

Practice Set 1

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.

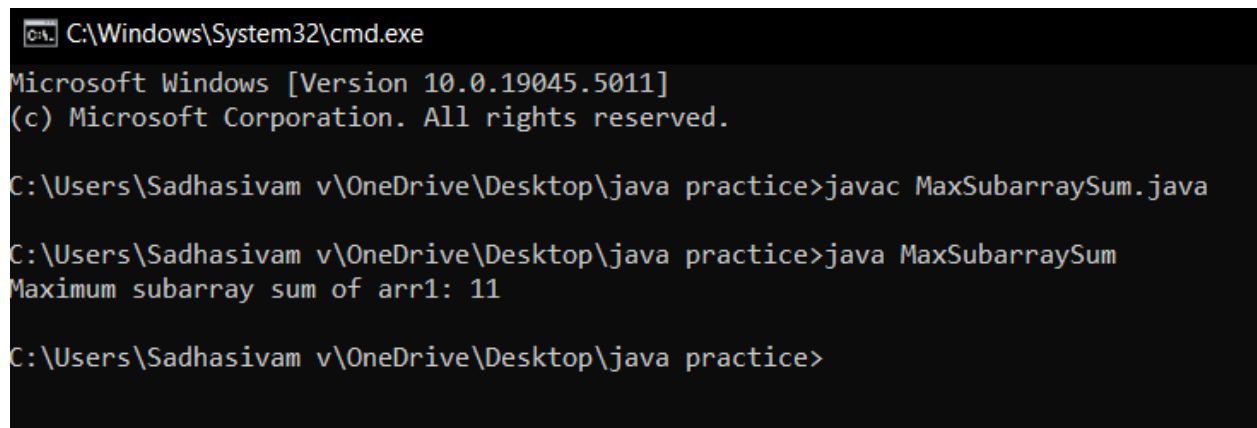
Input: `arr[] = {2, 3, -8, 7, -1, 2, 3}`

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

Solution:

```
import java.util.*;
public class MaxSubarraySum {
    public static int maxSubarraySum(int[] arr) {
        int maxSoFar = arr[0];
        int currentMax = arr[0];
        for (int i = 1; i < arr.length; i++) {
            currentMax = Math.max(arr[i], currentMax + arr[i]);
            maxSoFar = Math.max(maxSoFar, currentMax);
        }
        return maxSoFar;
    }
    public static void main(String[] args) {
        int[] arr1 = {2, 3, -8, 7, -1, 2, 3};
        System.out.println("Maximum subarray sum of arr1: " + maxSubarraySum(arr1));
    }
}
```

output:

A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\System32\cmd.exe'. The window content shows the following text: 'Microsoft Windows [Version 10.0.19045.5011]', '(c) Microsoft Corporation. All rights reserved.', 'C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac MaxSubarraySum.java', 'C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java MaxSubarraySum', 'Maximum subarray sum of arr1: 11', and 'C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>'.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac MaxSubarraySum.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java MaxSubarraySum
Maximum subarray sum of arr1: 11

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>
```

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}.

Solution:

```
import java.util.*;

public class MaxProductSubarray {
    public static int maxProductSubarray(int[] arr) {
        int maxProduct = arr[0], minProduct = arr[0], result = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < 0) {
                int temp = maxProduct;
                maxProduct = minProduct;
                minProduct = temp;
            }
            maxProduct = Math.max(arr[i], maxProduct * arr[i]);
            minProduct = Math.min(arr[i], minProduct * arr[i]);
            result = Math.max(result, maxProduct);
        }
        return result;
    }

    public static void main(String[] args) {
        int[] arr = {-2, 6, -3, -10, 0, 2};
        System.out.println(maxProductSubarray(arr));
    }
}
```

Output:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac MaxProductSubarray.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java MaxProductSubarray
180

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>
```

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

Solution:

```
import java.util.*;

public class RotatedArray {
    public static int search(int[] arr, int key) {
        int left = 0, right = arr.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == key) {
                return mid;
            }
            if (arr[left] <= arr[mid]) {
                if (arr[left] <= key && key < arr[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            } else {
                if (arr[mid] < key && key <= arr[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {4, 5, 6, 7, 0, 1, 2};
        int key = 0;
        System.out.println(search(arr, key));
    }
}
```

Coding:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac RotatedArray.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java RotatedArray
4
```

Time complexity: $O(\log n)$

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 = $3 * 2 = 6$

Coding:

```
public class ContainerWithMostWater {
    public static int maxArea(int[] arr) {
        int left = 0;
        int right = arr.length - 1;
        int maxArea = 0;
        while (left < right) {
            int height = Math.min(arr[left], arr[right]);
            int width = right - left;
            int area = height * width;
            maxArea = Math.max(maxArea, area);
        }
    }
}
```

```

        if (arr[left] < arr[right]) {
            left++;
        } else {
            right--;
        }
    }
    return maxArea;
}

public static void main(String[] args) {
    int[] arr = {1, 5, 4, 3};
    System.out.println("Maximum area: " + maxArea(arr));
}
}

```

Solution:

```

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac ContainerWithMostWater.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java ContainerWithMostWater
Maximum area: 6

```

Time complexity: $O(n)$

5.

Find the Factorial of a large number Input: 100 Output:

9332621544394415268169923885626670049071596826438162146859296389521759999322
99

1560894146397615651828625369792082722375825118521091686400000000000000000000

00 00 Input: 50 Output:

30414093201713378043612608166064768844377641568960512000000000000

Solution:

```

import java.math.BigInteger;
import java.util.Scanner;
public class LargeFactorial {
    public static BigInteger factorial(int n) {
        BigInteger result = BigInteger.ONE;
        for (int i = 1; i <= n; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
    }
}

```

$$\left. \begin{array}{l} \{ \\ \} \end{array} \right\}$$

output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac LargeFactorial.java  
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java LargeFactorial  
Enter a number: 100  
Factorial of 100 is:  
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582  
5118521091686400000000000000000000000
```

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

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

Code:

```
import java.util.Scanner;

public class TrappingRainwater {
    public static int trap(int[] arr) {
        if (arr == null || arr.length == 0) return 0;
        int n = arr.length;
        int left = 0, right = n - 1;
        int leftMax = arr[left], rightMax = arr[right];
        int waterTrapped = 0;
        while (left <= right) {
            if (arr[left] <= arr[right]) {
                if (arr[left] >= leftMax) {
                    leftMax = arr[left];
                } else {
                    waterTrapped += leftMax - arr[left];
                }
            }
        }
    }
}
```

```

    }
    left++;
} else {
    if (arr[right] >= rightMax) {
        rightMax = arr[right];
    } else {
        waterTrapped += rightMax - arr[right];
    }
    right--;
}
}
return waterTrapped;
}
public static void main(String[] args) {
    int[] arr1 = {3, 0, 1, 0, 4, 0, 2};
    System.out.println(trap(arr1));
}

```

Output:

```

C:\Users\rajad\OneDrive\Desktop\New folder>java TrappingRainwater.java
Trapped water: 10
Trapped water: 7
Trapped water: 0
Trapped water: 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:

```

import java.util.Arrays;
public class ChocolateDistribution {

```

```

public static int distributeChocolates(int[] arr, int m) {
    int n = arr.length;
    if (n < m) {
        return -1;
    }
    Arrays.sort(arr);
    int minDiff = Integer.MAX_VALUE;
    for (int i = 0; i <= n - m; i++) {
        int diff = arr[i + m - 1] - arr[i];
        minDiff = Math.min(minDiff, diff);
    }
    return minDiff;
}

public static void main(String[] args) {
    int[] arr1 = {7, 3, 2, 4, 9, 12, 56};
    int m1 = 3;
    System.out.println(distributeChocolates(arr1, m1));
}
}

```

Output:

```

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac ChocolateDistribution.java
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java ChocolateDistribution
2

```

Time complexity: $O(n)$

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 $[[1, 4], [6, 8], [9, 10]]$.

Code:

```

import java.util.Arrays;
import java.util.ArrayList;
public class MergeIntervals {
    public static int[][] mergeIntervals(int[][] intervals) {
        if (intervals.length == 0) {
            return new int[0][0];
        }
        Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
        ArrayList<int[]> merged = new ArrayList<>();

```



```

int[] currentInterval = intervals[0];
merged.add(currentInterval);
for (int[] interval : intervals) {
    if (interval[0] <= currentInterval[1]) {
        currentInterval[1] = Math.max(currentInterval[1], interval[1]);
    } else {
        currentInterval = interval;
        merged.add(currentInterval);
    }
}
return merged.toArray(new int[merged.size()][]);
}

public static void main(String[] args) {
    int[][] intervals = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
    int[][] result = mergeIntervals(intervals);
    for (int[] interval : result) {
        System.out.println "[" + interval[0] + ", " + interval[1] + "]");
    }
}
}

```

Output:

```

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac MergeIntervals.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java MergeIntervals
[1, 4]
[6, 8]
[9, 10]

```

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}}

Code:

```

public class BooleanMatrix {
    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) {
    int[][] mat1 = {{1, 0}, {0, 0}};
    modifyMatrix(mat1);
    printMatrix(mat1);
    int[][] mat2 = {{0, 0, 0}, {0, 0, 1}};
    modifyMatrix(mat2);
    printMatrix(mat2);
}

public static void printMatrix(int[][] mat) {
    for (int[] row : mat) {
        for (int val : row) {
            System.out.print(val + " ");
        }
        System.out.println();
    }
}
}

```

Output:

```

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac BooleanMatrix.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java BooleanMatrix
1 1
1 0
0 0 1
1 1 1

```

Time complexity: $O(M*N)$

10.

Print a given matrix in spiral form Given an m x 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

Solution:

```
public class SpiralMatrix {
    public static void printSpiral(int[][] matrix) {
        if (matrix == null || matrix.length == 0) return;
        int m = matrix.length;
        int n = matrix[0].length;
        int top = 0, bottom = m - 1, left = 0, right = n - 1;
        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++;
            }
        }
    }

    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3, 4},
            {5, 6, 7, 8},
            {9, 10, 11, 12},
            {13, 14, 15, 16}
        };
        printSpiral(matrix);
    }
}
```

Output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac SpiralMatrix.java
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java SpiralMatrix
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

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:

```
public class ParenthesesBalanced {
    public static String isBalanced(String str) {
        if (str == null || str.length() == 0) return "Balanced";
        int count = 0;
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == '(') count++;
            else if (str.charAt(i) == ')') count--;
            if (count < 0) return "Not Balanced";
        }
        return count == 0 ? "Balanced" : "Not Balanced";
    }
    public static void main(String[] args) {
        String str1 = "((( )))()";
        String str2 = "()(( ))";
        System.out.println(isBalanced(str1));
        System.out.println(isBalanced(str2));
    }
}
```

Output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac ParenthesesBalanced.java
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java ParenthesesBalanced
Balanced
Not 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.

Solution:

```
import java.util.Arrays;
public class AnagramCheck {
    public static boolean areAnagrams(String s1, String s2) {
        if (s1.length() != s2.length()) return false;
        char[] str1 = s1.toCharArray();
        char[] str2 = s2.toCharArray();
        Arrays.sort(str1);
        Arrays.sort(str2);
        return Arrays.equals(str1, str2);
    }

    public static void main(String[] args) {
        String s1 = "geeks";
        String s2 = "kseeg";
        System.out.println(areAnagrams(s1, s2));
    }
}
```

Output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac AnagramCheck.java
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java AnagramCheck
true
```

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

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.

Code:

```
public class LongestPalindromicSubstring {
```

```

public static String longestPalindrome(String str) {
    int n = str.length();
    if (n == 0) return "";
    String longest = str.substring(0, 1);
    for (int i = 0; i < n; i++) {
        String oddPalindrome = expandAroundCenter(str, i, i);
        String evenPalindrome = expandAroundCenter(str, i, i + 1);

        if (oddPalindrome.length() > longest.length()) {
            longest = oddPalindrome;
        }
        if (evenPalindrome.length() > longest.length()) {
            longest = evenPalindrome;
        }
    }
    return longest;
}

private static String expandAroundCenter(String str, int left, int right) {
    while (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {
        left--;
        right++;
    }
    return str.substring(left + 1, right);
}

public static void main(String[] args) {
    String str = "forgeeksskeegfor";
    System.out.println(longestPalindrome(str));
}
}

```

Output:

```

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac LongestPalindromicSubstring.java
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java LongestPalindromicSubstring
geeksskeeg

```

Time complexity:

$O(n^2)$

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.

Code:

```
import java.util.Arrays;

public class LongestCommonPrefix {

    public static String longestCommonPrefix(String[] arr) {
        if (arr == null || arr.length == 0) return "-1";
        Arrays.sort(arr);
        String first = arr[0];
        String last = arr[arr.length - 1];
        int i = 0;
        while (i < first.length() && i < last.length() && first.charAt(i) == last.charAt(i)) {
            i++;
        }
        return i > 0 ? first.substring(0, i) : "-1";
    }

    public static void main(String[] args) {
        String[] arr = {"geeksforgeeks", "geeks", "geek", "geezer"};
        System.out.println(longestCommonPrefix(arr));
    }
}
```

Output:

```
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac LongestCommonPrefix.java
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java LongestCommonPrefix
gee
```

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

Code:

```
import java.util.Stack;

public class DeleteMiddleElement {
    public static void deleteMiddle(Stack<Integer> stack, int size) {
        if (stack.isEmpty() || size == 0) return;
        int middle = size / 2;
        removeMiddleElement(stack, middle, 0);
    }

    private static void removeMiddleElement(Stack<Integer> stack, int middle, int currentIndex) {
```

```

        if (currentIndex == middle) {
            stack.pop();
            return;
        }
        int temp = stack.pop();
        removeMiddleElement(stack, middle, currentIndex + 1);
        stack.push(temp);
    }
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        int size = stack.size();
        deleteMiddle(stack, size);
        while (!stack.isEmpty()) {
            System.out.print(stack.pop() + " ");
        }
    }
}

```

Output:

```

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac DeleteMiddleElement.java
C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java DeleteMiddleElement
5 4 2 1

```

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 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

Code:

```

import java.util.Stack;
public class NextGreaterElement {
    public static void findNextGreater(int[] arr) {
        int n = arr.length;
    }
}

```



```

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

for (int i = 0; i < n; i++) {
    while (!stack.isEmpty() && arr[stack.peek()] < arr[i]) {
        int index = stack.pop();
        System.out.println("Element " + arr[index] + " --> NGE " + arr[i]);
    }
    stack.push(i);
}
while (!stack.isEmpty()) {
    int index = stack.pop();
    System.out.println("Element " + arr[index] + " --> NGE -1");
}
}
public static void main(String[] args) {
    int[] arr = {4, 5, 2, 25};
    findNextGreater(arr);
}
}

```

Output:

```

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>javac NextGreaterElement.java

C:\Users\Sadhasivam v\OneDrive\Desktop\java practice>java NextGreaterElement
Element 4 --> NGE 5
Element 2 --> NGE 25
Element 5 --> NGE 25
Element 25 --> NGE -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.*

