Lab Assignment 6

1) Minimum Number of Platforms Required for a Railway/Bus Station

Given the arrival and departure times of all trains that reach a railway station, the task is to find the minimum number of platforms required for the railway station so that no train waits.
We are given two arrays that represent the arrival and departure times of trains that stop.

- **Approach:** The idea is to consider all events in sorted order. Once the events are in sorted order, trace the number of trains at any time keeping track of trains that have arrived, but not departed.

For example, consider the above example.

arr[] = {9:00, 9:40, 9:50, 11:00, 15:00, 18:00}
dep[] = {9:10, 12:00, 11:20, 11:30, 19:00, 20:00}

All events are sorted by time.
Total platforms at any time can be obtained by subtracting total departures from total arrivals by that time.

Time	Event Type	Total Platforms Needed at this Time
9:00	Arrival	1
9:10	Departure	0
9:40	Arrival	1
9:50	Arrival	2
11:00	Arrival	3
11:20	Departure	2
11:30	Departure	1
12:00	Departure	0
15:00	Arrival	1
18:00	Arrival	2
19:00	Departure	1
20:00	Departure	0

Minimum Platforms needed on railway station = Maximum platforms needed at any time = 3

2) **Greedy Algorithms**

Greedy Algorithms: At every iteration, you make a myopic decision. That is, you make the choice that is best at the time, without worrying about the future. And decisions are irrevocable; you do not change your mind once a decision is made.

Problem Description:

With all these definitions in mind now, recall the music festival event scheduling problem. You have a pass to your favorite music fest, and want to attend/watch as many performances as possible. You like all the artists/bands equally; you just want to watch as many live shows as possible. But once you go to a performance, you cannot leave half way through to attend another one.

Clearly every performance has a start and a finish time, and you are given the schedule ahead of time. As we saw in class, we can think of each performance as a time interval (from its start time until it is over), and we can abstract this problem as the interval scheduling problem (ISP), defined below more formally:

Interval Scheduling Problem:

Input: A list of intervals I = {I 1 , I 2 , . . . , I n }, where each interval I i is defined by two integers I i = (s i , f i) with s i < f i . Two intervals I i , I j are compatible, i.e. disjoint, if they do not intersect (f i < s j or s i < f j).
Output: A maximum subset of pairwise compatible (disjoint) intervals in I.

A number of greedy heuristics we tried in class failed quickly and miserably. Heuristics such as the Greedy Early Start Time algorithm (sorting the intervals by nondecreasing start time s 1 ≤ s 2 ≤ . . . ≤ s n), or the Greedy by Duration (sorting the intervals by nondecreasing duration (f 1 − s 1) ≤ (f 2 − s 2) ≤ . . . ≤ (f n − s n)) etc, but the Early Finish Time greedy algorithm (EFT) seemed to work, and we proved it is indeed optimal!

Algorithm: EFT
Input: A list of intervals I = {I 1 , I 2 , . . . , I n }, where I i = (s i , f i) with s i < f i for all 1 ≤ i ≤ n
Output: A maximum subset of pairwise compatible intervals in I

Algorithm 1 EFT

Input: A list of intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, where $I_i = (s_i, f_i)$ with $s_i < f_i$ for all $1 \leq i \leq n$
Output: A maximum subset of pairwise compatible intervals in $\mathcal{I}$

```
1: Sort the input list I of intervals in nondecreasing order of finish time.
2: f = 0
3: S_0 = ∅
4: for i ← 1 to n do
5:     if f < s_i then                          ▷ I_i is compatible with S_i
6:         S_i = S_{i−1} ∪ {I_i}
7:         f = f_i
8:     end if
9: end for
10: return S_n
```