

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.

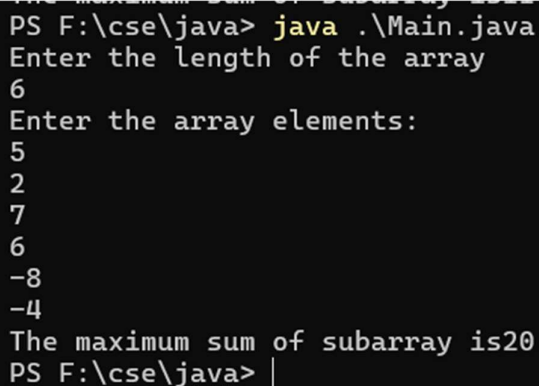
Solution:

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the length of the array");
        int n=s.nextInt();
        int[] a = new int[n];
        System.out.println("Enter the array elements:");

        for(int i=0;i<n;i++){
            a[i]=s.nextInt();
        }
        int m = Integer.MIN_VALUE, s = 0;
        for(int i=0;i<n;i++){
            sum += nums[i];
            max = Math.max(s,m);

            if(s<0) s = 0;
        }
        System.out.println(m);
    }
}
```

Output:



```
PS F:\cse\java> java .\Main.java
Enter the length of the array
6
Enter the array elements:
5
2
7
6
-8
-4
The maximum sum of subarray is20
PS F:\cse\java> |
```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

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}.

Solution :

```
import java.util.*;
class Product{
public static void main(String[] args){
Scanner s=new Scanner(System.in);
System.out.println("Enter the length of the array");
int n=s.nextInt();
int[] a=new int[n];
System.out.println("Enter the array elements");
for(int i=0;i<n;i++){
a[i]=s.nextInt();
}
int m=Integer.MIN_VALUE;
for(int i=0;i<n;i++){
int p=1;
for(int j=i;j<n;j++){
p*=a[j];
m=Math.max(p,m);
}
}
System.out.println("The maximum product of an array is "+m);
}
}
```

Output:

```
PS F:\cse\java> java .\Product.java
Enter the length of the array
6
Enter the array elements
8
5
4
-9
6
2
The maximum product of an array is 160
PS F:\cse\java> |
```

Time Complexity: $O(n^2)$

Space Complexity: $O(1)$

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

Solution:

```
import java.util.*;
public class Search {
    public static void main(String[] args) {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the length");
        int n=s.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter Array Elements:");
        for(int i=0;i<n;i++){
            nums[i]=s.nextInt();
        }
        System.out.println("Enter Target element");
        int target = s.nextInt();
        int left = 0;
        int right = nums.length - 1;
        int result = -1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] == target) {
                result = mid;
                break;
            }

            if (nums[left] <= nums[mid]) {
                if (target >= nums[left] && target < nums[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            } else {
                if (target > nums[mid] && target <= nums[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }
        }

        if (result != -1) {
```

```

        System.out.println("Element found at index: " + result);
    } else {
        System.out.println("Element not found");
    }
}
}
}

```

Output:

```

PS F:\cse\java> javac .\Search.java
PS F:\cse\java> java .\Search.java
Enter the length
6
Enter Array Elements:
4
5
6
7
0
1
Enter Target element
0
Element found at index: 4
PS F:\cse\java> |

```

Time Complexity: $O(\log n)$

Space Complexity: $O(1)$

4.Container With most Water:

Given n non-negative integers a_1, a_2, \dots, a_n 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 = $4 * 3 = 12$

Solution:

```
import java.util.*;
class MaxArea{
public static void main(String[] args) {
    System.out.println("enter the length:");
    Scanner s=new Scanner(System.in);
    int n=s.nextInt();
    int h=new int[n];
    System.out.println("Enter the array elements");

    int l=0;
    int r=h.length-1;
    int m=0;
    while(l<r){
        int w=r-l;
        int c= Math.min(h[l],h[r]);
        int a=w*c;
        m=Math.max(m,a);
        if(h[l]<h[r]){
            l++;
        }
        else{
            r--;
        }
    }
    System.out.println(m);
}
```

Output:

```
PS F:\cse\java> javac .\MaxArea.java
PS F:\cse\java> java .\MaxArea.java
enter the length:
5
Enter the array elements
3
1
2
4
5
The maximum area of the water contained is 12
PS F:\cse\java> |
```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

5. Find the Factorial of a large number

Input: 100

Output:

933262154439441526816992388562667004907159682643816214685929638952175999932299
15608941463976156518286253697920827223758251185210916864000000000000000000
00

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

Solution:

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

public class Factorial {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = s.nextInt();

        BigInteger f= BigInteger.ONE;
        for (int i = 1; i <= number; i++) {
            f = f.multiply(BigInteger.valueOf(i));
        }

        System.out.println("Factorial of " + number + " is: " + f);
    }
}
```

Output:

```
PS F:\cse\java> javac .\Factorial.java
PS F:\cse\java> java .\Factorial.java
Enter a number: 100
Factorial of 100 is: 933262154439441526816992388562667004907159682643816214685929
638952175999932299156089414639761565182862536979208272237582511852109168640000000
000000000000000000
PS F:\cse\java> |
```

Time Complexity: $O(n)$

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

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

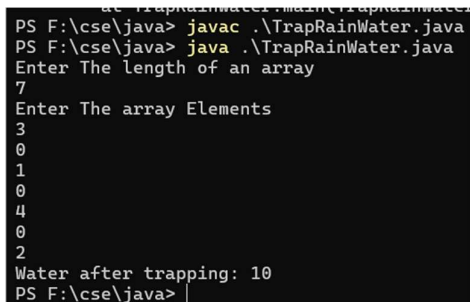
Explanation : We trap $0 + 0 + 5 + 0 = 5$

Solution :

```
import java.util.*;
class TrapRainWater {

    public static void main(String[] args) {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter The length of an array");
        int n=s.nextInt();
        int height[]=new int[n];
        System.out.println("Enter The array Elements");
        for(int i=0;i<n;i++){
            height[i]=s.nextInt();
        }
        int left = 0;
        int right = height.length - 1;
        int leftMax = height[left];
        int rightMax = height[right];
        int water = 0;
        while (left < right) {
            if (leftMax < rightMax) {
                left++;
                leftMax = Math.max(leftMax, height[left]);
                water += leftMax - height[left];
            } else {
                right--;
                rightMax = Math.max(rightMax, height[right]);
                water += rightMax - height[right];
            }
        }
        System.out.println("Water after trapping: " +water);
    }
}
```

Output:



```
PS F:\cse\java> javac .\TrapRainWater.java
PS F:\cse\java> java .\TrapRainWater.java
Enter The length of an array
7
Enter The array Elements
3
0
1
0
4
0
2
Water after trapping: 10
PS F:\cse\java> |
```

Time Complexity: $O(n)$

Space 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 $9 - 2 = 7$.

Solution:

```
import java.util.*;
class Chocolate {
    public static void main(String[] args){
        Scanner s =new Scanner(System.in);
        System.out.println("ENter the length of the array:");
        int n = s.nextInt();
        int arr[]=new int[n];
        System.out.println("Enter then array elements");
        for(int i=0;i<n;i++){
            arr[i]=s.nextInt();
        }
        System.out.println("Enter the number of Students: ");
        int m=s.nextInt();
        Arrays.sort(arr);
        int minDiff = Integer.MAX_VALUE;
        for (int i = 0; i + m - 1 < n; i++) {
            int diff = arr[i + m - 1] - arr[i];
            if (diff < minDiff)
                minDiff = diff;
        }
        System.out.println("The minimum Difference of the chocolates is :"+ minDiff) ;
    }
}
```

Output :

```
PS F:\cse\java> javac .\Chocolate.java
PS F:\cse\java> java .\Chocolate.java
ENter the length of the array:
7
Enter then array elements
7
3
2
4
9
12
56
Enter the number of Students:
5
The minimum Difference of the chocolates is :7
PS F:\cse\java> |
```

Time Complexity: $O(n)$

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

8. Merge Overlapping Intervals

Given an array of time intervals where $\text{arr}[i] = [\text{start}_i, \text{end}_i]$, the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

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

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: $\text{arr}[] = [[7, 8], [1, 5], [2, 4], [4, 6]]$

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

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

Solution:

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

```
class MergeIntervals {
```

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

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

```
        System.out.println("Enter the rows and columns");
```

```
        int a=s.nextInt();
```

```
        int b=s.nextInt();
```

```
        int[][] intervals = new int[a][b];
```

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

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

```
                intervals[i][j]=s.nextInt();
```

```
            }
```

```
        }
```

```
        Arrays.sort(intervals, Comparator.comparingInt(c -> c[0]));
```

```
        Stack<int[]> stack = new Stack<>();
```

```
        stack.push(new int[]{intervals[0][0], intervals[0][1]});
```

```
        for (int i = 1; i < intervals.length; i++) {
```

```
            int[] upper = stack.peek();
```

```
            if (intervals[i][0] <= upper[1]) {
```

```
                int[] newInterval = new int[]{upper[0], Math.max(intervals[i][1], upper[1])};
```

```
                stack.pop();
```

```
                stack.push(newInterval);
```

```
            } else {
```

```
                stack.push(new int[]{intervals[i][0], intervals[i][1]});
```

```
            }
```

```
        }
```

```
        int[][] result = stack.toArray(new int[stack.size()][]);
```

```
        System.out.println("Merged Intervals:");
```

```
        for (int[] interval : result) {
```

```
            System.out.println(Arrays.toString(interval));
```

```
        }
```

```
    }
```

```
}
```

Output:

```

PS F:\cse\java> java .\MergeIntervals.java
Enter the rows and columns
4
2
1
3
2
4
6
8
9
10
Merged Intervals:
[1, 4]
[6, 8]
[9, 10]
PS F:\cse\java> |

```

Time Complexity: $O(n^2)$

Space Complexity: $O(n)$

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}}

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}}

Solution:

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

```

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

        System.out.print("Enter the number of rows (M): ");
        int M = s.nextInt();

        System.out.print("Enter the number of columns (N): ");
        int N = scanner.nextInt();

        int[][] mat = new int[M][N];

        System.out.println("Enter the matrix elements (0 or 1): ");
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                mat[i][j] = scanner.nextInt();
            }
        }

        boolean[] rowFlags = new boolean[M];
        boolean[] colFlags = new boolean[N];

        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                if (mat[i][j] == 1) {
                    rowFlags[i] = true;
                    colFlags[j] = true;
                }
            }
        }

        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                if (rowFlags[i] || colFlags[j]) {
                    mat[i][j] = 1;
                }
            }
        }

        System.out.println("Modified Matrix:");
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(mat[i][j] + " ");
            }
            System.out.println();
        }
    }
}

```

Output:

```
1 14 15 16
PS F:\cse\java> java .\BooleanMatrix.java
Enter the number of rows (M): 2
Enter the number of columns (N): 2
Enter the matrix elements (0 or 1):
1
0
0
0
Modified Matrix:
1 1
1 0
PS F:\cse\java> java .\BooleanMatrix.java
Enter the number of rows (M): 2
Enter the number of columns (N): 3
Enter the matrix elements (0 or 1):
0
0
0
0
0
1
Modified Matrix:
0 0 1
1 1 1
PS F:\cse\java> |
```

Time Complexity: $O(m*n)$

Space 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.

Solution:

```
import java.util.Scanner;
class SpiralMatrix {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter the number of rows (m): ");
        int m = s.nextInt();
        System.out.print("Enter the number of columns (n): ");
        int n = s.nextInt();
        int[][] matrix = new int[m][n];
```

```

System.out.println("Enter the matrix elements:");
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        matrix[i][j] = s.nextInt();
    }
}

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

System.out.println("Spiral Order:");
while (top <= bottom && left <= right) {
    for (int i = left; i <= right; i++) {
        System.out.print(matrix[top][i] + " ");
    }
    top++;
    for (int i = top; i <= bottom; i++) {
        System.out.print(matrix[i][right] + " ");
    }
    right--;
    if (top <= bottom) {
        for (int i = right; i >= left; i--) {
            System.out.print(matrix[bottom][i] + " ");
        }
        bottom--;
    }
    if (left <= right) {
        for (int i = bottom; i >= top; i--) {
            System.out.print(matrix[i][left] + " ");
        }
        left++;
    }
}
}
}

```

Output:

```

PS F:\cse\java> javac .\SpiralMatrix.java
PS F:\cse\java> java .\SpiralMatrix.java
Enter the number of rows (m): 4
Enter the number of columns (n): 4
Enter the matrix elements:
1
2
3
4
5
6
7
87
9
10
11
12
13
14
15
16
Spiral Order:
1 2 3 4 87 12 16 15 14 13 9 5 6 7 11 10
PS F:\cse\java> |

```

Time Complexity: $O(m*n)$

Space 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

Solution:

```
import java.util.*;

public class BalancedParentheses {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter the parentheses expression: ");
        String str = s.nextLine();
        Stack<Character> stack = new Stack<>();
        boolean isBalanced = true;
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            if (ch == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                if (stack.isEmpty()) {
                    isBalanced = false;
                    break;
                } else {
                    stack.pop();
                }
            }
        }
        if (!stack.isEmpty()) {
            isBalanced = false;
        }
        if (isBalanced) {
            System.out.println("Balanced");
        } else {
            System.out.println("Not Balanced");
        }
    }
}
```

Output:

```
PS F:\cse\java> javac .\BalancedParentheses.java
PS F:\cse\java> java .\BalancedParentheses.java
Enter the parentheses expression: "((( )))()()"
Balanced
PS F:\cse\java> java .\BalancedParentheses.java
Enter the parentheses expression: "() )(( )"
Not Balanced
PS F:\cse\java> |
```

Time Complexity:O(n)

Space 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 = "kseeeg"

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.

Solution:

```
import java.util.*;
class AnagramCheck {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter the first string (s1): ");
        String s1 = s.nextLine();
        System.out.print("Enter the second string (s2): ");
        String s2 = s.nextLine();
        if (s1.length() != s2.length()) {
            System.out.println("false");
        } else {
            char[] arr1 = s1.toCharArray();
            char[] arr2 = s2.toCharArray();
            Arrays.sort(arr1);
            Arrays.sort(arr2);
            boolean isAnagram = Arrays.equals(arr1, arr2);
            System.out.println(isAnagram);
        }
    }
}
```

Output:

```
PS F:\cse\java> javac .\AnagramCheck.java
PS F:\cse\java> java .\AnagramCheck.java
Enter the first string (s1): allergy
Enter the second string (s2): allergic
false
PS F:\cse\java> |
```

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

Space 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", "eeksske" etc.

But the substring "geeksskeeg" is the longest among all.

Input: str = "Geeks"

Output: "ee"

Input: str = "abc"

Output: "a"

Input: str = ""

Output: ""

Solution:

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

```
public class LongestPalindromicSubstring {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String str = scanner.nextLine();

        int n = str.length();
        if (n == 0) {
            System.out.println("");
            scanner.close();
            return;
        }

        int start = 0;
        int maxLength = 1;

        for (int i = 0; i < n; i++){
            int left = i;
            int right = i;
            while (left >= 0 && right < n && str.charAt(left) == str.charAt(right)) {
                if (right - left + 1 > maxLength) {
                    start = left;
                    maxLength = right - left + 1;
                }
                left--;
                right++;
            }
            left = i;
            right = i + 1;
            while (left >= 0 && right < n && str.charAt(left) == str.charAt(right)) {
                if (right - left + 1 > maxLength) {
                    start = left;
                    maxLength = right - left + 1;
                }
                left--;
            }
        }
    }
}
```



```

        right++;
    }
}

System.out.println("Longest Palindromic Substring: " + str.substring(start, start + maxLength));
scanner.close();
}
}

```

Output:

```

PS F:\cse\java> javac .\LongestPalindromicSubstring.java
PS F:\cse\java> java .\LongestPalindromicSubstring.java
Enter the string: "leetcode"
Longest Palindromic Substring: ee
PS F:\cse\java> java .\LongestPalindromicSubstring.java
Enter the string: "codingninjas"
Longest Palindromic Substring: ingni
PS F:\cse\java> |

```

Time Complexity: $O(n^2)$

Space 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.

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

Output: -1

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

Solution:

```

import java.util.*;
class LongestPrefix {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the length of the array:");
        int n = sc.nextInt();
        sc.nextLine();
        String[] arr = new String[n];
        System.out.println("Enter the array Elements");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextLine();
        }

        Arrays.sort(arr);
        String first = arr[0];
    }
}

```

```

String last = arr[n - 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;
    }
}

if (commonPrefix.length() == 0) {
    System.out.println("-1");
} else {
    System.out.println(commonPrefix.toString());
}
}
}

```

Solution:

```

PS F:\cse\java> java .\LongestPrefix.java
Enter the length of the array:
4
Enter the array Elements
"geeksforgeeks"
"geeks"
"geek"
"geezer"
"gee
PS F:\cse\java> |

```

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

Space Complexity: $O(n)$

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]

Solution:

```

import java.util.*;

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

```

```

int n = sc.nextInt();
Stack<Integer> stack = new Stack<>();
for (int i = 0; i < n; i++) {
    stack.push(sc.nextInt());
}

int middleIndex = n / 2;
deleteMiddle(stack, middleIndex, 0);

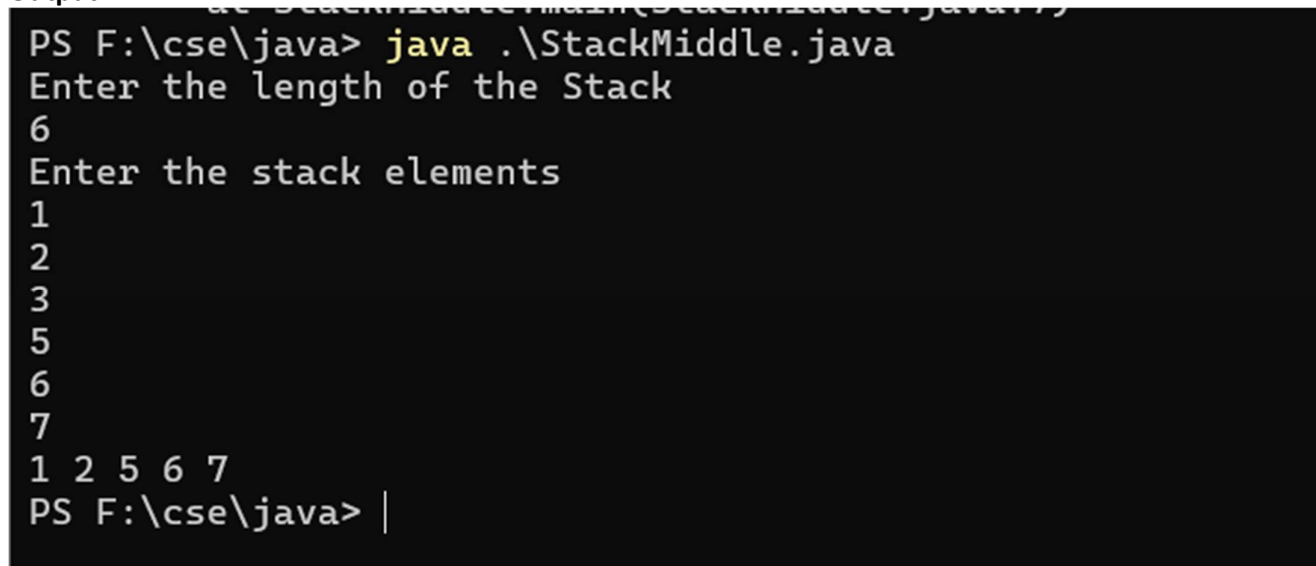
if (stack.isEmpty()) {
    System.out.println("-1");
} else {
    for (int i : stack) {
        System.out.print(i + " ");
    }
}
}

public static void deleteMiddle(Stack<Integer> stack, int middleIndex, int currentIndex) {
    if (currentIndex == middleIndex) {
        stack.pop();
        return;
    }

    int temp = stack.pop();
    deleteMiddle(stack, middleIndex, currentIndex + 1);
    stack.push(temp);
}
}

```

Output:



```

PS F:\cse\java> java .\StackMiddle.java
Enter the length of the Stack
6
Enter the stack elements
1
2
3
5
6
7
1 2 5 6 7
PS F:\cse\java> |

```

Time Complexity: $O(n)$

Space Complexity: $O(n)$

18. 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

2 ->

5 -> 25 -> 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 ->

7 -1 -> 12

6

12 -> 12 -> -1

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

Solution:

```
import java.util.*;
public class NGE {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Stack Length");
        int n = sc.nextInt();
        System.out.println("Enter array elements");
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        Stack<Integer> stack = new Stack<>();
        int[] result = new int[n];

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

            if (stack.isEmpty()) {
                result[i] = -1;
            } else {
                result[i] = stack.peek();
            }
            stack.push(arr[i]);
        }

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

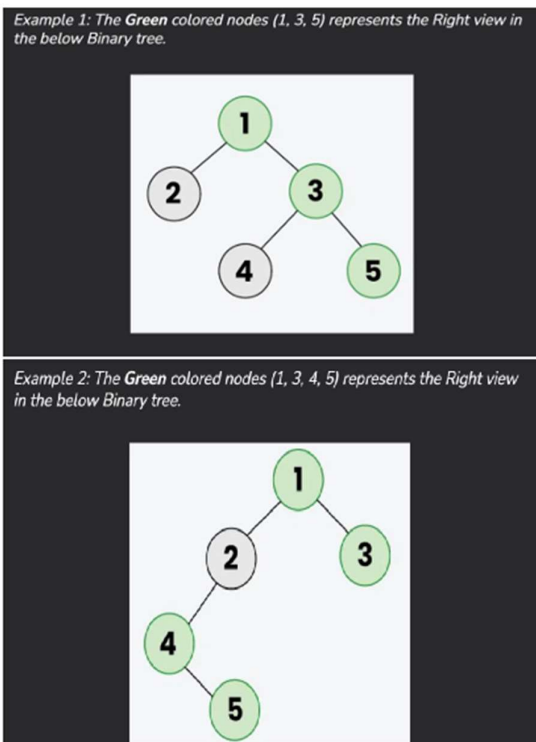
Output:

```
PS F:\cse\java> java NGE.java
Enter Stack Length
4
Enter array elements
13
7
6

12
13 - > -1
7 - > 12
6 - > 12
12 - > -1
PS F:\cse\java> |
```

Time Complexity: $O(n)$
Space 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.



Solution:

```
import java.util.*;
class TreeNode {
    int val;
    TreeNode left, right;
```



```

        curr.right = node;
        queue.add(node);
    }
}
}

```

```

Solution sol = new Solution();
List<Integer> result = sol.rightSideView(root);
System.out.println("Right side view: " + result);
}
}

```

Output:

```

PS F:\cse\java> javac .\BinaryRightView.java
PS F:\cse\java> java .\BinaryRightView.java
Enter the number of nodes in the tree:
5
Enter the tree elements (level order):
1
2
3
4
5
6
7
8
Right side view: [1, 3, 7, 8]
PS F:\cse\java> |

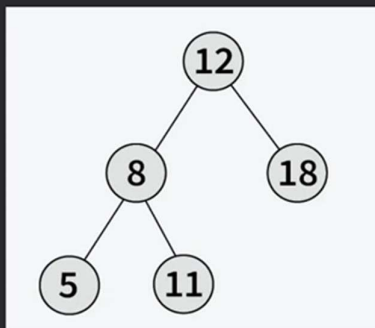
```

Time Complexity: $O(n)$

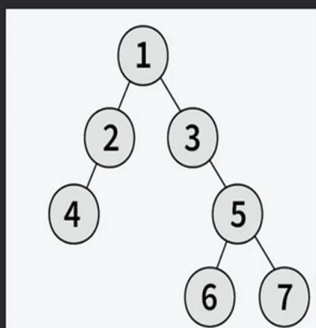
Space Complexity: $O(n)$

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 1: The height of the below binary tree is 3.



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



Solution:

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left, right;

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

class DepthTree {
    public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int leftDepth = maxDepth(root.left);
        int rightDepth = maxDepth(root.right);
        return Math.max(leftDepth, rightDepth) + 1;
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of nodes in the tree:");
    int n = sc.nextInt();

    if (n == 0) {
        System.out.println("Tree is empty. Maximum depth is 0.");
        return;
    }

    System.out.println("Enter the tree elements in level order (use -1 for null nodes):");
    Queue<TreeNode> queue = new LinkedList<>();
    TreeNode root = new TreeNode(sc.nextInt());
    queue.add(root);

    for (int i = 1; i < n; i++) {
        TreeNode curr = queue.poll();
        int leftVal = sc.nextInt();
        if (leftVal != -1) {
            curr.left = new TreeNode(leftVal);
            queue.add(curr.left);
        }
        if (++i < n) {
            int rightVal = sc.nextInt();
            if (rightVal != -1) {
                curr.right = new TreeNode(rightVal);
                queue.add(curr.right);
            }
        }
    }
}
```



```

    }
}

DepthTree sol = new DepthTree();
int depth = sol.maxDepth(root);
System.out.println("Maximum depth of the tree: " + depth);
}
}

```

Output:

```

PS F:\cse\java> javac .\DepthTree.java
PS F:\cse\java> java .\DepthTree.java
Enter the number of nodes in the tree:
7
Enter the tree elements in level order (use -1 for null nodes):
1
3
5
7
8
9
4
Maximum depth of the tree: 3
PS F:\cse\java> |

```

Time Complexity: $O(n)$

Space Complexity: $O(n)$