Algorithms Laboratory (CS29203)
Assignment 3: Greedy Algorithms
Department of CSE, IIT Kharagpur

29th August 2024

Question-1

(50 points)
CDC placements are scheduled to be taking place in the campus soon, and there needs to be arrangements of student registrations. Due to some reasons, there are shortage of volunteers to conduct registrations and the institute needs to have an estimate of how many volunteers are needed for smoothly performing the registration. One volunteer can perform the registration of one student at a time (and can start the next registration immediately after the current one). Based on the student background data, the exact time needed for registration of individual students have been computed by CDC, and they have provided the starting and ending time of individual student registration timings. Let us consider the starting time of all the student registration is denoted by an array `start[ ]` and the ending time by the array `end[ ]`. Assume that timings are denoted as PM. So if `start = [2.00, 2.34, 3.10]` and `end = [2.32, 2.56, 3.30]`, that means there are 3 students with registration starting time as 2.00 PM, 2.34 PM and 3.10 PM, and respective ending times are 2.32 PM, 2.56 PM and 3.30 PM.

Given the starting and ending time of registration, your task is to find out the **minimum** number of volunteers needed to perform the registration process without having any delay.

For example, consider `start = [2.00, 2.10, 3.00, 3.20, 3.50, 5.00]` and `end = [2.30, 3.40, 3.20, 4.30, 4.00, 5.20]`. Then the minimum number of volunteers needed is 2 because of the following configuration:

- Volunteer 1 serves registration at 2.00 PM
- Volunteer 2 serves registration at 2.10 PM
- Volunteer 1 finishes a registration at 2.30 PM
- Volunteer 1 serves registration at 3.00 PM
- Volunteer 1 finishes a registration at 3.20 PM
- Volunteer 1 serves registration at 3.20 PM
- Volunteer 2 finishes a registration at 3.40 PM
- Volunteer 2 serves registration at 3.50 PM
- Volunteer 2 finishes a registration at 4.00 PM
- Volunteer 1 finishes a registration at 4.30 PM
- Volunteer 1 serves registration at 5.00 PM
- Volunteer 1 finishes a registration at 5.20 PM

To get the full credit, your algorithm should not exceed $O(n \log n)$ time.

Example:

```
(Input) Enter the number of students n: 10
        Enter the start time array: 2.00, 2.07, 2.10, 3.05, 3.06, 3.07, 3.10, 3.20, 3.50, 4.12, 5.00, 5.15
        Enter the end time array: 2.30, 2.55, 3.40, 3.09, 3.08, 3.15, 3.17, 4.30, 4.00, 4.32, 5.20, 5.27

(Output) Minimum number of volunteers needed is 4
```

Question-2

(50 points)
You are taking an online course of advanced algorithms from *Coursera*. There are n number of homework assignments in the course. Each assignments have multiple parts associated with it. The submission portal of the assignments have some rules that you have to maintain. These rules are as follows:

- Every week you are allowed to submit exactly one part of one assignment only. And you must perform submission every week.
- You are not allowed to submit two parts of the same assignment for two consecutive weeks.

The submission portal stops accepting assignments in one of the following two cases. *First*, when all the parts of all the assignments are submitted. *Second*, submission of the remaining parts of the assignments will violate the above mentioned rules. Note that due to these rules, you may not able to submit all parts of all assignments of the course.

Given an integer array `parts[ ]` where `parts[i]` denotes the number of parts of the i-th assignment, your task is to find the **maximum** number of weeks you can work on the assignments without violating the above mentioned rules. Assume that the assignments are numbered from 0 to $n-1$.

For example, consider `parts = [1,2,3]`. Then the total number of weeks needed is 6 due to the following possible scenario:

- Week 1: submit a part of assignment 0
- Week 2: submit a part of assignment 2
- Week 3: submit a part of assignment 1
- Week 4: submit a part of assignment 2
- Week 5: submit a part of assignment 1
- Week 6: submit a part of assignment 2

To get the full credit, your algorithm should not exceed $O(n)$ time.

Example:

```
(Input) Enter the number of assignments: 3
        Enter the array parts[ ]: 5  2  1

(Output) total number of weeks needed is 7

Explanation: Week 1: submit a part of assignment 0
             Week 2: submit a part of assignment 1
             Week 3: submit a part of assignment 0
             Week 4: submit a part of assignment 1
             Week 5: submit a part of assignment 0
             Week 6: submit a part of assignment 2
             Week 7: submit a part of assignment 0
Note: you can't work on the last remaining part of assignment 0 on 8th week since this will violate the rule.
```