DSA (CSE102) Lab 2: Sorting Algorithms

17-01-25, Time: 2 Hours, Total Points: 100

Instructions

- All code must be written in **C** only.
- There will be no partial marks unless specified.
- You have to submit only the `.c` files. **Do not submit .exe or binary files. Submission of binary files will result in 0 marks in the LAB.**
- Problem 2 must be done using **Insertion sort**. Using any other algorithm will result in a 50% marks deduction.
- You can use any algorithm for Problem 1.
- Problem 3 must be done using **Merge Sort** algorithm (For **Section A**) and **Selection Sort or Bubble sort** algorithm (For **Section B**). Using any other algorithm will result in a 50% marks deduction.
- You cannot use any **inbuilt function** , any function you want to use , you should implement by scratch
- You can use any **Documentation** . for c programming language . https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html

Problem 1: Merge Intervals – One Piece Adventure (40 Points)

In the world of *One Piece*, Luffy and his Straw Hat Pirates discover a treasure map that contains several time intervals, each representing the time during which certain islands are accessible. However, the intervals on the map are scattered and overlapping, making it difficult to plan their journey efficiently.

Your task is to help the crew merge overlapping intervals so that they can plan their journey effectively without any redundancy.

Input Format

The input consists of:

- An array of intervals, where each interval is represented as a pair [start, end].
- Each interval satisfies the condition start $\leq$ end.

Output Format

The output should be:

- An array of merged intervals, where overlapping intervals are combined into a single interval.

Example

Input:

```
intervals = [[1, 4], [2, 5], [7, 9], [8, 10], [11, 13]]
```

Output:

```
Merged Intervals = [[1, 5], [7, 10], [11, 13]]
```

Test Cases

1. **test Case 1:**

```
Input: [[1, 10], [2, 9], [3, 8]]
Output: [[1, 10]]
```

2. **test Case 2:**

```
Input: [[1, 3], [4, 6], [7, 9]]
Output: [[1, 3], [4, 6], [7, 9]]
```

3. **test Case 3:**

```
Input: [[1, 4], [2, 5], [7, 8]]
Output: [[1, 5], [7, 8]]
```

4. **test Case 4:**

```
Input: [[1, 2], [2, 3], [5, 6], [6, 7]]
Output: [[1, 3], [5, 7]]
```

5. **test Case 5:**

```
Input: [[5, 10]]
Output: [[5, 10]]
```

Problem 2: Sorting Chaos (30 Points)

You must use Insertion sort for this problem. Using any other algorithm will result in a 50% marks deduction.

At Odyssey 2024, IIIT-Delhi's iconic fest, one of the highlights is the "Sorting Chaos" challenge. The organizers have prepared a leaderboard representing the initial scores of participants in a completely random order. Participants must sort the leaderboard to rank everyone properly. However, the organizers are interested in identifying the participant whose position changed the most after sorting.

Your task is to write a program that determines the participant with the largest shift in their index between the original and sorted leaderboard.

Input Format

- An integer array `nums` of size n, where $n \geq 0$, representing the scores on the leaderboard.

Output Format

- A single integer representing the largest index shift for any participant.

Example

Input:

```
nums = [40, 10, 20, 50, 30]
```

Output:

```
3
```

Explanation:

- The sorted array is $[10, 20, 30, 40, 50]$. - Comparing indices:

- 40 moves from index 0 to 3: Shift $|3 - 0| = 3$.
- 10 moves from index 1 to 0: Shift $|0 - 1| = 1$.
- 20 moves from index 2 to 1: Shift $|1 - 2| = 1$.
- 50 moves from index 3 to 4: Shift $|4 - 3| = 1$.
- 30 moves from index 4 to 2: Shift $|2 - 4| = 2$.

- The largest index shift is **3**.

Input:

```
nums = [10, 20, 30]
```

Output:

```
0
```

Explanation:

The array is already sorted, so no elements change position. The largest shift is 0.

Test Cases

1. **test Case 1:**

```
Input: [1, 2, 3, 4, 5]
Output: Largest index shift: 0
```

2. **test Case 2:**

```
Input: [7, 7, 7, 7]
Output: Largest index shift: 0
```

3. **test Case 3:**

```
Input: [40, 10, 20, 50, 30]
Output: Largest index shift: 3
```

4. **test Case 4:**

```
Input: [5, 4, 3, 2, 1]
Output: Largest index shift: 4
```

5. **test Case 5:**

```
Input: [10]
Output: Largest index shift: 0
```

Problem 3: The Library Catalog Sorting (30 Points)

You must use Merge Sort algorithm (For Section A) and Selection Sort or Bubble sort algorithm (For Section B)for this problem. Using any other algorithm will result in a 50% marks deduction.

In the prestigious IIIT-Delhi library, there's an old catalog of books, each assigned a unique ID number. The catalog has been getting more disorganized over the years, with books scattered randomly. The catalog manager, in a bid to restore order, has tasked you with a special sorting challenge.

The catalog has a peculiar system of arrangement:

- The books at **odd positions** (1st, 3rd, 5th, ...) are categorized separately.

- The books at **even positions** (2nd, 4th, 6th, ...) are categorized separately.

Your job is to help the catalog manager restore the order by sorting the books at odd and even positions independently, while maintaining the overall structure of the catalog.

Input Format

An integer array *arr* representing the catalog, where each integer corresponds to a book ID.

Output Format

Return an array where the books at odd positions are sorted and the books at even positions are sorted, but their relative positions in the catalog (odd/even) should stay the same.

Example

Input:

$$arr = [8, 2, 7, 5, 4, 6]$$

Output:

$$[4, 2, 7, 5, 8, 6]$$

Explanation:

- Books at odd positions (1st, 3rd, 5th) are $[8, 7, 4]$. After sorting them, we get $[4, 7, 8]$.
- Books at even positions (2nd, 4th, 6th) are $[2, 5, 6]$. After sorting them, we get $[2, 5, 6]$.
- The final catalog array is $[4, 2, 7, 5, 8, 6]$.

Constraints

- You must use the **Merge Sort** algorithm (For **Section A**) and **Selection Sort or Bubble sort** algorithm (For **Section B**) algorithm to sort the books in odd and even positions.
- The original relative positions of odd and even elements must be preserved in the final result.

Test Cases

1. **test Case 1:**

```
Input: [10]
Output: [10]
```

2. **test Case 2:**

```
Input: [5, 1]
Output: [5, 1]
```

3. **test Case 3:**

```
Input: [8, 2, 7, 10, 4, 6]
Output: [4, 2, 8, 6, 7, 10]
```

4. **test Case 4:**

```
Input: [9, 8, 7, 6, 5, 4]
Output: [5, 4, 7, 6, 9, 8]
```

5. **test Case 5:**

```
Input: [3, 3, 3, 3, 3]
Output: [3, 3, 3, 3, 3]
```