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

February 2025

Instructions

- Please use the following questions as practice questions for Sorting Algorithms.
- 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.

Questions

1. *** Sort an array in waveform.**

   Given an unsorted array of integers, sort the array into a wave array. An array $arr[0..n-1]$ is sorted in wave form if:

   $$arr[0] \geq arr[1] \leq arr[2] \geq arr[3] \leq arr[4] \geq \dots$$

   There can be multiple valid waveforms.

   Example: Input:

   $$arr[] = \{10, 5, 6, 3, 2, 20, 100, 80\}$$

   Output:

   $$arr[] = \{10, 5, 6, 2, 20, 3, 100, 80\}$$

   Explanation: The elements are arranged in an alternating high-low pattern.

   - Solve the problem in $O(n \log n)$ time complexity.
   - Optimize your solution to $O(n)$ time complexity and $O(1)$ space complexity.

2. *** Inversion Count using Merge Sort.**

   Given an array $arr[]$ of size n, count the number of inversions in the array.An inversion is defined as a pair (i, j) where $i < j$ and $arr[i] > arr[j]$.

   Example: Input:

   $$arr[] = \{3, 1, 2\}$$

   Output:

   $$\text{Number of inversions} = 2$$

   Explanation: The two inversions are: $(3, 1), (3, 2)$.

- Solve this problem efficiently using the merge sort algorithm.
- Your solution should run in $O(n \log n)$ time complexity.

3. *** Beautiful Array**

   Consider an array of n distinct integers, $arr = [a_0, a_1, \ldots, a_{n-1}]$. George can swap any two elements of the array any number of times.

   An array is called **beautiful** if the sum of absolute differences between adjacent elements is minimized, i.e.,

   $$\sum_{i=1}^{n-1} |arr[i] - arr[i-1]|$$

   is minimal.

   Given the array arr, determine and return the **minimum number of swaps** required to make the array beautiful.

   Example
   Input

   $$arr = [7, 15, 12, 3]$$

   Process One minimal array is $[3, 7, 12, 15]$. To achieve this, George performed the following swaps:

Swap	Result
$3 \leftrightarrow 7$	$[3, 15, 12, 7]$
$7 \leftrightarrow 15$	$[3, 7, 12, 15]$

   It took **2 swaps** to make the array beautiful. This is the minimal number of swaps required.

4. **Insertion Sort Anaylsis** Insertion Sort is a simple sorting technique. However, for large arrays, performing an insertion sort directly may take too long. Instead, we need an efficient way to calculate the number of shifts that occur when sorting an array using insertion sort.

   If $k[i]$ represents the number of elements over which the i^{th} element of the array has to shift, then the total number of shifts required to sort the array is given by:

   $$k[1] + k[2] + \cdots + k[n]$$

Example

Input

$$arr = [4, 3, 2, 1]$$

Process

Array	Shifts
$[4, 3, 2, 1]$	
$[3, 4, 2, 1]$	1
$[2, 3, 4, 1]$	2
$[1, 2, 3, 4]$	3
Total Shifts	$1 + 2 + 3 = 6$

Given the array of the integers return the the number of shifts required to sort the array

5. **Sort using Radix Sort** Given an array of numbers of size n. It is also given that the array elements are in the range from 0 to $n^2 - 1$. Sort the given array in linear time.

   Example:

   Input:

   $$\text{arr}[] = \{3, 1, 2\}$$

   Output:

   $$\text{arr}[] = \{1, 2, 3\}$$

   Hint: Convert the numbers to base n from base 10 and then proceed digit by digit from least significant digit to most significant. You may refer to the Radix Sort implementation for better understanding.

   Note: The approach can further be generalized to sort an array of numbers within the range 0 to $n^k - 1$ in $O(nk)$ time complexity.

6. **Max Difference:** Given an integer array nums, return the maximum difference between two successive elements in its sorted form. If the array contains less than two elements, return 0.

   Example:

   Input:

   $$\text{arr} = \{3, 6, 9, 1\}$$

   Output: 3

   Explanation: After sorting, we get:

   $$\text{arr} = \{1, 3, 6, 9\}$$

   The difference is calculated as:

   $$9 - 6 = 3, \quad 6 - 3 = 3, \quad 3 - 1 = 2$$

   Thus, the maximum difference is 3.

   - You are not allowed to use any in-built sorting function.
   - Your solution should run in $O(n)$ time complexity.

7. * **Sorting Three Values:** Given an array nums of length n, where each element is either 0, 1, or 2, sort the array in a single pass using constant additional space.

   Example:

   Input:

   $$\text{nums} = \{2, 0, 2, 1, 1, 0\}$$

   Output:

   $$\text{nums} = \{0, 0, 1, 1, 2, 2\}$$

   - You are not allowed to use any built-in sorting function.
   - Your solution should run in $O(n)$ time complexity and use $O(1)$ additional space.

8. **Sorting with Reversing:** You are given an array arr of length n, which is a permutation of the numbers from 1 to n. You can perform the following operation any number of times:

   - Choose an integer k where $1 \leq k \leq n$.

- Reverse the subarray $\text{arr}[0 \ldots k-1]$.

A permutation of size n is an array containing each number from 1 to n exactly once. Your task is to return a sequence of k-values that, when applied in order, sorts arr in ascending order.

Example:

Input:

$$\text{arr} = \{3, 2, 4, 1\}$$

Output:

$$\text{Answer} = \{4, 2, 4, 3\}$$

Explanation: The given array is $\{3, 2, 4, 1\}$. We sort it using the following steps:

(a) Choose $k = 4$, reverse the subarray $[3, 2, 4, 1] \rightarrow [1, 4, 2, 3]$.

(b) Choose $k = 2$, reverse the subarray $[1, 4] \rightarrow [4, 1, 2, 3]$.

(c) Choose $k = 4$, reverse the subarray $[4, 1, 2, 3] \rightarrow [3, 2, 1, 4]$.

(d) Choose $k = 3$, reverse the subarray $[3, 2, 1] \rightarrow [1, 2, 3, 4]$.

After these operations, the array is sorted, and the sequence of k-values is $\{4, 2, 4, 3\}$.

Time Complexity: $O(n^2)$