An Activity Selection Problem

- An Activity Selection problem is the problem of scheduling a resource among several competing activity.
- A set $S = \{a_1, a_2, a_3 \dots a_n\}$ of n activities that wish to use a resource (classroom) which can serve only one activity at a time.
- Each activity a_i has a start time s_i and finish time f_i where $s_i < f_i$
- Two activities i and j are compatible if they donot overlap (if $s_i \geq f_j$ or $s_j \geq f_i$)
- In this, we wish to select a maximum size subset of mutually compatible activities.
- Assuming that activities are sorted in monotonically increasing order of finish time. $f_1 < f_2 < f_3 \dots f_n$
- Consider the following set of activities

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- a_3, a_9, a_{11} consist of mutually compatible activities.
- a_1, a_4, a_8, a_{11} is the largest subset of mutually compatible activities
- a_2, a_4, a_9, a_{11} is another largest set.

(AJAY RAWAT)

Greedy Choice

- Choose an activity that leaves the resources available for as many other activities as possible.
- Intuition is to choose the activity in S with the earliest finish time, since that would leave the resource available for as many of the activities that follow it as possible.
- Sort the input activities by increasing finishing time as $f_1 \leq f_2 \leq \cdots \leq f_n$.
- Since the activities are sorted in monotonically increasing order by finish time, the greedy choice is activity a_1.
- If we make the greedy choice, we have only one remaining subproblem to solve – finding activities that start after a_1 finishes.
- So all activities that are compatible with activity a_1 must start after a_1 finishes.

Recursive-Activity-Selector (s, f, k, n)

1. $m = k+1$
2. while $m \leq n$ and $s[m] < f[k]$ // find first activity in S_k to finish
3. $\quad m = m+1$
4. if $m \leq n$
5. $\quad$ return $\{a_m\} \cup$ Recursive-Activity-Selector (s, f, m, n)
6. else return $\emptyset$.

Space Complexity - The storage of start time, finish time and subset S all take $\Theta(n)$. The space requirement depends on sorting algo yet will not exceed $O(n)$. So overall complexity $O(n)$.

- Use a top-down greedy algorithm.

- Initial Call is Recursive-Activity-Selector$(S, f, 0, n)$

Time Complexity

- Assum 'n' input activities are already sorted, if not we can sort them in $O(n \log n)$ time.

- The Running time of Recursive Activity-Selector is $\Theta(n)$.

Iterative Greedy Algorithm

Greedy-Activity-Selector (S, f)

```
1- n = S.length
2- A = {a1}
3- K = 1
4- for m = 2 to n
5-     if s[m] ≥ f[K]
6-         A = A ∪ {am}
7-         K = m
8- return A
```

- Like recursive version, Greedy-Activity-Selector schedules a set of n activities in $\Theta(n)$ time.