

Coding practice Problems (09.11.2024)

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

Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.

Input: arr[] = {-2, -4}

Output: -2

Explanation: The subarray {-2} has the largest sum -2.

Input: arr[] = {5, 4, 1, 7, 8}

Output: 25

Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

Code:

```
package com;

import java.util.Scanner;

public class maxsubarray {

    public static int maxsubarraysum(int arr[]) {
        int a=arr[0];
        int b=arr[0];
        for (int i=1; i<arr.length; i++) {
            a=Math.max(arr[i],a+arr[i]);
            b=Math.max(a, b);
        }
        return b;
    }

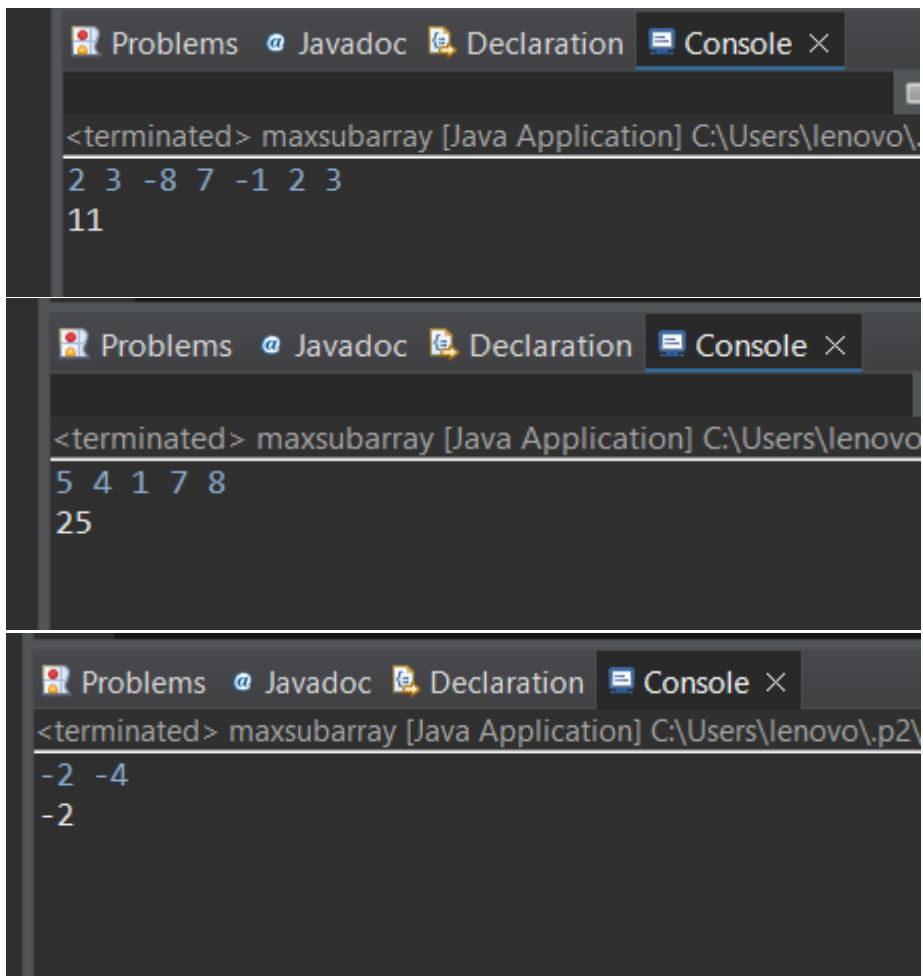
    public static void main(String[] args) {
        Scanner scanner =new Scanner(System.in);
        String input = scanner.nextLine();
    }
}
```

```

        String[] parts = input.split(" ");
        int[] arr = new int[parts.length];
        for (int i = 0; i < parts.length; i++) {
            arr[i] = Integer.parseInt(parts[i]);
        }
        int maxSum = maxsubarraysum(arr);
        System.out.println(maxSum);
        scanner.close();
    }
}

```

Output:



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

Input: arr[] = {-1, -3, -10, 0, 60}

Output: 60

Explanation: The subarray with maximum product is {60}.

Code:

```
package com;

import java.util.Scanner;

public class maxsubarray {

    public static int maxsubarrayproduct(int arr[]) {
        if (arr.length==0) return 0;
        int a=arr[0];
        int b=arr[0];
        int c=arr[0];
        for (int i=1; i<arr.length; i++) {
            if (arr[i]<0) {
                int temp=a;
                a=b;
                b=temp;
            }
            a=Math.max(arr[i], a*arr[i]);
            b=Math.min(arr[i], b*arr[i]);
            c=Math.max(c, a);
        }
        return c;
    }
}
```

```

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String input = scanner.nextLine();

        String[] parts = input.split(" ");

        int[] arr = new int[parts.length];

        for (int i = 0; i < parts.length; i++) {

            arr[i] = Integer.parseInt(parts[i]);

        }

        int maxPro = maxsubarrayproduct(arr);

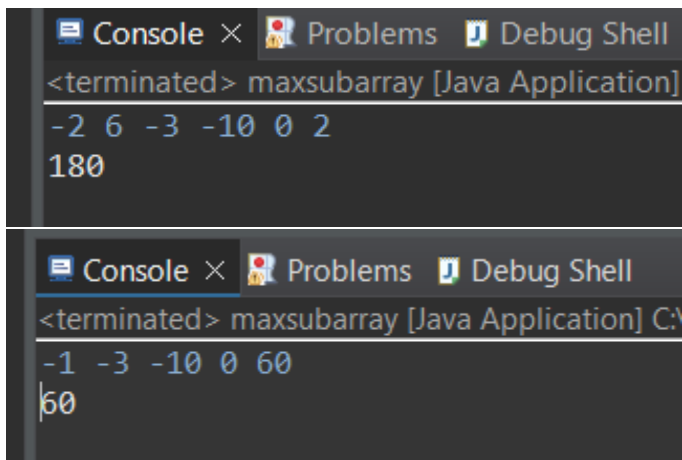
        System.out.println(maxPro);

        scanner.close();

    }
}

```

Output:



The image shows two screenshots of a Java IDE's console window. The top screenshot shows the output for the input array [-2, 6, -3, -10, 0, 2], which is 180. The bottom screenshot shows the output for the input array [-1, -3, -10, 0, 60], which is 60. Both screenshots show the console tabs at the top: 'Console', 'Problems', and 'Debug Shell'. The console text is as follows:

```

<terminated> maxsubarray [Java Application]
-2 6 -3 -10 0 2
180

```

```

<terminated> maxsubarray [Java Application] C:\
-1 -3 -10 0 60
60

```

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

Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3

Output : -1

Input : arr[] = {50, 10, 20, 30, 40}, key = 10

Output : 1

Code:

```
package com;
```

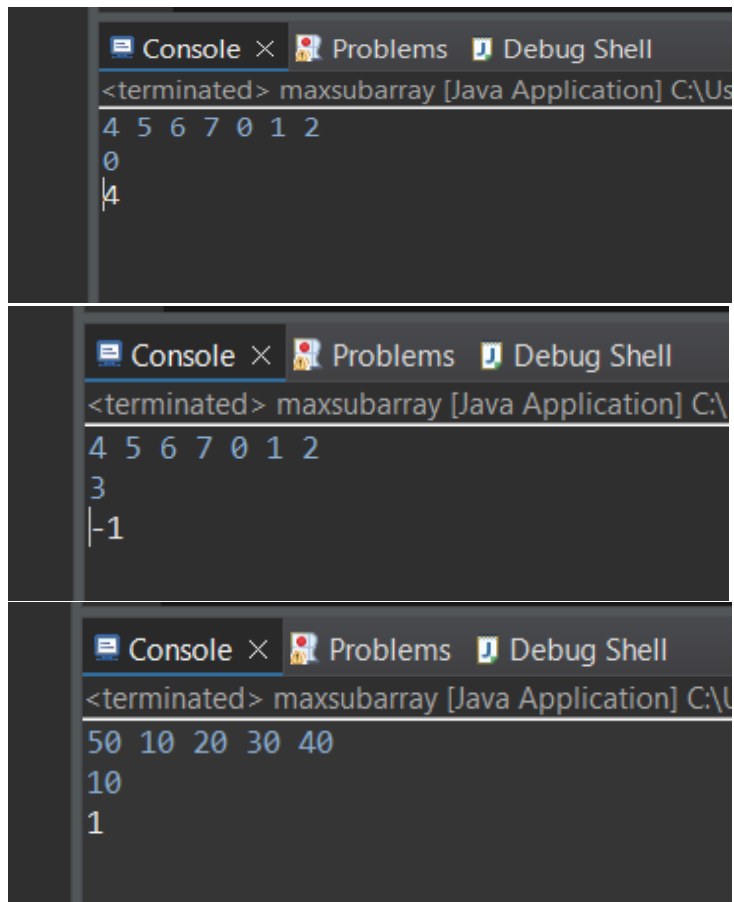
```
import java.util.Scanner;
```

```
public class maxsubarray {  
    public static int Search(int arr[], int k) {  
        int low = 0;  
        int high = arr.length - 1;  
        while (low <= high) {  
            int mid = low + (high-low) / 2;  
            if (arr[mid] == k) {  
                return mid;  
            }  
            if (arr[low] <= arr[mid]) {  
                if (k >= arr[low] && k < arr[mid]) {  
                    high = mid - 1;  
                } else {  
                    low = mid + 1;  
                }  
            } else {  
                if (k > arr[mid] && k <= arr[high]) {  
                    low = mid + 1;  
                }  
            }  
        }  
    }  
}
```

```
        } else {  
            high = mid - 1;  
        }  
    }  
}  
return -1;  
}
```

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    String input = scanner.nextLine();  
    String[] parts = input.split(" ");  
    int[] arr = new int[parts.length];  
    for (int i = 0; i < parts.length; i++) {  
        arr[i] = Integer.parseInt(parts[i]);  
    }  
    int k = scanner.nextInt();  
    int maxPro = Search(arr, k);  
    System.out.println(maxPro);  
    scanner.close();  
}  
}
```

Output:



The image displays three screenshots of an IDE's console window, each showing the output of a function named 'maxsubarray'. The console tabs are 'Console', 'Problems', and 'Debug Shell'. The first screenshot shows the array [4, 5, 6, 7, 0, 1, 2] with output 0 and index 4. The second screenshot shows the array [4, 5, 6, 7, 0, 1, 2] with output 3 and index -1. The third screenshot shows the array [50, 10, 20, 30, 40] with output 10 and index 1.

```
<terminated> maxsubarray [Java Application] C:\Us  
4 5 6 7 0 1 2  
0  
4  
  
<terminated> maxsubarray [Java Application] C:\Us  
4 5 6 7 0 1 2  
3  
-1  
  
<terminated> maxsubarray [Java Application] C:\Us  
50 10 20 30 40  
10  
1
```

Time Complexity: $O(\log n)$

4. Container with Most Water

Given n non-negative integers a_1, a_2, \dots , where each represents a point at coordinate (i, a_i) . ' n ' vertical lines are drawn such that the two endpoints of line i is at (i, a_i) and $(i, 0)$. Find two lines, which together with x-axis forms a container, such that the container contains the most water.

The program should return an integer which corresponds to the maximum area of water that can be contained (maximum area instead of maximum volume sounds weird but this is the 2D plane we are working with for simplicity).

Note: You may not slant the container.

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 = $3 * 2 = 6$

Input: $arr = [3, 1, 2, 4, 5]$

Output: 12

Explanation:

5 and 3 are distance 4 apart. So the size of the base = 4.

Height of container = $\min(5, 3) = 3$. So total area = $4 * 3 = 12$

Code:

```
package com;
```

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

```
public class maxsubarray {
    public static int maxarea(int[] h) {
        int a = 0;
        int b = 0;
        int c = h.length - 1;
        while (b < c) {
            int w = c - b;
            int newh = Math.min(h[b], h[c]);
            int newa = w * newh;
            a = Math.max(a, newa);
            if (h[b] < h[c]) {
                b++;
            } else {
                c--;
            }
        }
        return a;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
    }
}
```



```

String[] parts = input.split(" ");
int[] arr = new int[parts.length];
for (int i = 0; i < parts.length; i++) {
    arr[i] = Integer.parseInt(parts[i]);
}
int maxPro = maxarea(arr);
System.out.println(maxPro);
scanner.close();
}
}

```

Output:

The image shows two screenshots of a Java IDE console. The top screenshot shows the output for the input "1 5 4 3", which is "6". The bottom screenshot shows the output for the input "3 1 2 4 5", which is "12". Both screenshots show the console window with the title "Console" and the content "<terminated> maxsubarray [Java Application] C:\U".

Time complexity: $O(n)$

5. Find the Factorial of a large number

Input: 100

Output:

933262154439441526816992388562667004907159682643816214685929638952175999932
299156089414639761565182862536979208272237582511852109168640000000000000000
00000000

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

Code:

```
package com;

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

public class maxsubarray {

    public static BigInteger factorial(int n) {
        BigInteger ans=BigInteger.ONE;
        for (int i=1; i<=n; i+=1) {
            ans=ans.multiply(BigInteger.valueOf(i));
        }
        return ans;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int factNumber = scanner.nextInt();
        BigInteger fact = factorial(factNumber);
        System.out.println(fact);
        scanner.close();
    }
}
```

Output:

[illegible]

```
<terminated> maxsubarray [Java Application] C:\Users\lenovo\.p2\pool\plugins\org.eclipse.j
50
30414093201713378043612608166064768844377641568960512000000000000
```

Time complexity: $O(n^2 \log n)$

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.

Input: `arr[] = {3, 0, 2, 0, 4}`

Output: 7

Explanation: We trap $0 + 3 + 1 + 3 + 0 = 7$ units.

Input: `arr[] = {1, 2, 3, 4}`

Output: 0

Explanation : We cannot trap water as there is no height bound on both sides

Input: `arr[] = {10, 9, 0, 5}`

Output: 5

Explanation : We trap $0 + 0 + 5 + 0 = 5$

Code:

```
package com;

import java.util.Scanner;

class maxsubarray {
    static int trapWater(int[] arr) {
        int n = arr.length;
        if (n <= 2) return 0;
        int[] l = new int[n];
        int[] r = new int[n];
        int ans = 0;
        l[0] = arr[0];
        for (int i = 1; i < n; i++) {
            l[i] = Math.max(l[i - 1], arr[i]);
        }
    }
}
```

```

    r[n - 1] = arr[n - 1];
    for (int i = n - 2; i >= 0; i--) {
        r[i] = Math.max(r[i + 1], arr[i]);
    }
    for (int i = 1; i < n - 1; i++) {
        ans += Math.min(l[i], r[i]) - arr[i];
    }
    return ans;
}

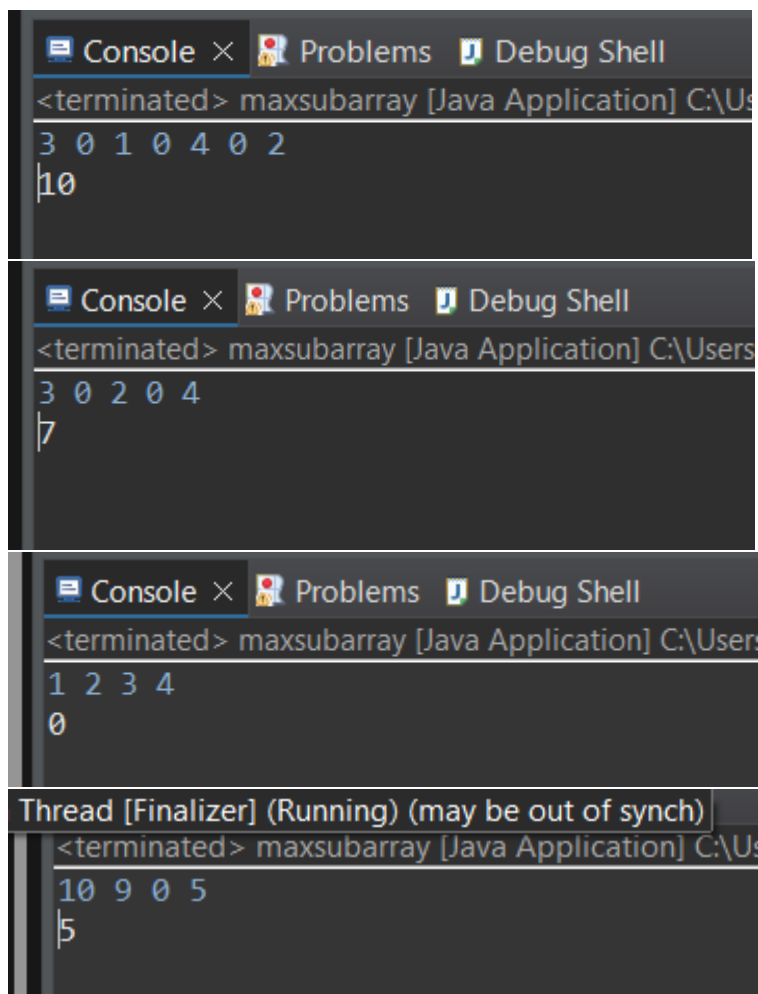
```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    String input = scanner.nextLine();
    String[] parts = input.split(" ");
    int[] arr = new int[parts.length];
    for (int i = 0; i < parts.length; i++) {
        arr[i] = Integer.parseInt(parts[i]);
    }
    System.out.println(trapWater(arr));
    scanner.close();
}
}

```

Output:



The image displays four sequential screenshots of a Java IDE's console window, showing the output of a function named `maxsubarray`. Each screenshot includes a tab bar with 'Console', 'Problems', and 'Debug Shell' tabs, and a status bar indicating the application has terminated.

- First screenshot:** The input array is `3 0 1 0 4 0 2`. The output is `10`.
- Second screenshot:** The input array is `3 0 2 0 4`. The output is `7`.
- Third screenshot:** The input array is `1 2 3 4`. The output is `0`.
- Fourth screenshot:** The input array is `10 9 0 5`. The output is `5`.

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.

Input: `arr[] = {7, 3, 2, 4, 9, 12, 56}`, $m = 5$

Output: 7

Explanation: If we distribute chocolate packets `{3, 2, 4, 9, 7}`, we will get the minimum difference, that is $9 - 2 = 7$.

Code:

```
package com;

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

public class maxsubarray {

    public static int chocolate(int[] arr, int m) {
        int n = arr.length;
        if (n < m) {
            return -1;
        }
        Arrays.sort(arr);
        int md = Integer.MAX_VALUE;

        for (int i = 0; i + m - 1 < n; i++) {
```

```

        int diff = arr[i + m - 1] - arr[i];

        md = Math.min(md, diff);
    }

    return md;
}

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

    String input = scanner.nextLine();
    String[] parts = input.split(" ");
    int[] arr = new int[parts.length];

    for (int i = 0; i < parts.length; i++) {
        arr[i] = Integer.parseInt(parts[i]);
    }

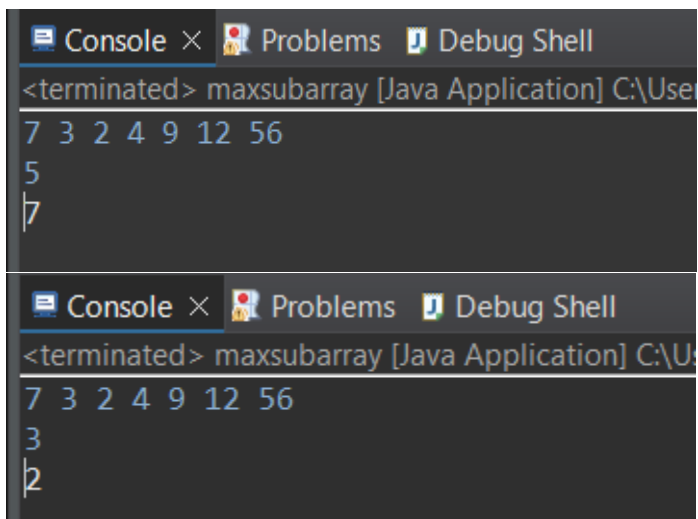
    int m = scanner.nextInt();

    System.out.println(chocolate(arr, m));

    scanner.close();
}
}

```

Output:



```

Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\Use
7 3 2 4 9 12 56
5
7

Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\Use
7 3 2 4 9 12 56
3
3

```

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

8. Merge Overlapping Intervals

Given an array of time intervals where $arr[i] = [start_i, end_i]$, the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

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

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

$[1, 6]$.

Code:

```
package com;
```

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

```
public class maxsubarray {  
    public static int[][] mergeint(int[][] arr) {  
        if (arr.length <= 1) return arr;  
        Arrays.sort(arr, (a, b) -> Integer.compare(a[0], b[0]));  
        List<int[]> merged = new ArrayList<>();  
        merged.add(arr[0]);  
        for (int i = 1; i < arr.length; i++) {  
            int[] lastInterval = merged.get(merged.size() - 1);  
            if (arr[i][0] <= lastInterval[1]) {  
                lastInterval[1] = Math.max(lastInterval[1], arr[i][1]);  
            } else {  
                merged.add(arr[i]);  
            }  
        }  
        return merged.toArray(new int[merged.size()][]);  
    }  
}
```

```

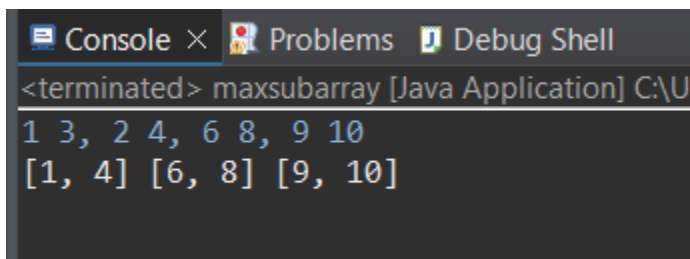
    }

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

        String input = scanner.nextLine();
        String[] parts = input.split(",\\s*");
        int[][] arr = new int[parts.length][2];
        for (int i = 0; i < parts.length; i++) {
            String[] interval = parts[i].split("\\s+");
            arr[i][0] = Integer.parseInt(interval[0]);
            arr[i][1] = Integer.parseInt(interval[1]);
        }
        int[][] result = mergeint(arr);
        for (int[] interval : result) {
            System.out.print(Arrays.toString(interval) + " ");
        }
        scanner.close();
    }
}

```

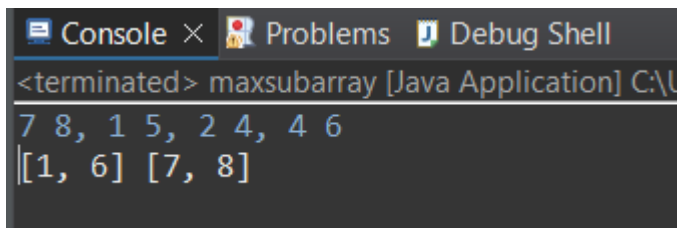
Output:



```

Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\U
1 3, 2 4, 6 8, 9 10
[1, 4] [6, 8] [9, 10]

```



```

Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\U
7 8, 1 5, 2 4, 4 6
[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 X 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}}`

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

`{0, 0, 1}}`

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

`{1, 1, 1}}`

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

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

`{0, 0, 0, 0}}`

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

`{1, 1, 1, 1},`

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

Code:

```
package com;
```

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

```
public class maxsubarray {  
    public static void modifyMatrix(int[][] mat) {  
        int m = mat.length;  
        int n = mat[0].length;  
        boolean[] rowFlag = new boolean[m];  
        boolean[] colFlag = new boolean[n];  
        for (int i = 0; i < m; i++) {  
            for (int j = 0; j < n; j++) {  
                if (mat[i][j] == 1) {
```

```

        rowFlag[i] = true;
        colFlag[j] = true;
    }
}
}
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        if (rowFlag[i] || colFlag[j]) {
            mat[i][j] = 1;
        }
    }
}
}

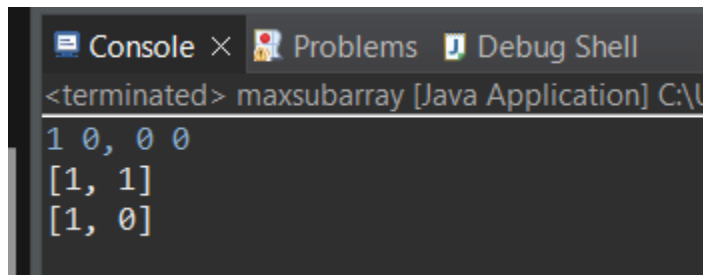
```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    String input = scanner.nextLine();
    String[] parts = input.split("\\s*");
    int numIntervals = parts.length;
    int[][] mat = new int[numIntervals][2];
    for (int i = 0; i < numIntervals; i++) {
        String[] interval = parts[i].split("\\s+");
        mat[i][0] = Integer.parseInt(interval[0].trim());
        mat[i][1] = Integer.parseInt(interval[1].trim());
    }
    modifyMatrix(mat);
    for (int[] row : mat) {
        System.out.println(Arrays.toString(row));
    }
    scanner.close();
}
}

```

Output:



The screenshot shows an IDE console window with three tabs: 'Console', 'Problems', and 'Debug Shell'. The 'Console' tab is active, displaying the following text: '<terminated> maxsubarray [Java Application] C:\U'. Below this, the output of the program is shown on three lines: '1 0, 0 0', '[1, 1]', and '[1, 0]'. The text is color-coded, with '1' in blue, '0' in orange, and '[' in green.

```
<terminated> maxsubarray [Java Application] C:\U
1 0, 0 0
[1, 1]
[1, 0]
```

Time complexity: $O(M+N)$

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

Input: matrix = { {1, 2, 3, 4, 5, 6},

{7, 8, 9, 10, 11, 12},

{13, 14, 15, 16, 17, 18} }

Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11

Explanation: The output is matrix in spiral format.

Code:

```
package com;
```

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

```
public class maxsubarray {
```

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

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

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

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

```
        int[][] matrix = new int[m][n];
```

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

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

```
                matrix[i][j] = sc.nextInt();
```

```
            }
```

```
        }
```

```
        printSpiral(matrix, m, n);
```

```
    }
```

```
    public static void printSpiral(int[][] matrix, int m, int n) {
```

```

int t = 0, b = m - 1, l = 0, r = n - 1;

while (t <= b && l <= r) {
    for (int i = l; i <= r; i++) {
        System.out.print(matrix[t][i] + " ");
    }
    t++;

    for (int i = t; i <= b; i++) {
        System.out.print(matrix[i][r] + " ");
    }
    r--;

    if (t <= b) {
        for (int i = r; i >= l; i--) {
            System.out.print(matrix[b][i] + " ");
        }
        b--;
    }

    if (l <= r) {
        for (int i = b; i >= t; i--) {
            System.out.print(matrix[i][l] + " ");
        }
        l++;
    }
}
}

```

Output:

```
Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\Users\len
4 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\Users\lenovo\p2\p
3 6
1 2 3 4 5 6
7 8 9 10 11 12

13 14 15 16 17 18
1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
```

Time complexity: $O(m*n)$

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

Input: str = “()()())”

Output: Not Balanced

Code:

```
package com;

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

public class maxsubarray {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String str = sc.next();

        if (isBalanced(str)) {

            System.out.println("Balanced");

        } else {

            System.out.println("Not Balanced");

        }

    }

    public static boolean isBalanced(String str) {

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

        for (int i = 0; i < str.length(); i++) {

            char ch = str.charAt(i);

            if (ch == '(') {

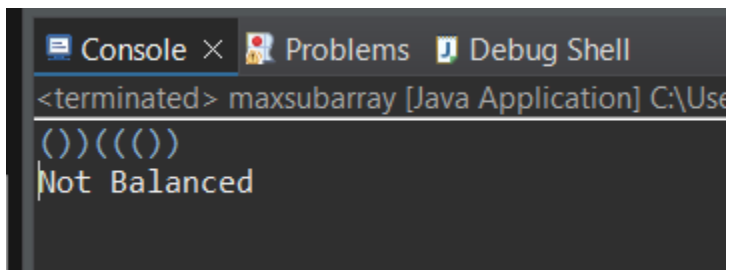
                stack.push(ch);

            } else if (ch == ')') {

                if (stack.isEmpty()) {
```

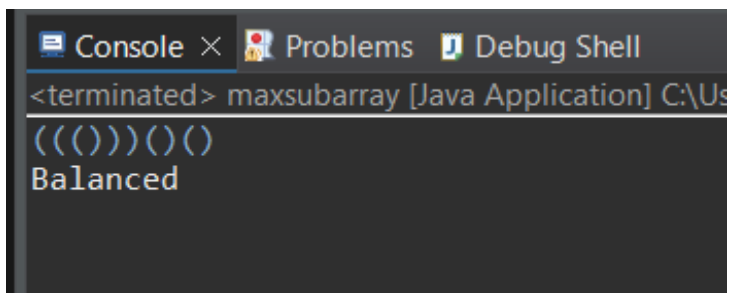
```
        return false;
    }
    stack.pop();
}
}
return stack.isEmpty();
}
}
```

Output:



The screenshot shows a Java IDE console with the following text:

```
<terminated> maxsubarray [Java Application] C:\Us
()(())
Not Balanced
```



The screenshot shows a Java IDE console with the following text:

```
<terminated> maxsubarray [Java Application] C:\Us
((()))()
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 string have same characters with same frequency. So, they are anagrams.

Input: s1 = "allergy" s2 = "allergic"

Output: false

Explanation: Characters in both the strings are not same. s1 has extra character „y“ and s2 has extra characters „i“ and „c“, so they are not anagrams.

Input: s1 = "g", s2 = "g"

Output: true

Explanation: Characters in both the strings are same, so they are anagrams.

Code:

```
package com;

import java.util.Scanner;

public class maxsubarray {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String s1 = sc.next();

        String s2 = sc.next();

        if (areAnagrams(s1, s2)) {

            System.out.println("true");

        } else {

            System.out.println("false");

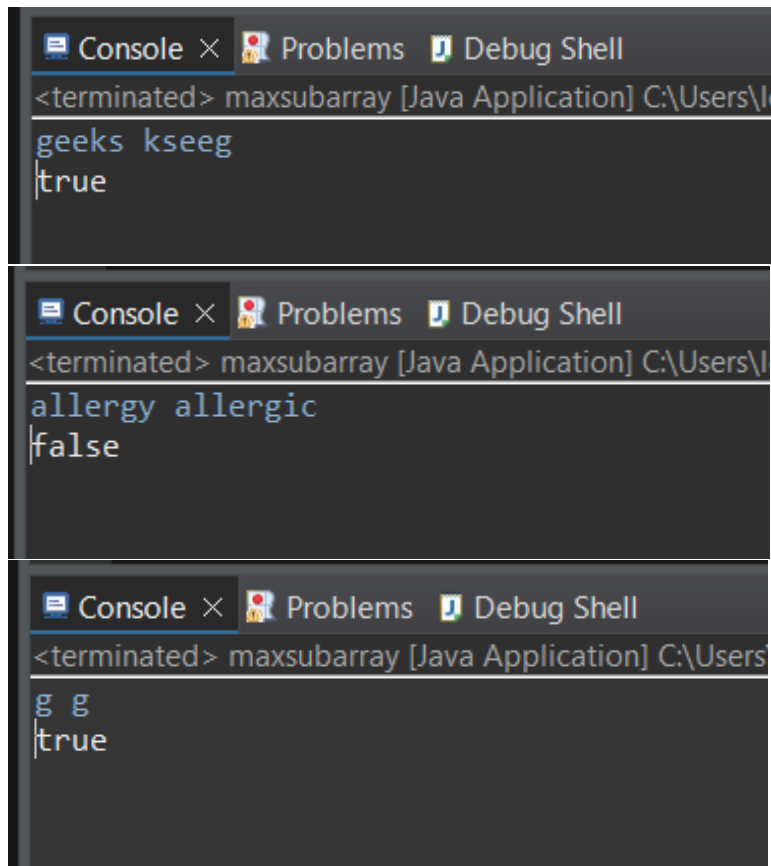
        }

    }

}
```

```
public static boolean areAnagrams(String s1, String s2) {  
    if (s1.length() != s2.length()) {  
        return false;  
    }  
    int[] count = new int[26];  
    for (int i = 0; i < s1.length(); i++) {  
        count[s1.charAt(i) - 'a']++;  
        count[s2.charAt(i) - 'a']--;  
    }  
    for (int i = 0; i < 26; i++) {  
        if (count[i] != 0) {  
            return false;  
        }  
    }  
    return true;  
}
```

Output:



The image displays three screenshots of a Java IDE's console window, showing the output of a function named 'maxsubarray'. Each screenshot has a tab bar at the top with 'Console', 'Problems', and 'Debug Shell'. The first screenshot shows the input 'geeks kseeg' and the output 'true'. The second screenshot shows the input 'allergy allergic' and the output 'false'. The third screenshot shows the input 'g g' and the output 'true'. In all cases, the output is preceded by the text '<terminated> maxsubarray [Java Application] C:\Users\l'.

```
<terminated> maxsubarray [Java Application] C:\Users\l  
geeks kseeg  
true  
  
<terminated> maxsubarray [Java Application] C:\Users\l  
allergy allergic  
false  
  
<terminated> maxsubarray [Java Application] C:\Users\l  
g g  
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.

Input: str = "Geeks"

Output: "ee"

Input: str = "abc"

Output: "a"

Input: str = ""

Output: ""

Code:

```
package com;

import java.util.Scanner;

public class maxsubarray {

    static boolean checkPal(String s, int low, int high) {
        while (low < high) {
            if (s.charAt(low) != s.charAt(high))
                return false;
            low++;
            high--;
        }
        return true;
    }

    static String longestPalSubstr(String s) {
        int n = s.length();
```

```

int maxLen = 1, start = 0;

for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        if (checkPal(s, i, j) && (j - i + 1) > maxLen) {
            start = i;
            maxLen = j - i + 1;
        }
    }
}

return s.substring(start, start + maxLen);
}

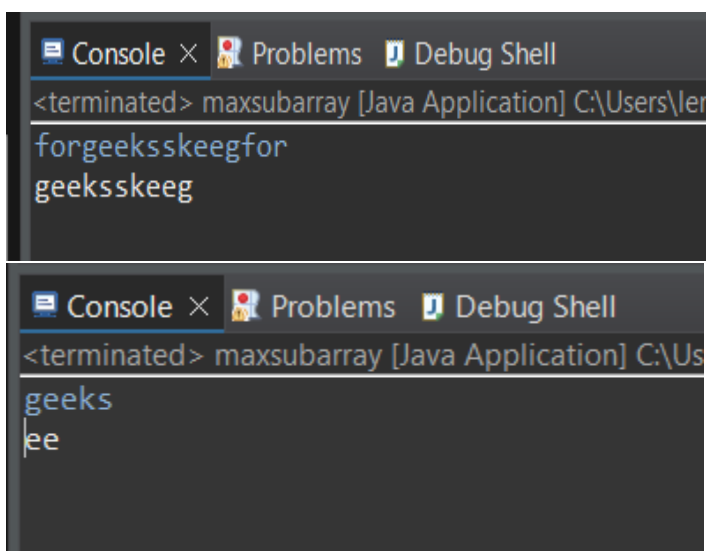
```

```

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String s = sc.nextLine();
    System.out.println(longestPalSubstr(s));
    sc.close();
}
}

```

Output:



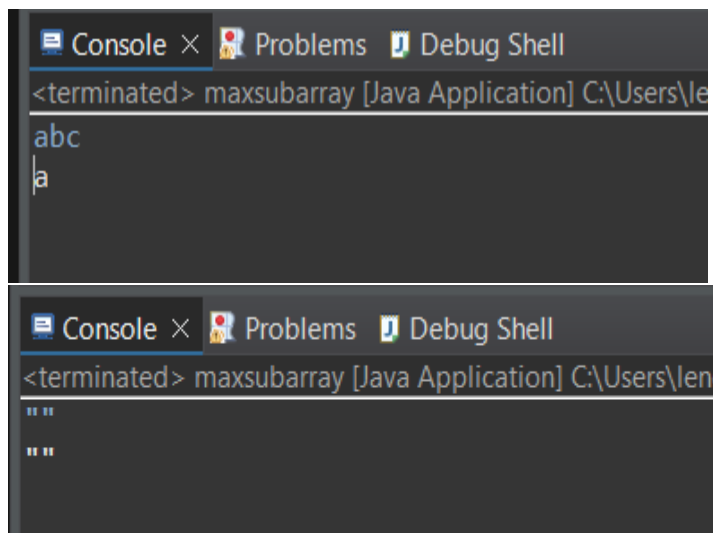
The image shows two screenshots of a Java IDE's console window. The top screenshot shows the input string "forgeeksskeegfor" and the output "forgeeksskeeg". The bottom screenshot shows the input string "geekslee" and the output "geeks".

```

Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\Users\ler
forgeeksskeegfor
forgeeksskeeg

Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\Us
geeks
lee

```



Time Complexity: $O(N^3)$

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.

Input: `arr[] = ["hello", "world"]`

Output: `-1`

Explanation: There's no common prefix in the given strings.

Code:

```
package com;

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

public class maxsubarray {

    static String longestCommonPrefix(String[] arr) {
        Arrays.sort(arr);
        String first = arr[0];
        String last = arr[arr.length - 1];
        int n = Math.min(first.length(), last.length());
        int i = 0;
        while (i < n && first.charAt(i) == last.charAt(i)) {
            i++;
        }
        if (i == 0) {
            return "-1";
        }
        return first.substring(0, i);
    }

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

```

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

sc.nextLine();

String[] arr = new String[n];

for (int i = 0; i < n; i++) {
    arr[i] = sc.nextLine();
}

System.out.println(longestCommonPrefix(arr));

sc.close();
}
}

```

Output:

The image shows two screenshots of a Java IDE console. The first screenshot shows the output of the program for the input 'geeksforgeeks', 'geeks', 'geek', 'geezer', and 'gee'. The output is '4' (the length of the longest common prefix) followed by the prefix 'gee'. The second screenshot shows the output for the input 'hello', 'world', and '-1'. The output is '2' (the length of the longest common prefix) followed by the prefix 'h'.

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

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]

Input : Stack[] = [1, 2, 3, 4, 5, 6]

Output : Stack[] = [1, 2, 4, 5, 6]

Code:

```
package com;

import java.util.Stack;
import java.util.Vector;
import java.util.Scanner;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class maxsubarray {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        sc.nextLine();

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

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

            st.push(sc.nextLine().charAt(0));

        }

        if (st.isEmpty()) {

            System.out.println("Stack is empty.");

            return;

        }

        Vector<Character> v = new Vector<Character>();

        while (!st.empty()) {

            v.add(st.pop());

        }

    }

}
```

```

        int midIndex = v.size() / 2;

        v.remove(midIndex);

        List<Character> sortedList = new ArrayList<>(v);

        Collections.sort(sortedList);

        for (Character c : sortedList) {

            System.out.print(c + " ");

        }

        sc.close();

    }

}

```

Output:

The image shows two screenshots of a Java IDE console. The top screenshot shows the input sequence '5 1 2 3 4 5' and the sorted output '1 2 4 5'. The bottom screenshot shows the input sequence '6 1 2 3 4 5 6' and the sorted output '1 2 4 5 6'. Both screenshots show the console window with tabs for 'Console', 'Problems', and 'Debug Shell'. The console title is '<terminated> maxsubarray [Java Application] C:\User'.

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

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 exist, 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

Input: arr[] = [13 , 7 , 6 , 12]

Output: 13 → -1

7 → 12

6 → 12

12 → -1

Explanation: 13 and 12 don't have any element greater than them present on the right side.

Code:

```
package com;

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

public class maxsubarray {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

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

            arr[i] = sc.nextInt();

        }

    }

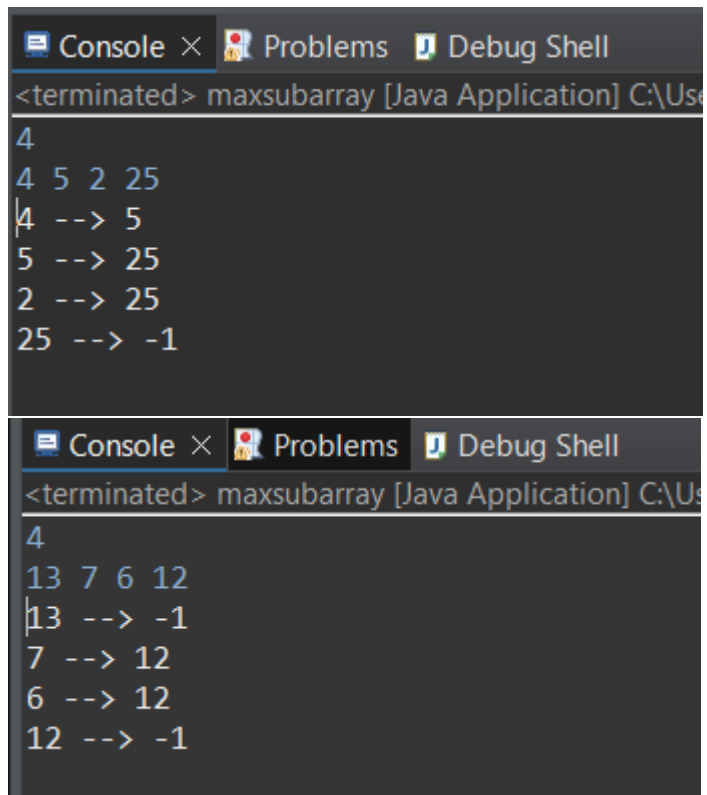
}
```

```

Stack<Integer> stack = new Stack<>();
int[] nge = new int[n];
for (int i = 0; i < n; i++) {
    nge[i] = -1;
}
for (int i = n - 1; i >= 0; i--) {
    while (!stack.isEmpty() && stack.peek() <= arr[i]) {
        stack.pop();
    }
    if (!stack.isEmpty()) {
        nge[i] = stack.peek();
    }
    stack.push(arr[i]);
}
for (int i = 0; i < n; i++) {
    System.out.println(arr[i] + " --> " + nge[i]);
}
sc.close();
}
}

```

Output:



```
Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\Us
4
4 5 2 25
4 --> 5
5 --> 25
2 --> 25
25 --> -1

Console × Problems Debug Shell
<terminated> maxsubarray [Java Application] C:\Us
4
13 7 6 12
13 --> -1
7 --> 12
6 --> 12
12 --> -1
```

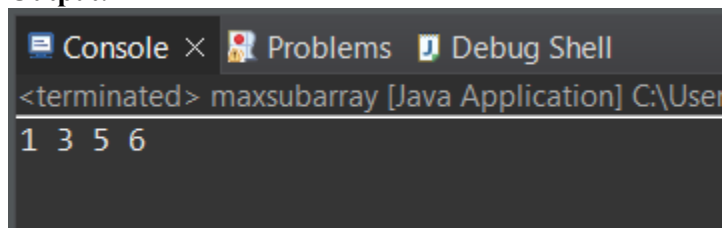
Time complexity: $O(n)$

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.

Code:

```
package com;
import java.util.LinkedList;
import java.util.Queue;
class Node {
    int data;
    Node left, right;
    Node(int data) {
        this.data = data;
        left = right = null;
    }
}
public class maxsubarray {
    Node root;
    public void printRightView() {
        if (root == null) return;
        Queue<Node> queue = new LinkedList<>();
        queue.offer(root);
        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                Node node = queue.poll();
                if (i == size - 1) {
                    System.out.print(node.data + " ");
                }
                if (node.left != null) queue.offer(node.left);
                if (node.right != null) queue.offer(node.right);
            }
        }
    }
    public static void main(String[] args) {
        maxsubarray tree = new maxsubarray();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.right = new Node(4);
        tree.root.left.right.right = new Node(6);
        tree.root.right.right = new Node(5);
        tree.printRightView();
    }
}
```


Output:



The screenshot shows an IDE console window with three tabs: 'Console', 'Problems', and 'Debug Shell'. The 'Console' tab is active, displaying the text '<terminated> maxsubarray [Java Application] C:\User' on the first line and '1 3 5 6' on the second line. The text is in a light blue font on a dark background.

```
<terminated> maxsubarray [Java Application] C:\User
1 3 5 6
```

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.

Code:

```
package com;

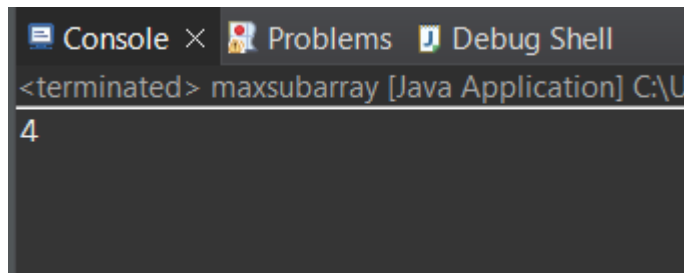
class Node {
    int data;
    Node left, right;
    public Node(int item) {
        data = item;
        left = right = null;
    }
}

class maxsubarray {
    Node root;
    int maxDepth(Node node) {
        if (node == null)
            return 0;
        else {
            int leftDepth = maxDepth(node.left);
            int rightDepth = maxDepth(node.right);
            return Math.max(leftDepth, rightDepth) + 1;
        }
    }

    public static void main(String[] args) {
        maxsubarray tree = new maxsubarray();
        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(6);
        System.out.println(tree.maxDepth(tree.root));
    }
}
```

```
}  
}
```

Output:

A screenshot of an IDE's console window. The title bar shows three tabs: 'Console' (active), 'Problems', and 'Debug Shell'. The console text shows the command prompt '<terminated> maxsubarray [Java Application] C:\U' followed by the output '4' on the next line.

```
<terminated> maxsubarray [Java Application] C:\U  
4
```

Time complexity: $O(n)$

Name: Vaishalini R

Department: AI-DS

Reg no.: 22AD152