottom = n - 1, right = m - 1;

        while (top <= bottom && left <= right) {

            for (int i = left; i <= right; i++)
                ans.add(mat[top][i]);

            top++;

            for (int i = top; i <= bottom; i++)
                ans.add(mat[i][right]);

            right--;

            if (top <= bottom) {
                for (int i = right; i >= left; i--)
```

```
        ans.add(mat[bottom][i]);

        bottom--;
    }

    if (left <= right) {
        for (int i = bottom; i >= top; i--)
            ans.add(mat[i][left]);

        left++;
    }
}

return ans;
}
```

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

```
    int[][] mat = {{1, 2, 3, 4},
                   {5, 6, 7, 8},
                   {9, 10, 11, 12},
                   {13, 14, 15, 16}};
```

```
    List<Integer> ans = printSpiral(mat);
```

```
    for(int i = 0; i < ans.size(); i++) {
        System.out.print(ans.get(i) + " ");
    }
```

```

    }

    System.out.println();
}
}

```

Output:

```

1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

```

Time Complexity: $O(n \times m)$

13. Check if given Parentheses expression is balanced or not

Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not.

Input: str = “((()))()()”

Output: Balanced

Program:

```

import java.util.Scanner;
import java.util.Stack;

public class Problem13 {

    public static String isBalanced(String s) {

```

```

Stack<Character> stack = new Stack<>();

for (int i = 0; i < s.length(); i++) {
    char ch = s.charAt(i);

    if (ch == '(') {
        stack.push(ch);
    } else if (ch == ')') {
        if (stack.isEmpty()) {
            return "Not Balanced";
        }
        stack.pop();
    }
}

return stack.isEmpty() ? "Balanced" : "Not Balanced";
}

public static void main(String[] args) {
    Scanner sc=new Scanner(system.in);
    System.out.println("Enter the string paranthesis : ");
    String s=sc.next();
    System.out.println("Input: " + str1 + " - Output: " + isBalanced(s));
}
}

```

Output:

```
Input: (((()))()) - Output: Balanced
```

Time Complexity: $O(n)$

14. Check if two Strings are Anagrams of each other

Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

Input: s1 = "geeks" s2 = "kseeg"

Output: true

Explanation: Both the strings have the same characters with the same frequency. So, they are anagrams.

Program:

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

```
public class Anagram {
```

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

```
        Scanner sc = new Scanner(System.in);
```

```
System.out.println("Enter the first string: ");
String a = sc.nextLine();

System.out.println("Enter the second string: ");
String b = sc.nextLine();

if (a.length() != b.length()) {
    System.out.println("The strings are not anagrams.");
    return;
}

Map<Character, Integer> map = new HashMap<>();

for (int i = 0; i < a.length(); i++) {
    map.put(a.charAt(i), map.getDefault(a.charAt(i), 0) + 1);
}

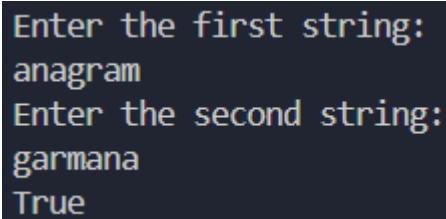
for (int i = 0; i < b.length(); i++) {
    if (!map.containsKey(b.charAt(i)) || map.get(b.charAt(i)) == 0) {
        System.out.println("The strings are not anagrams.");
        return;
    }
    map.put(b.charAt(i), map.get(b.charAt(i)) - 1);
}

System.out.println("The strings are anagrams.");
```



```
}  
}
```

Output:



```
Enter the first string:  
anagram  
Enter the second string:  
garmana  
True
```

Time Complexity: $O(n)$

15. Longest Palindromic Substring

Given a string `str`, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

Input: `str = "forgeeksskeegfor"`

Output: "geeksskeeg" Explanation: There are several possible palindromic substrings like "kssk", "ss", "eeksskee" etc. But the substring "geeksskeeg" is the longest among all.

Program:

```
import java.util.Scanner;  
  
public class Solution {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Enter a string: ");
```

```
String s = sc.nextLine();

String res = "";
int resLen = 0;
for (int i = 0; i < s.length(); i++) {
    int l = i, r = i;
    while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
        if ((r - l + 1) > resLen) {
            res = s.substring(l, r + 1);
            resLen = r - l + 1;
        }
        l--;
        r++;
    }
    l = i; r = i + 1;
    while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
        if ((r - l + 1) > resLen) {
            res = s.substring(l, r + 1);
            resLen = r - l + 1;
        }
        l--;
        r++;
    }
}

System.out.println("Longest Palindromic Substring: " + res);
```

```
        sc.close();
    }
}
```

Output:

```
Enter a string:
geeksskeeg
Longest Palindromic Substring: geeksskeeg
```

Time Complexity: $O(n)$

16. Longest Common Prefix using Sorting

Given an array of strings `arr[]`. The task is to return the longest common prefix among each and every strings present in the array. If there's no prefix common in all the strings, return "-1".

Input: `arr[] = ["geeksforgeeks", "geeks", "geek", "geezer"]`

Output: gee Explanation: "gee" is the longest common prefix in all the given strings.

Program:

```
import java.util.*;

public class Problem16 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number of Strings : ");
```

```

int n=sc.nextInt();
String[] arr=new String[n];
System.out.println("Enter the Strings : ");
for(int i=0;i<n;i++)
{
    arr[i]=sc.next();
}
if (arr == null || arr.length == 0) {
    System.out.println(-1);
}
Arrays.sort(arr);
String first = arr[0];
String last = arr[arr.length - 1];
StringBuilder commonPrefix = new StringBuilder();
for (int i = 0; i < Math.min(first.length(), last.length()); i++) {
    if (first.charAt(i) == last.charAt(i)) {
        commonPrefix.append(first.charAt(i));
    } else {
        break;
    }
}
System.out.println(commonPrefix.length() > 0 ? commonPrefix.toString() :
"-1");
}
}

```

Output:

```
Enter the number of Strings :  
3  
Enter the Strings :  
flower  
flight  
flow  
fl
```

Time Complexity: $O(n \log n + m)$

17. Delete middle element of a stack

Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

Input : Stack[] = [1, 2, 3, 4, 5]

Output : Stack[] = [1, 2, 4, 5]

Program:

```
import java.util.Stack;  
import java.util.Scanner;  
  
public class DeleteMid{  
    public static void MidDel(Stack<Integer> stk, int cur, int mid) {  
        if (cur == mid) {  
            stk.pop();  
            return;  
        }  
  
        int top = stk.pop();
```

```

        MidDel(st, cur + 1, mid);
        st.push(top);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Stack<Integer> st = new Stack<>();

        System.out.print("Enter the number of elements in the stack: ");
        int n = sc.nextInt();

        System.out.println("Enter the elements of the stack:");
        for (int i = 0; i < n; i++) {
            int element = sc.nextInt();
            st.push(element);
        }
        System.out.println("Original Stack: " + st);
        int mid = n / 2;
        MidDel(st, 0, mid);
        System.out.println("Stack after deleting middle element: " + st);
    }
}

```

Output:

```
Enter the number of elements in the stack: 6
Enter the elements of the stack:
1 2 3 4 5 6
Original Stack: [1, 2, 3, 4, 5, 6]
Stack after deleting middle element: [1, 2, 4, 5, 6]
```

Time Complexity: $O(n)$

18. Next Greater Element (NGE) for every element in given Array

Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exists, consider the next greater element as -1.

Input: `arr[] = [4 , 5 , 2 , 25]`

Output: 4 → 5

5 → 25

2 → 25

25 → -1

Explanation: Except 25 every element has an element greater than them present on the right side

Program:

```
import java.util.Stack;

import java.util.*;

public class Problem18 {

    public static void printNGE(int[] arr) {

        int n = arr.length;

        int[] nge = new int[n];
```

```
Stack<Integer> stack = new Stack<>();
```

```
for (int i = n - 1; i >= 0; i--) {  
    while (!stack.isEmpty() && stack.peek() <= arr[i]) {  
        stack.pop();  
    }  
    nge[i] = stack.isEmpty() ? -1 : stack.peek();  
    stack.push(arr[i]);  
}
```

```
for (int i = 0; i < n; i++) {  
    System.out.println(arr[i] + " -> " + nge[i]);  
}  
}
```

```
public static void main(String[] args) {  
    Scanner sc=new Scanner(System.in);  
    System.out.println("Enter the size of the array : ");  
    int n=sc.nextInt();  
    int[] arr = new int[n];  
    System.out.println("Enter the elements of the array : ");  
    for (int i = 0; i < n; i++)  
    {  
        arr[i]=sc.nextInt();  
    }  
    printNGE(arr);  
}
```



```
}  
}
```

Output:

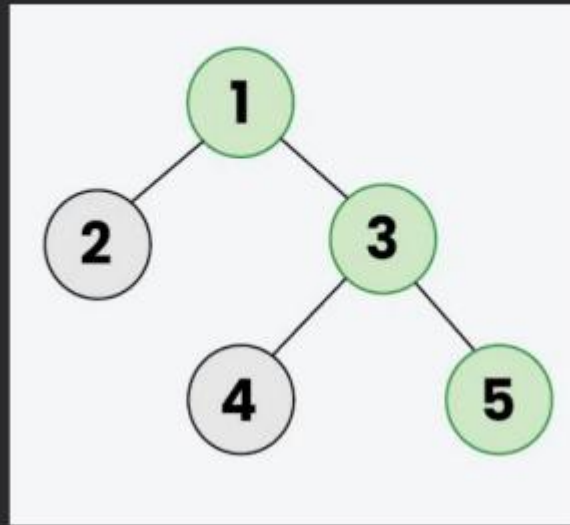
```
Enter the size of the array :  
5  
Enter the elements of the array :  
3 6 1 8 10  
3 -> 6  
6 -> 8  
1 -> 8  
8 -> 10  
10 -> -1
```

Time Complexity: $O(n)$

19. Print Right View of a Binary Tree

Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.

*Example 1: The **Green** colored nodes (1, 3, 5) represents the Right view in the below Binary tree.*



Program:

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

```
class TreeNode {
```

```
    int val;
```

```
    TreeNode left;
```

```
    TreeNode right;
```

```
    TreeNode(int x) {
```

```
        val = x;
```

```
        left = null;
```

```
        right = null;
```

```
    }
```

```
}
```

```
class Problem19 {  
    public List<Integer> rightSideView(TreeNode root) {  
        List<Integer> result = new ArrayList<Integer>();  
        rightView(root, result, 0);  
        return result;  
    }  
  
    public void rightView(TreeNode curr, List<Integer> result, int currDepth)  
    {  
        if(curr == null) {  
            return;  
        }  
        if(currDepth == result.size()) {  
            result.add(curr.val);  
        }  
        rightView(curr.right, result, currDepth + 1);  
        rightView(curr.left, result, currDepth + 1);  
    }  
  
    public static void main(String[] args)  
    {  
        TreeNode root = new TreeNode(1);  
        root.left = new TreeNode(2);  
        root.right = new TreeNode(3);  
        root.left.left = new TreeNode(4);  
    }  
}
```

```

root.left.right = new TreeNode(5);
root.right.right = new TreeNode(6);
root.left.left.left = new TreeNode(7);
Problem19 solution = new Problem19();
List<Integer> rightViewList = solution.rightSideView(root);
System.out.println("Right View of the Binary Tree:");
for (Integer val : rightViewList)
{
    System.out.print(val + " ");
}
}

```

Output:

```

Right View of the Binary Tree:
1 3 6 7

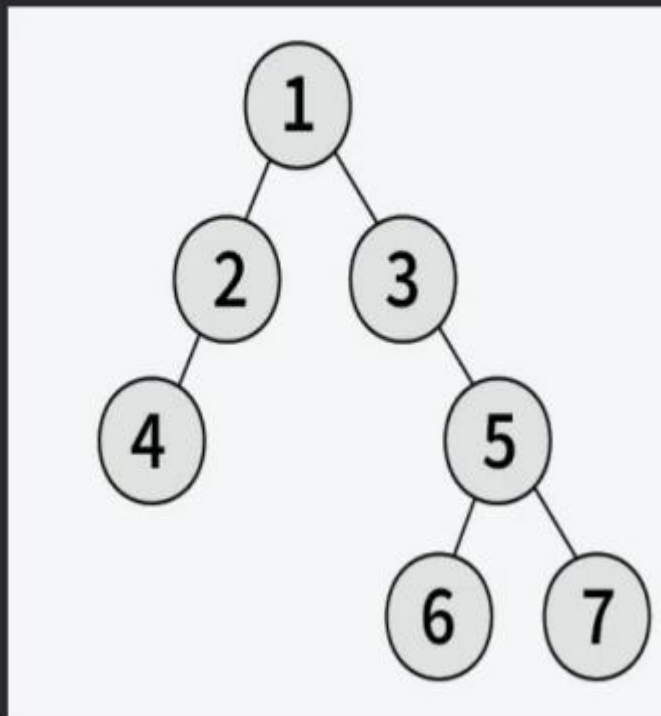
```

Time Complexity: $O(n)$

20. Maximum Depth or Height of Binary Tree

Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.

Example 2: The height of the below binary tree is 4



Program:

```
class TreeNode {  
    int val;  
    TreeNode left;  
    TreeNode right;  
  
    TreeNode(int x) {  
        val = x;  
        left = null;  
        right = null;  
    }  
}
```

```
class Problem20 {  
    public int maxDepth(TreeNode root) {  
        if (root == null) {  
            return 0;  
        }  
        int lh = maxDepth(root.left);  
        int rh = maxDepth(root.right);  
        return 1 + Math.max(lh, rh);  
    }  
  
    public static void main(String[] args) {  
        TreeNode root = new TreeNode(1);  
        root.left = new TreeNode(2);  
        root.right = new TreeNode(3);  
        root.left.left = new TreeNode(4);  
        root.left.right = new TreeNode(5);  
        root.right.right = new TreeNode(6);  
        root.left.left.left = new TreeNode(7);  
  
        Problem20 solution = new Problem20();  
        int maxDepth = solution.maxDepth(root);  
  
        System.out.println("Maximum Depth of the Binary Tree: " + maxDepth);  
    }  
}
```

Output:

```
Maximum Depth of the Binary Tree: 4
```

Time Complexity: $O(n)$