Name- Deepak S
Department- Information Technology

DSA-Practice-1

Coding practice Problems:

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.

Code:

```
import java.util.Arrays;

public class maxsubarray{
    public static int maxSubarraySum(int[] arr) {
        int res = arr[0];
        for (int i = 0; i < arr.length; i++) {
            int currSum = 0;
            for (int j = i; j < arr.length; j++) {
                currSum += arr[j];
                res = Math.max(res, currSum);
            }
        }
        return res;
    }

    public static void main(String[] args) {
        int[] arr = {2, 3, -8, 7, -1, 2, 3};
        int[] arr2 = {-2, -4};
        System.out.println("Maximum subarray sum is " + maxSubarraySum(arr));
             System.out.println("Maximum  subarray  sum  for  arr2  is  "  +
maxSubarraySum(arr2));
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac maxsubarray.java && java maxsubarray
Maximum subarray sum is 11
Maximum subarray sum for arr2 is -2

[Done] exited with code=0 in 0.926 seconds
```

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:

```java
public class maxProduct{

    public static int maxProduct(int[] arr) {
        int n = arr.length;
        int result = arr[0];
        for (int i = 0; i < n; i++) {
            int currentProduct = 1;
            for (int j = i; j < n; j++) {
                currentProduct *= arr[j];
                result = Math.max(result, currentProduct);
            }
        }
        return result;
    }

    public static void main(String[] args) {
        int[] arr1 = { -2, 6, -3, -10, 0, 2 };
        int[] arr2 = { -1, -3, -10, 0, 60};
        System.out.println("Maximum subarray product is " + maxProduct(arr1));
        System.out.println("Maximum subarray product is " + maxProduct(arr2));
```

```
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac maxProduct.java && java maxProduct
Maximum subarray product is 180
Maximum subarray product is 60

[Done] exited with code=0 in 0.928 seconds
```

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:

```
public class RotatedArraySearch {
    public static int search(int[] arr, int key) {
        int left = 0;
        int 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 (key >= arr[left] && key < arr[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            }
            else {
                if (key > arr[mid] && key <= arr[right]) {
                    left = mid + 1;
                } else {
```

```
                    right = mid - 1;
                }
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr1 = {4, 5, 6, 7, 0, 1, 2};
        int[] arr2 = {4, 5, 6, 7, 0, 1, 2};
        int[] arr3 = {50, 10, 20, 30, 40};
        System.out.println("Index of 0: " + search(arr1, 0));
        System.out.println("Index of 3: " + search(arr2, 3));
        System.out.println("Index of 10: " + search(arr3, 10));
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac RotatedArraySearch.java && java RotatedArraySearch
Index of 0: 4
Index of 3: -1
Index of 10: 1

[Done] exited with code=0 in 0.818 seconds
```

4.Container with Most Water:

Given n non-negative integers a_1, a_2, ..., 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).
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:

```
public class mostwater {
    public static int maxArea(int[] arr) {
        int n = arr.length;
        int area = 0;
```

```
        int left = 0;
        int right = n - 1;

        while (left < right) {
             area = Math.max(area, Math.min(arr[left], arr[right]) * (right -
left));

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

    public static void main(String[] args) {
        int[] a = {1, 5, 4, 3};
        int[] b = {3, 1, 2, 4, 5};

        System.out.println(maxArea(a));
        System.out.println(maxArea(b));
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac mostwater.java && java mostwater
6
12

[Done] exited with code=0 in 1.084 seconds
```

5. Find the Factorial of a large number

Input: 100

Output:
933262154439441526816992388562667004907159682643816214685929638952175999932299
156089414639761565182862536979208272237582511852109168640000000000000000000000 00

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

Code:

```java
import java.math.BigInteger;

public class factoriallarge {
    public static void main(String[] args) {
        System.out.println("factorial of 100 :" + fact(100));
        System.out.println("factorial of 50 :"+ fact(50));


    }
    public static BigInteger fact(int n){
        if(n == 1) return BigInteger.ONE;
        return BigInteger.valueOf(n).multiply(fact(n-1));
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac factoriallarge.java && java factoriallarge
factorial of 100
:933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
factorial of 50 :30414093201713378043612608166064768844377641568960512000000000000

[Done] exited with code=0 in 1.233 seconds
```

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:

```java
public class RainTest {
    public static void main(String[] args) {
        int[][] testCases = {
            {3, 0, 1, 0, 4, 0, 2},
            {3, 0, 2, 0, 4},
            {1, 2, 3, 4},
            {10, 9, 0, 5}
        };

        for (int i = 0; i < testCases.length; i++) {
                System.out.println("Test Case " + (i + 1) + ": " +
trappingRainWater(testCases[i]));
        }
    }

    public static int trappingRainWater(int boundary[]) {
        int leftMax = boundary[0];
        int leftBoundary[] = new int[boundary.length];

        for (int i = 0; i < boundary.length; i++) {
            if (boundary[i] > leftMax) {
                leftMax = boundary[i];
            }
            leftBoundary[i] = leftMax;
        }

        int rightMax = boundary[boundary.length - 1];
        int rightBoundary[] = new int[boundary.length];

        for (int i = boundary.length - 1; i >= 0; i--) {
            if (rightMax < boundary[i]) {
                rightMax = boundary[i];
            }
            rightBoundary[i] = rightMax;
        }

        int water = 0;
        for (int i = 0; i < boundary.length; i++) {
            int minHeightBar = Math.min(leftBoundary[i], rightBoundary[i]);
            water += minHeightBar - boundary[i];
        }
        return water;
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac RainTest.java && java RainTest
Test Case 1: 10
Test Case 2: 7
Test Case 3: 0
Test Case 4: 5
```

7. Chocolate Distribution Problem

Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith 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:

```java
import java.util.Arrays;
public class chocolate {
    public static void main(String[] args) {
        int[] arr1 = {7, 3, 2, 4, 9, 12, 56};
        int m1 = 3;
        int[] arr2 = {7, 3, 2, 4, 9, 12, 56};
        int m2 = 5;
        System.out.println(findMinDiff(arr1, m1));
        System.out.println(findMinDiff(arr2, m2));
    }
    static int findMinDiff(int[] arr, int m) {
        int n = arr.length;
        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;
        }
```

```
        return minDiff;
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac chocolate.java && java chocolate
2
7

[Done] exited with code=0 in 1.315 seconds
```

---

8. Merge Overlapping Intervals

Given an array of time intervals where arr[i] = [starti, endi], 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:

```
import java.util.Arrays;

public class mergeinterval {
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> a[0] - b[0]);

        int[][] merged = new int[intervals.length][2];
        int i = 0;

        merged[i] = intervals[0];

        for (int j = 1; j < intervals.length; j++) {
            if (merged[i][1] >= intervals[j][0]) {
```

```
                merged[i][1] = Math.max(merged[i][1], intervals[j][1]);
            } else {
                i++;
                merged[i] = intervals[j];
            }
        }

        return Arrays.copyOf(merged, i + 1);
    }

    public static void main(String[] args) {
        mergeinterval solution = new mergeinterval();

        int[][] arr1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        int[][] result1 = solution.merge(arr1);
        System.out.println(Arrays.deepToString(result1));

        int[][] arr2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};
        int[][] result2 = solution.merge(arr2);
        System.out.println(Arrays.deepToString(result2));
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac mergeinterval.java && java mergeinterval
[[1, 4], [6, 8], [9, 10]]
[[1, 6], [7, 8]]

[Done] exited with code=0 in 1.193 seconds
```

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 ith row and jth 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:

```
import java.util.HashSet;
```

```java
public class boolean{
    public static void main(String[] args) {
        int[][] a = {{1, 0}, {0, 0}};
        int[][] result = spirals(a);
        for (int[] row : result) {
            for (int val : row) {
                System.out.print(val + " ");
            }
            System.out.println();
        }
    }

    public static int[][] spirals(int[][] matrix) {
        HashSet<Integer> rows = new HashSet<>();
        HashSet<Integer> cols = new HashSet<>();
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (matrix[i][j] == 1) {
                    rows.add(i);
                    cols.add(j);
                }
            }
        }
        for (int i : rows) {
            for (int j = 0; j < matrix[0].length; j++) {
                matrix[i][j] = 1;
            }
        }
        for (int j : cols) {
            for (int i = 0; i < matrix.length; i++) {
                matrix[i][j] = 1;
            }
        }
        return matrix;
    }
}
```

Output;

```
[Running] cd "e:\Dsa hw\" && javac BooleanMatrix.java && java BooleanMatrix
[[1, 1], [1, 0]]
[[0, 0, 1], [1, 1, 1]]
[[1, 1, 1, 1], [1, 1, 1, 1], [1, 0, 1, 1]]

[Done] exited with code=0 in 0.863 seconds
```

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

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:

```
import java.util.*;

public class spiral {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        if (matrix == null || matrix.length == 0) {
            return result;
        }

        int top = 0;
        int bottom = matrix.length - 1;
        int left = 0;
        int right = matrix[0].length - 1;

        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) {
                result.add(matrix[top][i]);
            }
            top++;

            for (int i = top; i <= bottom; i++) {
                result.add(matrix[i][right]);
            }
            right--;

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

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

```
                    result.add(matrix[i][left]);
                }
                left++;
            }
        }

        return result;
    }

    public static void main(String[] args) {
        spiral sol = new spiral();
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        System.out.println(sol.spiralOrder(matrix));
    }
}
```

Outut:

```
[Running] cd "e:\Dsa hw\" && javac spiral.java && java spiral
[1, 2, 3, 6, 9, 8, 7, 4, 5]

[Done] exited with code=0 in 0.874 seconds
```

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 Parentheses {
    public static boolean isBalanced(String exp)
{
    boolean flag = true;
    int count = 0;
```

```
    for(int i = 0; i < exp.length(); i++)
    {
        if (exp.charAt(i) == '(')
        {
            count++;
        }
        else
        {
            count--;
        }
        if (count < 0)
        {
            flag = false;
            break;
        }
    }
    if (count != 0)
    {
        flag = false;
    }
    return flag;
}
public static void main(String[] args)
{
    String exp1 = "((()))()()";

    if (isBalanced(exp1))
        System.out.println("Balanced");
    else
        System.out.println("Not Balanced");

    String exp2 = "())((()))";

    if (isBalanced(exp2))
        System.out.println("Balanced");
    else
        System.out.println("Not Balanced");
}
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac Parentheses.java && java Parentheses
Balanced
Not Balanced

[Done] exited with code=0 in 1.201 seconds
```

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:

```java
import java.util.Arrays;

public class Anagram {
    public boolean isAnagram(String s, String t) {
        if (s.length() != t.length()) {
            return false;
        }

        char[] sArray = s.toCharArray();
        char[] tArray = t.toCharArray();

        Arrays.sort(sArray);
        Arrays.sort(tArray);

        return Arrays.equals(sArray, tArray);
    }

    public static void main(String[] args) {
        Anagram solution = new Anagram();
```

```
        String[][] testCases = {
            {"geeks", "kseeg"},
            {"allergy", "allergic"},
            {"g", "g"}
        };

        for (String[] testCase : testCases) {
            String s1 = testCase[0];
            String s2 = testCase[1];
            boolean result = solution.isAnagram(s1, s2);
            System.out.println("s1 = \"" + s1 + "\", s2 = \"" + s2 + "\" -> "
+ result);
        }
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac Anagram.java && java Anagram
s1 = "geeks", s2 = "kseeg" -> true
s1 = "allergy", s2 = "allergic" -> false
s1 = "g", s2 = "g" -> true

[Done] exited with code=0 in 1.168 seconds
```

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:

```
public class Palindromic {

    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();
        if (n == 0) return "";

        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) {
        String[] testCases = {"forgeeksskeegfor", "Geeks", "abc", ""};

        for (String s : testCases) {
            System.out.println("Input: \"" + s + "\"");
            System.out.println("Output: \"" + longestPalSubstr(s) + "\"\n");
        }
    }
}
```

Output

```
[Running] cd "e:\Dsa hw\" && javac Palindromic.java && java Palindromic
Input: "forgeeksskeegfor"
Output: "geeksskeeg"

Input: "Geeks"
Output: "ee"

Input: "abc"
Output: "a"

Input: ""
Output: ""


[Done] exited with code=0 in 1.109 seconds
```

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

Code:

```java
import java.util.Arrays;

public class LongestCommonPrefix {

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

```
        String prefix = first.substring(0, i);
        return prefix.isEmpty() ? "-1" : prefix;
    }
    public static void main(String[] args) {
        String[][] testCases = {
            {"geeksforgeeks", "geeks", "geek", "geezer"},
            {"hello", "world"},
            {"interview", "internet", "internal"},
            {"apple", "applesauce", "applause"}
        };
        for (String[] arr : testCases) {
            System.out.println("Input: " + Arrays.toString(arr));
            System.out.println("Prefix: " + longestCommonPrefix(arr) + "\n");
        }
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac LongestCommonPrefix.java && java Longes
Input: [geeksforgeeks, geeks, geek, geezer]
Prefix: gee

Input: [hello, world]
Prefix: -1

Input: [interview, internet, internal]
Prefix: inter

Input: [apple, applesauce, applause]
Prefix: appl
```

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:

```java
import java.util.Stack;
import java.util.Vector;
public class stack {

    public static void main(String[] args) {
        Stack<Character> st = new Stack<Character>();
        st.push('1');
        st.push('2');
        st.push('3');
        st.push('4');
        st.push('5');
        st.push('6');

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

        while (!st.empty()) {
            v.add(st.pop());
        }

        int n = v.size();
        int target = n / 2;

        for (int i = n - 1; i >= 0; i--) {
            if (i == target) continue;
            st.push(v.get(i));
        }

        System.out.print("Printing stack after deletion of middle: ");
        while (!st.empty()) {
            char p = st.pop();
            System.out.print(p + " ");
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac stack.java && java stack
Printing stack after deletion of middle: 6 5 4 2 1
[Done] exited with code=0 in 0.903 seconds
```

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 –> 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

Code:

```
public class NextGreater {
    public static void main(String[] args) {
        int[][] testCases = {
            {4, 5, 2, 25},
            {13, 7, 6, 12}
        };

        for (int[] arr : testCases) {
            System.out.println("Input array: ");
            for (int num : arr) {
                System.out.print(num + " ");
            }
            System.out.println("\nOutput:");
            printNextGreaterElements(arr);
            System.out.println();
        }
    }

    public static void printNextGreaterElements(int[] arr) {
        int n = arr.length;
        int[] result = new int[n];
```

```
        int maxElement = Integer.MIN_VALUE;

        for (int i = n - 1; i >= 0; i--) {
            if (arr[i] < maxElement) {
                result[i] = maxElement;
            } else {
                result[i] = -1;
            }
            maxElement = Math.max(maxElement, arr[i]);
        }

        for (int i = 0; i < n; i++) {
             System.out.println(arr[i] + " -> " + (result[i] == -1 ? "-1" :
result[i]));
        }
    }
}
```

Output:

```
[Running] cd "e:\Dsa hw\" && javac NextGreater.java && java NextGreater
Input array:
4 5 2 25
Output:
4 -> 25
5 -> 25
2 -> 25
25 -> -1

Input array:
13 7 6 12
Output:
13 -> -1
7 -> 12
6 -> 12
12 -> -1
```