Activity Selection Problem

Given "N" number of activities with their starting and finishing time and the task is to **schedule maximum number of compatible activities**.

Compatible Activities: Two activities say "a_i" and "a_j" are said to be compatible if they are non-overlapping i.e., the starting time of one of the activities is greater than the finishing time of the other activity e.g., starting time of "a_j" is greater than the finishing time of "a_i".

Practical Example: Assume that we want to schedule maximum number of lectures in a single class-room where, starting and finishing time of each lecture is different e.g.,

Lectures	Start Time	Finish Time
OOP	8:00	10:00
DSA	9:00	12:30
PF	10:30	11:30
Algorithms	11:00	12:00
OS	8:30	9:30

What will be the greedy choice to select the activities from the given set of activities?

Greedy Choice Options:

1: Minimum duration activity first? If the activity with starting time greater than all other activities in the set has the minimum duration, then that activity will be selected and, in that case, only one activity can be scheduled because no other activity starts after the finishing time of that activity, so this greedy approach is not going to produce the optimal solution.

2: Minimum (Earliest) starting time activity first: There might be a possibility that finishing time of the activity with earliest starting time is greater than all other activities so we will be able to schedule only one activity in that case. So, this approach is also not going to produce an optimal solution.

3: Minimum (Earliest) finishing time activity first:

4: Maximum starting time activity first:

Let's explore these two options first and then we will check whether the solution produced by these greedy choices is optimal or not.

In greedy algorithms we also need to prove the correctness of algorithm.

Why? We didn't check the correctness of dynamic programming algorithms then why we need to check the correctness of greedy algorithms? Basically, in dynamic programming first we design a brute force solution and then select the best solution from all available solutions, so it is understood that when the best solution from all possible solutions is picked it will definitely be optimal but in greedy algorithms the solution is not going to explore all possible solutions, so we need to check the correctness of algorithm.

Greedy Algorithm for activity selection problem:

```
Given Data:
A[]: set of activities
s_time[]: starting time of activities
f_time[]: finishing time of activities
A_S_P(A[], s_time[], f_time[])
{
sort the arrays on the base of earliest finishing time i.e., in the ascending order of finishing time
   sort(A, s_time, f_time)
   vector sel_Act  //vector to store the selected activities.
   sel_Act.push_back(item[1])  //push the 1st item i.e.,(activity with earliest finishing time)
   count = 1   //variable to keep the count of selected activities. (We can also use vector size)

//since we want to compare the finishing time of selected activity with the starting time of
remaining available activities in the set to check the compatibility, so we also need to keep the
track of the index of last selected activity.
   k = 1      //index of last selected activity. it will be used in the process to determine the
compatibility of activities.

   for(i=2 to N)   //since first activity already selected so start from 2
   {
      if(s_time[i] > f_time[k])   //check for compatibility of the activity
      {
         sel_Act.push_back(items[i])
         k = i   //update the index of last selected activity.
         count++ //update the count of selected activities
      }
   }
   return (sel_Act, count)
}
//Time complexity:
Sorting + linear loop: O(N * log N) + O(N) = O(N * log N)
```

To prove the correctness of the greedy algorithm we need to check following two properties:

1: greedy choice property

2: optimal substructure

If our solution holds these two properties, then our algorithm is optimal.

What is greedy choice property and optimal substructure?

Basically, In greedy algorithms the decision taken by the algorithm is based on local optimal choice rather global optimal. (Graph example in class to explain local and global optimal choice)

Greedy choice property: Local optimal choice is global optimal

Optimal substructure: There exists optimal solution to each subproblem in the solution