COL106 : Data Structures and Algorithms, Semester II 2024-25
Practice Programming Questions on Arrays

Instructions

- Please use the following questions as practice questions for learning about Array datastructure.
- The questions with * next to them should be attempted during the lab sessions, and **your solutions must be uploaded on moodlenew**. Note that your submissions will not be evaluated, but will be used as a mark of your attendance. We will filter out all the submissions that are not from the lab workstations. So do not use your laptops for submitting the programs.

1 Questions

1. *** Find Maximum Element in an Array**

   Given an array `arr` of integers, find and return the maximum element in the array.

   Implementation: Construct a class `FindMax` containing a publicly accessible method `int findMax(int[] arr)`. This method should return the maximum value in the array.

   Example:

   - **Input:**
     `arr = {1, 3, 2, 5, 4}`
   - **Output:**
     `5`

2. *** Check If Array Is Sorted**

   Given an array `arr` of integers, check whether the array is sorted in non-decreasing order.

   Implementation: Construct a class `SortUtils` containing a publicly accessible method `boolean isSorted(int[] arr)`. This method should return `true` if the array is sorted, otherwise `false`.

   Example:

   - **Input:**
     `arr = {1, 2, 3, 4, 5}`
   - **Output:**
     `true`
   - **Input:**
     `arr = {5, 3, 4, 1, 2}`
   - **Output:**
     `false`

3. *** Sort an Array with atmost Three Values Without Using Sorting**

   You are given an array consisting of only three distinct types of elements. A specific order is defined between these three types, and the task is to rearrange the array such that all elements are sorted in the correct order. You must accomplish this without using any standard sorting algorithm.

   Constraints:

   - The array contains exactly three types of elements.
   - The order of the three types is predefined (e.g., Type1 $<$ Type2 $<$ Type3).
   - Array length: $1 \leq N \leq 10^6$.

   Accepted Solution:

   - **Time Complexity:** $O(N)$
   - **Space Complexity:** $O(1)$

   Example:

   - **Input:**
     $$\text{array} = \{2, 1, 3, 2, 1, 3, 2\}$$
   - **Output:**
     $$\{1, 1, 2, 2, 2, 3, 3\}$$
   - **Explanation:** The elements are sorted in the order $1 < 2 < 3$ using a single pass of the array.

4. **Sudoku Validation**

   Given a partially filled 9x9 Sudoku board, determine if it is valid. A Sudoku board is valid if:

   - Each row contains the digits 1-9 without repeating.
   - Each column contains digits 1-9 without repeating.
   - Each of the nine 3x3 sub-boxes contains the digits 1-9 without repeating.

   The Sudoku board may contain empty cells represented by the character `'.'`.

   Implementation: Construct a class `SudokuValidator` containing a publicly accessible method `boolean isValidSudoku(char[][] board)`. This method should return `true` if the board is a valid Sudoku, otherwise `false`.

   Example:

   - **Input:**

5	3	.	.	7	.	.	.	.
6	.	.	1	9	5	.	.	.
.	9	8	.	.	.	.	6	.
8	.	.	.	6	.	.	.	3
4	.	.	8	.	3	.	.	1
7	.	.	.	2	.	.	.	6
.	6	.	.	.	.	2	8	.
.	.	.	4	1	9	.	.	5
.	.	.	.	8	.	.	7	9

   - **Output:**
     `true`

   Note:

- The board size is always 9x9.
- The board may contain digits ('1'-'9') and empty cells ('.').

5. * **Maximum Subarray Sum**

Given an array of integers, find the maximum sum of a contiguous subarray within the array.

Example:

- **Input:**

$$\text{arr} = \{2, -1, 3, 4, -6, 7, -9, 8\}$$

- **Output:**

$$9$$

- **Explanation:** The subarray $\{2, -1, 3, 4, -6, 7\}$ has the maximum sum of 9.

Constraints:

- $1 \leq N \leq 10^6$, where N is the size of the array.
- Array elements can be both positive and negative integers.

Accepted Solution:

- **Time Complexity:** $O(N)$
- **Space Complexity:** $O(1)$

6. **Game of Life Simulation**

The "Game of Life" is a simulation game played on a 2D grid of size $m \times n$, where each cell has a state:

- 1: Live state.
- 0: Dead state.

Each cell interacts with its eight neighbors and decides its next state based on the following rules:

(a) Any live cell with fewer than two live neighbors dies, as if caused by under-population.
(b) Any live cell with two or three live neighbors lives on to the next generation.
(c) Any live cell with more than three live neighbors dies, as if by over-population.
(d) Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Input:

- A 2D array of size $m \times n$, where each element is either 0 (dead) or 1 (live).

Output:

- A 2D array of the same size after applying the Game of Life rules for one generation.

Constraints:

- $1 \leq m, n \leq 100000$
- $m \cdot n \leq 200000$

Accepted Solution:

- **Time Complexity:** $O(m \cdot n)$

- **Space Complexity:**
  - $O(m \cdot n)$ for the basic solution.
  - $O(1)$ (Bonus) for the optimized solution.

7. **ASCII Art Generation**

   You might have come across the statement that inside computers, images are stored as just 3-dimensional arrays of integers (or 2-dimensional arrays of 3-tuples). Given an image, you can load its matrix representation and perform certain operations on it. We wish to create an ASCII art generator similar to `https://ascii-generator.site/`.

   Steps:

   - **Resize the image**: To a width of 100 pixels and a proportional height i.e. maintaining the same aspect ratio.
   - **Load the Image Matrix**: Load the image into a 2D array where each element represents the RGB values of a pixel. The RGB values should be stored as hexadecimal values, with each pixel represented by three components: red (r), green (g), and blue (b). Check the implementation section for help.
   - **Calculate Average Brightness**: For each pixel in the image, calculate the average brightness value. The average brightness for a pixel is computed using the formula: average brightness = $\frac{r+g+b}{3}$ where r, g, and b are the red, green, and blue components of the pixel, respectively. This value will range from 0 (darkest) to 255 (brightest).
   - **Map Average Brightness to ASCII Character**: For each pixel, assign an ASCII character from the provided string `pixel_ascii_map` based on its average brightness. The string `pixel_ascii_map` contains a range of ASCII characters sorted from light to dark, corresponding to brightness levels. A brighter pixel will be assigned a lighter ASCII character, while a darker pixel will be assigned a darker character. The string `pixel_ascii_map` is as follows:
     `pixel_ascii_map = "‘^ ;Ii~+_-\?\]\[1)(|\/tfjrxnuvczXYUJCLQOOZmwqpdbkhao*#MW&8%\$"` The mapping will be done proportionally, so the brightness values between 0 and 255 will be mapped to characters in the string, with the darkest pixel corresponding to the first character and the brightest pixel corresponding to the last character.

   Implementation: Construct a class `ASCIIArt` containing a publicly accessible method `String[] generateASCIIArt(string image)`. The method should take a 2D array of RGB pixel values as input and return an array of strings representing the ASCII art. Each string in the array corresponds to a row of ASCII characters. You can import `javax.imageio.ImageIO` and use `ImageIO.read()` to read the input file and convert it into a `BufferedImage` object. Look up the various methods provided for `BufferedImage` class to find out which ones you would need to use.

   Note: The output will differ based on font used in your terminal or the text-editor where you print it. Use a monospaced font like Consolas or Courier New and reduce the font-size enough to accomodate one entire string from the array in a single row. An example of the kind of output you can expect is given in Fig 1

8. **Tuples with Constraints on Three Sorted Arrays**

   You are given three sorted integer arrays A, B, and C, each of length N, and an integer D. Your task is to find the number of tuples of distinct indices $\langle i, j, k \rangle$ such that:

   $$|A[i] - B[j]| \leq D, \quad |B[j] - C[k]| \leq D, \quad |C[k] - A[i]| \leq D$$

   Example:

Figure 1: The original image and the corresponding ASCII Art generated

- **Input:**

$$A = \{1, 2, 3\}, \quad B = \{1, 2, 4\}, \quad C = \{2, 2, 5\}, \quad D = 1$$

- **Output:**

12

- **Explanation:**
  The possible tuples are:

  $\{0, 0, 0\}, \{0, 0, 1\}, \{0, 1, 0\}, \{0, 1, 1\}, \{1, 0, 0\}, \{1, 0, 1\}, \{1, 1, 0\}, \{1, 1, 1\}, \{1, 1, 0\}, \{1, 1, 1\}, \{2, 1, 0\}$ and $\{2, 1, 1\}$

Constraints:

- $1 \leq N \leq 10^5$
- Array elements are integers within the range $[-10^9, 10^9]$.
- $0 \leq D \leq 10^9$.

Accepted Solution:

- **Time Complexity:** $O(N)$
- **Space Complexity:** $O(N)$

9. **Range Minimum Query (RMQ)**

   You are given an array of integers a of size n, and q queries. Each query is represented by two integers ℓ and r $(1 \leq \ell \leq r \leq n)$.

   For each query, your task is to find the minimum value in the subarray:

$$a[\ell], a[\ell + 1], \ldots, a[r].$$

   Input Format:

- The first line contains an integer n ($1 \leq n \leq 10^5$), the size of the array.
- The second line contains n space-separated integers, representing the array a, where $-10^9 \leq a[i] \leq 10^9$.
- The third line contains an integer q ($1 \leq q \leq 10^5$), the number of queries.
- The next q lines each contain two integers ℓ and r ($1 \leq \ell \leq r \leq n$), representing a query.

Output Format: For each query, output a single integer — the minimum value in the range $[a[\ell], a[\ell+1], \ldots, a[r]]$.

Constraints:

- $1 \leq n, q \leq 10^5$
- $-10^9 \leq a[i] \leq 10^9$ for all $1 \leq i \leq n$
- $1 \leq \ell \leq r \leq n$

Expected Complexity:

- Preprocessing Time: $O(n \log n)$
- Space Complexity: $O(n \log n)$
- Query Time: $O(1)$ per query

Example:

Input:

```
5
1 3 -2 8 -7
3
1 3
2 4
1 5
```

Output:

```
-2
-2
-7
```

Explanation:

Query	Subarray and Minimum
$(1, 3)$	$[1, 3, -2]$, Minimum: -2
$(2, 4)$	$[3, -2, 8]$, Minimum: -2
$(1, 5)$	$[1, 3, -2, 8, -7]$, Minimum: -7