GREEDY ALGORITHMS

→ Greedy algorithms solves problems by making the choice that seems best at the particular moment.

→ Some optimization problems have no efficient solutions but a greedy algorithm may provide a solution that is close to optimal

→ A greedy algorithm works if a problem having the following two properties:

1. Greedy choice property: A globally optimal solution can be arrived at by making a locally optimal solution. In other words, an optimal solution can be obtained by making "greedy" choices.

2. Optimal substructure: optimal solutions contain optimal sub solutions. In other words, solutions to subproblems of an optimal solution are optimal.

→ In dynamic programming we make a choice at each step, but the choice usually depends on the solution to the subproblems.

→ In greedy algorithm, we make whatever choice seems best at the moment and then solves the subproblem arising after the choice is made.

→ The dynamic programming problem solve in bottom-up manner, progressing from smaller subproblem to larger subproblems.

→ But greedy ~~star~~ algorithm usually progress in a top-down fashion (making one greedy choice after another).

→ Examples of greedy technique:

(1) Activity selection problem (ASP).

(2) Fractional knapsack problem (FKP).

(3) Hoffman code.

Steps for Greedy Programming

1. Determine the optimal substructure of the problem.
2. Develop a recursive solution.
3. prove that at any stage of recursion, one of the optimal choices is the greedy choice.
4. Show that all but one of the subproblems induced by having made the greedy choice are empty.
5. Develope the recursive algorithm that complements the greedy strategy.
6. Convert the recursive algorithm to an iterative algorithm.

Activity-Selection Problem

→ Suppose, we have a set $S = \{a_1, a_2, \ldots, a_n\}$ of n activities wish to use a resource which can be used by only one activity at a time.

→ Each activity a_i has a start time s_i and a finish time f_i, where $0 \leq s_i < f_i < \infty$

→ The selected activity a_i can be represented by half-open time interval $[s_i, f_i)$.

→ Activities a_i and a_j are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap i.e. a_i and a_j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$.

→ The activity-selection problem selects the subset of activities which are mutually compatible.

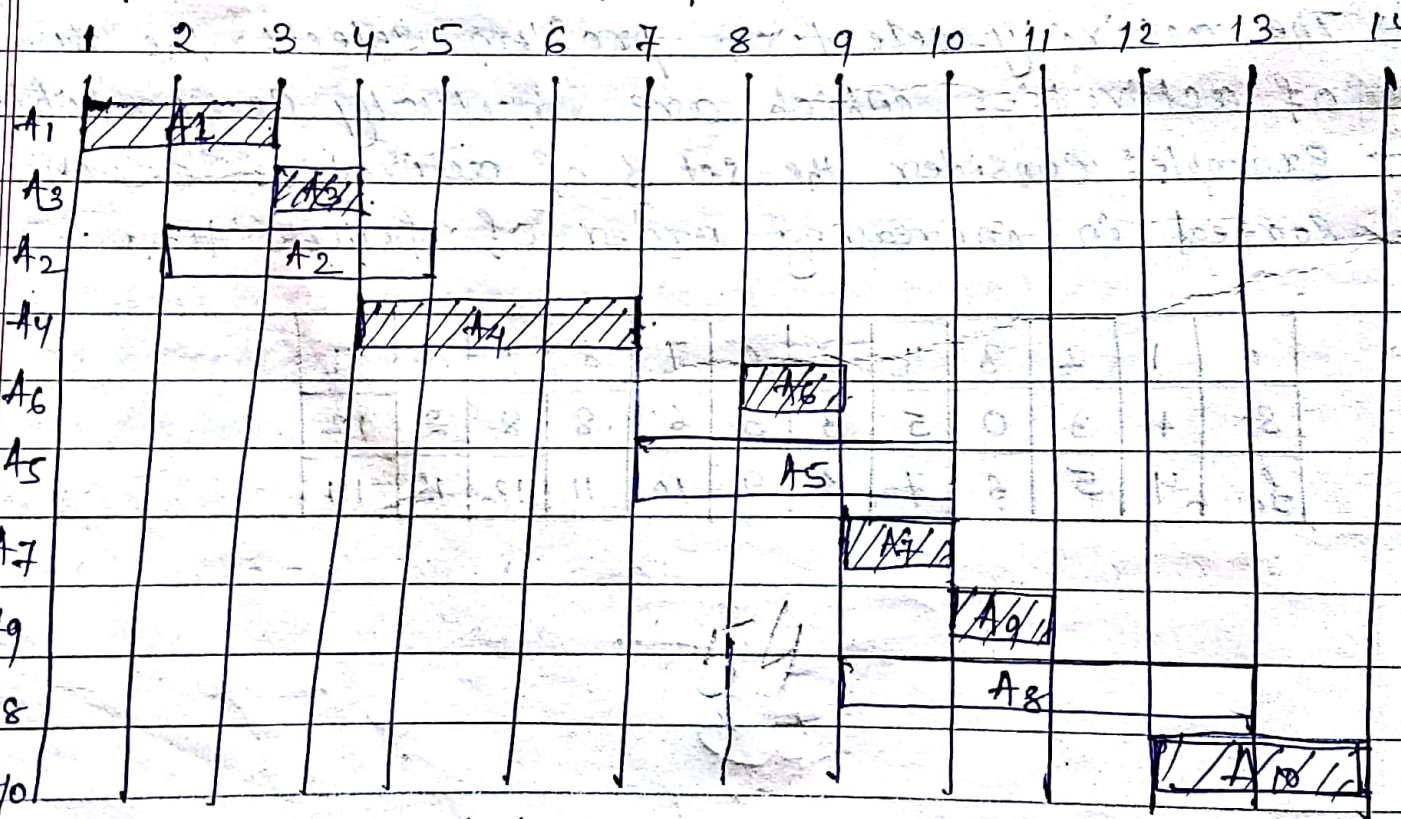
→ Example: Consider the set S of activities which are sorted in increasing order of finish time.

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	8	9	10	11	12	13	14

Example: Given 10 activities along with their start time and finish time as $S = \langle A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10} \rangle$, $s_i = \langle 1, 2, 3, 4, 7, 8, 9, 9, 11, 12 \rangle$, $f_i = \langle 3, 5, 4, 7, 10, 9, 11, 13, 12, 14 \rangle$.

Sol^n: Arrange the activities in increasing order of finishing time

	A_1	A_3	A_2	A_4	A_6	A_5	A_7	A_9	A_8	A_{10}
Start	1	3	2	4	8	7	9	11	9	12
Finish	3	4	5	7	9	10	11	12	13	14



Thus, the activity schedule is $\langle A_1, A_3, A_4, A_6, A_7, A_9, A_{10} \rangle$

→ A Recursive Greedy Algorithm :

RECURSIVE-ACTIVITY-SELECTOR (s, f, i, j)

1. $m \leftarrow i+1$
2. while $m < j$ and $s_m < f_i$ ▷ Find the first activity in S_{ij}.
3. do $m \leftarrow m+1$
4. if $m < j$
5. then return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, j)
6. else return $\emptyset$

Complexity : The running time of the call RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, n+1) is $\Theta(n)$ over all recursive calls, each activity is examined exactly once.

An iterative greedy algorithm:

→ Recursive algorithm works for any subproblems S_{ij} but we consider only subproblems for which $j = n+1$ i.e. subproblems that consists of the last activities to finish.

→ procedure GREEDY-ACTIVITY-SELECTOR is an iterative version where we assumes that the input activities are ordered by increasing finish time.

```
GREEDY-ACTIVITY-SELECTOR (s, f)
1. n ← length[s]
2. A ← {a1}
3. i ← 1
4. for m ← 2 to n
5.       do if sm ≥ fi
6.                then A ← A ∪ {am}
7.                     i ← m
8. return A.
```

→ Complexcity:

GREEDY-ACTIVITY-SELECTOR schedules a set of n activities in $\Theta(n)$ time assuming that the activities were sorted in increasing order of finish time.