LABSHEET - 6

CS F211 - Data Structures and Algorithms
Academic Year: 2025-2026 (2nd Semester)

General Instructions for this Labsheet:

- Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.
- All the questions **except the last one** should be done by implementing stacks using arrays or linked lists.
- Question 10 should be done by implementing a modified Linked list.
- All the basic operations such as push(x), pop(), top(), size() should be implemented. Any other function should be implemented, if specified.
- Consider 0-based indexing whenever needed.

Problem 1: The Stock Market Analysis (Stock Span)

Problem Description: A financial analyst wants to calculate the "Span" of a stock's price for each day. The **Span** of the stock's price on a given day is defined as the maximum number of consecutive days just before the given day, for which the price of the stock on the current day is **less than or equal to** its price on the given day.

- **Example:** If prices are [100, 80, 60, 70, 60, 75, 85].
    - On day 6 (Price 75), it is greater than 60, 70, 60. The span is 4 days (Indices 2, 3, 4, 5).
- **Constraint:** You must solve this using a **Stack** .
- **Hint:** The stack should store indices of days where prices are *decreasing* .

Input Format:

- Line 1: N(Total days).
- Line 2: N integers (Stock prices).

Output Format:

- Print the span for each day.

Sample Test Case 1:
Input:
7
100 80 60 70 60 75 85

Output:
1 1 1 2 1 4 6

Sample Test Case 2:
Input:
5
10 20 30 40 50

Output:
1 2 3 4 5

Problem 2: The Daily Temperatures (Next Greater Element)

Problem Description: You are given a list of daily temperatures. For each day, you want to know **how many days you have to wait** until a **warmer** temperature occurs. If there is no future day for which this is possible, put 0.

- **Example:** [73, 74, 75, 71, 69, 72, 76, 73]
    - On day 0 (73), the next warmer is day 1 (74). Wait: 1 day.
    - On day 2 (75), the next warmer is day 6 (76). Wait: 4 days.

Input Format:

- Line 1: N (Total days).
- Line 2: N Integers (Temperatures).

Output Format:

- Print the wait days for each position.

  Sample Test Case 1:

  Input:
  8
  73 74 75 71 69 72 76 73

  Output:
  1 1 4 2 1 1 0 0

  Sample Test Case 2:

  Input:
  4
  30 40 50 60

  Output:
  1 1 1 0

Problem 3: Largest Rectangle in Histogram

Problem Description:

You are given an array of integers heights representing the histogram's bar height where the width of each bar is 1. Find the area of the **largest rectangle** that can be formed within the histogram.

- **0-based indexing** has been used in this question.

Input Format:

- Line 1: N (Number of bars).
- Line 2: N integers (Heights).

Output Format:

- Print the maximum area found.

Sample Test Case 1:

Input:
5
2 5 6 2 3

Output:
10

Explanation: The largest rectangle is formed by bars 5 and 6 (indices 1 and 2), which both have height >= 5. Width is 2. Area = 5 * 2 = 10.
Other possible solution: all indices from [0, 4] with height = 2; Area = 2 * 5 = 10

Sample Test Case 2:

Input:
7
1 3 7 9 3 6 5

Output:
18

Explanation: The largest rectangle is in the subarray: [1, 6] with height 3.
Area = 3 * 6 = 18

Problem 4: The Peak Tracker (Min-Stack)

Problem Description

Design a custom stack data structure that supports all standard operations: **Push**, **Pop**, and **Peek**, while also providing a specialized operation called **GetMin**. The **GetMin** operation must return the minimum element currently present in the stack.

Input Format

- The first line contains an integer **N**, representing the number of operations to be performed.
- Each of the next **N** lines contains an operation:
    - **1 x**: Push the integer **x** onto the stack.
    - **2**: Pop the top element from the stack.
    - **3**: Display the current minimum element (**GetMin**).
    - **4**: Display the top element (**Peek**).

Output Format

- For **Pop**, print: "Popped: [value]"
- For **GetMin**, print: "Minimum: [value]"
- For **Peek**, print: "Top: [value]"
- If the stack is empty during Pop, Peek, or GetMin, print: "Stack Empty"

Sample Test Case 1

Input:
7
1 30
1 15
3
4
2
3
4

Output:
Minimum: 15
Top: 15
Popped: 15
Minimum: 30
Top: 30

Sample Test Case 2

Input:
8
1 50
2
3
1 10
1 20
3
4
2

Output:
Popped: 50
Stack Empty
Minimum: 10
Top: 20
Popped: 20

Problem 5: The Linear Expression Evaluator

Problem Description:

Write a program to evaluate the numerical result of a mathematical expression written in **Postfix Notation** (also known as Reverse Polish Notation). In this notation, every operator follows all of its operands, which eliminates the need for parentheses and complex operator precedence rules. The expression consists of single-digit operands (0-9) and four basic arithmetic operators: + (addition), - (subtraction), * (multiplication), and / (division).

Evaluation Rules:

1. Process the expression string from left to right.
2. If the character is a digit, push its integer value onto the stack.
3. If the character is an operator, pop the two most recent values from the stack.
4. Apply the operator: The first value popped is the right operand, and the second value popped is the left operand.
5. Push the result back onto the stack.
6. Division by Zero: If the right operand is 0 during a division operation, the program must print "Invalid Expression" and terminate.
7. Final Result: After the string is fully processed, the single value remaining on the stack is the answer, printed to two decimal places. If the stack is empty or contains more than one value at the end, print "Invalid Expression".
8. All intermediate calculations must be performed using floating-point arithmetic.

Input Format

- A single string representing the expression (no spaces).

Output Format

- A single value representing the final result, formatted to two decimal places.
- For a structurally incorrect expression, print: "Invalid Expression"

Sample Test Case 1

Input:
842*+93/-
Output:
13.00
Logic: 8 + (4 * 2) - (9 / 3) = 8 + 8 - 3 = 13
Explanation:The expression 842*+93/- is evaluated as follows: First, 4 * 2 is calculated to get 8, then added to 8 to get 16. Next, 9/3 is calculated to get 3, which is then subtracted from 16 to yield the final result of 13.00.

Sample Test Case 2

Input:
93/5+2-6*8/
Output:
4.50
Logic: (((9 / 3) + 5) - 2) * 6 / 8
Explanation: The expression 93/5+2-6*8/ is evaluated as follows: First, 9 / 3 gives 3, which is added to 5 (8) and then subtracted by 2 to get 6. Finally, 6 is multiplied by 6 (36) and divided by 8 to yield the final result of 4.50.

Problem 6: The Maximum Valid Segment

Description

Given a string consisting only of the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

A substring is considered valid if every opening bracket has a corresponding closing bracket and they are nested or sequenced correctly. For example, in the string) ( ) ( )), the longest valid substring is ( ) ( ), which has a length of 4.

Input Format

- A single line containing the string of parentheses.

Output Format

- A single integer representing the length of the longest valid substring.

Sample Test Case 1

Input:
)()())

Output:
4

Explanation: The longest valid substring is "()()"

Sample Test Case 2

Input:
()((()))

Output:
6

Explanation: The longest valid substring is "()((())"

Problem 7: String Decoder

Problem Description:

Given an encoded string, return its decoded string. The encoding rule is:

- k[es], where the string **es** (can be a string or an encoded string) inside the square brackets is being repeated exactly **k** times.

Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid, and the decoded string will contain characters only from a-z.

Input Format:
Line 1: N (Size of the encoded string)
Line 2: A single encoded string.

Output Format:
Fully decoded string

Sample Test Case 1:

Input:
13
2[a]1[b]2[cd]

Output:
aabcdcd

Sample Test Case 2:

Input:
8
4[a2[b]]

Output:
abbabbabbabb

Sample Test Case 3:

Input:
16
3[a1[b2[c]1[d]]]

Output:

abccdabccdabccd

Problem 8: Convert String

Problem Description:

You are given two sequences of integers: Sequence1 (Input) and Sequence2 (Target).
Your task is to determine if Sequence2 can be generated from Sequence1 using a **single** stack.

You process Sequence1 from left to right. For each element in Sequence1, you must push it onto the stack. At any point, if the top of the stack matches the next required element in Sequence2, you may pop it to match the target sequence.

Input Format:
Line 1: N (Length of the sequences)
Line 2: N integers (Sequence1)
Line 3: N integers (Sequence2)

Output Format:
Print the sequence of operations ("Push" or "Pop") required to transform Sequence1 into Sequence2.
If the transformation is impossible, print "Impossible".

Sample Test Case 1:

Input:
3
1 2 3
1 2 3

Output:
Push
Pop
Push
Pop
Push
Pop

Explanation:
Push 1; Top is 1; Target is 1; Match found -> **Pop** 1.
Push 2; Top is 2; Target is 2; Match found -> **Pop** 2.
Push 3; Top is 3; Target is 3; Match found -> **Pop** 3.

Sample Test Case 2:

Input:
4
1 4 2 3
2 3 4 1

Output:
Push
Push
Push
Pop
Push
Pop
Pop
Pop

Explanation:
Push 1; Top is 1; Target is 2; No match -> Push next.
Push 4; Top is 4; Target is 2; No match -> Push next.
Push 2; Top is 2; Target is 2; Match found -> **Pop** 2.
Top is now 4; Target is 3; No match -> Push next.
Push 3. Top is 3; Target is 3; Match found -> **Pop** 3.
Top is now 4; Target is 4; Match found -> **Pop** 4.
Top is now 1; Target is 1; Match found -> **Pop** 1.

Sample Test Case 3:

Input:
3
1 2 3
3 1 2

Output:
Impossible

Explanation:
Push 1; **Push** 2; **Push** 3 to reach the first target 3.
Pop 3; Stack is now [1, 2] (with 2 at top).
Next target is 1. However, the top of the stack is 2.
We cannot pop 1 before popping 2.
Hence, the conversion is **impossible**.

Problem 9: The Minimalist Subarray Range

Problem Description

Given an array of N integers, your task is to find the sum of the minimum elements of every possible contiguous subarray.

Since the number of subarrays can be large (N*(N+1)/2), you must use a Stack to solve this efficiently in O(N) time.

Input Format

- The first line contains an integer N, representing the size of the array.
- The second line contains N space-separated integers(≥ 1).

Output Format

- A single integer representing the sum of the minimums of all subarrays.

Sample Test Case 1

Input:
3
1 2 3

Output:
10

(Explanation: Subarrays are [1], [2], [3], [1, 2], [2, 3], [1, 2, 3]. Minimums are 1, 2, 3, 1, 2, 1. Sum = 1+2+3+1+2+1 = 10)

Sample Test Case 2

Input:
4
3 1 2 4

Output:
17

(Explanation: Subarrays and their mins: [3]=3, [1]=1, [2]=2, [4]=4, [3,1]=1, [1,2]=1, [2,4]=2, [3,1,2]=1, [1,2,4]=1, [3,1,2,4]=1. Sum = 17)

Problem 10: Sparse Adjacency Matrix to Multi-Linked List (Undirected Graph)

Problem Description

In this question, you are supposed to convert a symmetric sparse adjacency matrix of an undirected graph into a specialized multi-linked list.

- A sparse matrix is a matrix where most elements are zero, making a standard 2D array space-inefficient.

The structure of a specialized multi-linked list uses a "grid-like" linked approach where each node contains a vertical pointer (down) and a horizontal pointer (right).

The vertical backbone must contain all vertices from 0 to V-1 in ascending order. Each vertex in this backbone then branches out horizontally to list every vertex it is connected to.

Input Format:

- An integer **V** representing the number of vertices.
- A **V x V** symmetric adjacency matrix (0s and 1s).

Output Format:

- The output must visualize the vertical progression and the horizontal connections.
- Vertical links are represented by | and horizontal links by ->.
- If a node has no horizontal connections, only the vertex ID is shown.

Sample Test Case 1

Input:

```
3
0 1 1
1 0 0
1 0 0
```

Output:

```
0 -> 1 -> 2
|
1 -> 0
|
2 -> 0
```

Sample Test Case 2

Input:

```
4
0 1 0 1
1 0 1 0
0 1 0 0
1 0 0 0
```

Output:

```
0 -> 1 -> 3
|
1 -> 0 -> 2
|
2 -> 1
|
3 -> 0
```