Greedy Algorithm.

* An activity-selection problem.

Problem of scheduling several competing activities that require exclusive use of a common resources, with a goal of selecting a maximum-size set of mutually compatible activities.

Let, $S = \{a_1, a_2 \ldots \ldots a_n\} \rightarrow n$ activities.

where, a_i is i^{th} activity that wish to use resources such as lecture hall, which can serve only one activity at a time.

It has start time s_i and finish time f_i

where $0 \leq s_i \leq f_i < \infty$

Activity a_i takes place during the half-open time interval $[s_i, f_i)$.

Activities a_i & a_j are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

i.e. $s_i \geq f_j$ or $s_j \geq f_i$

In activity-selection problem, we wish to select a maximum-size subset of mutually compatible activities.

Assume, activities are sorted in monotonically increasing order of finish time.

$f_1 \leq f_2 \leq f_3 \ldots \ldots \leq f_{n-1} \leq f_n$

Example.

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

$\{a_3, a_9, a_{11}\}$

$\{a_1, a_4, a_8, a_{11}\}$ Largest subset.

$\{a_2, a_4, a_9, a_{11}\}$

We need to consider only one choice - the greedy choice - when we make a greedy choice, only one subproblem remains.

Making the greedy choice.

Only one remaining subproblem to solve.

Finding activities that start after a_1 finishes.

$s_1 < f_1$ and f_1 is the earliest finish time of any activity, and therefore no activity can have a finish time less than or equal to s_1.

Thus all activities that are compatible with activity a_1 must start after a_1 finishes.

Let, $S_k = \{a_i \in S : s_i \geq f_k\}$ be the set of activity that start after activity a_k finishes.

If we make the greedy choice of activity a_1, then S_1 remains as the only subproblem to solve.

optimal substructure tell us that if a_1 is in the optimal solution, then an optimal solution to the original problem consists of activity a_1 and all the activities in an optimal solution to the subproblem S_1.

Choosing an activity to put into the optimal solution and then solving the subproblem of choosing activities from those that are compatible with those already chosen.

A recursive greedy algorithm.

$s \rightarrow$ start time of activities
$f \rightarrow$ finish time of activities.
$S_k \rightarrow$ index k for subproblem S_k
$n \rightarrow$ size of original problem.

It returns a maximum-size set of mutually compatible activities in S_k.
we assume that the n input activities are already ~~sort~~ ordered by monotonically increasing finish time or it can be done in $O(n \lg n)$ time.

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)

```
m = k+1
while m ≤ n and s[m] < f[k]
      m = m+1
if m ≤ n.
      return {am} ∪ RECURSIVE-ACTIVITY-SELECTOR
                    (s, f, m, n)
else
   return ϕ
```

→ Video classroom for delivering online lectures.

→ Different teachers wants to book the classroom – the slot for each instructor i starts at $S(i)$ and finishes at $f(i)$

→ Slot may overlap, so not all bookings can be honoured.

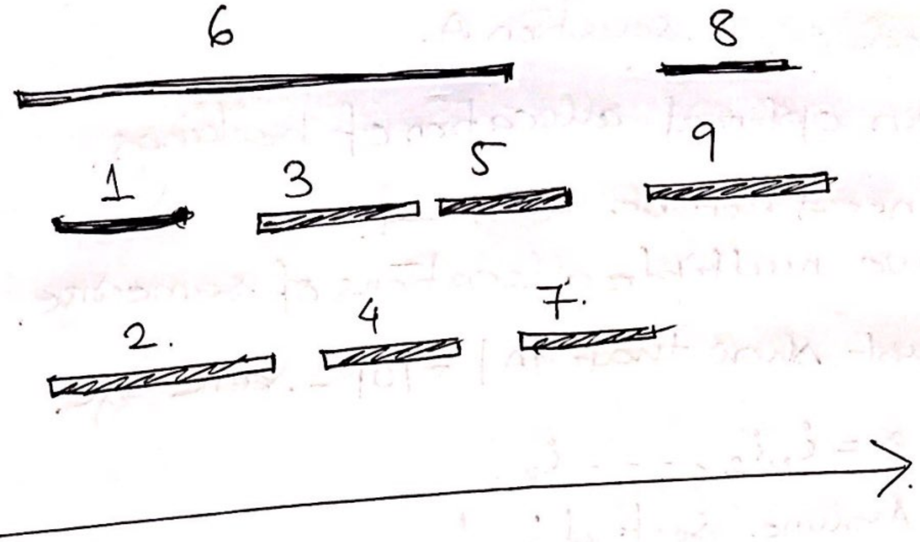
→ choose a subset of booking to maximize the number of teachers who get to use the room.

Greedy approach.

→ Pick the next booking to allot based on a local strategy.

→ Remove all bookings that overlap with this slot.

→ Argue that this sequence of booking will maximize the number of teachers who get to use the room.



$B = \{1, 2, 3, \ldots 9\}$ $A = \phi$.

smallest finish time, 1 overlapping activities 6 & 2.

$B = \{3, 4, 5, 7, 8, 9\}$ $A = \{1\}$

pick 3, overlapping activities 4.

$B = \{5, 7, 8, 9\}$ $A = \{1, 3\}$.

pick 5, overlapping activities 7.

$B = \{8, 9\}$ $A = \{1, 3, 5\}$.

pick 8, overlapping activities 9.

$B = \phi$ $A = \{1, 3, 5, 8\}$.

$B = \phi$ so we stop.

Soln.

Correctness of proposed algorithm.

Our algo produces a solution A.

Let O be an optimal allocation of bookings.

A and O need not be identical.

Can have multiple allocations of same size.

Instead, just show that $|A| = |O|$ - same size.

Let $A = i_1, i_2, \ldots i_k$.

Assume sorted:

$f(i_1) \leq S(i_2), f(i_2) \leq S(i_3) \ldots$

Let $O = j_1, j_2 \ldots j_m$.

Assume sorted:

$f(j_1) \leq S(j_2), f(j_2) \leq S(j_3) \ldots$

To show that $k = m$.

Claim For each $r \leq k$ $f(i_r) \leq f(j_r)$

Our greedy solution "stays ahead" of O.

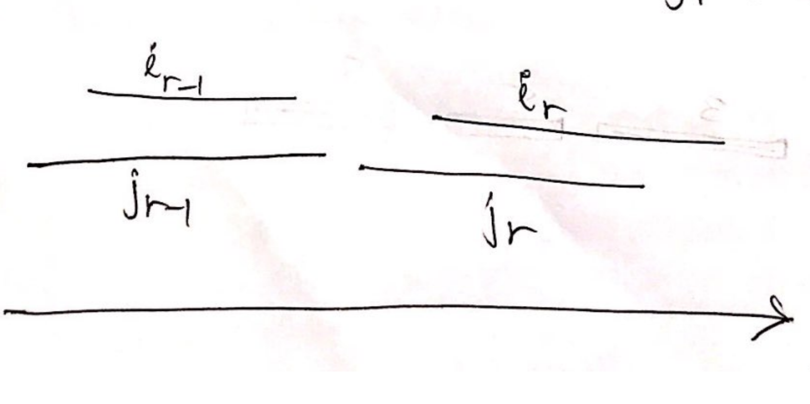
Proof: By induction on r.

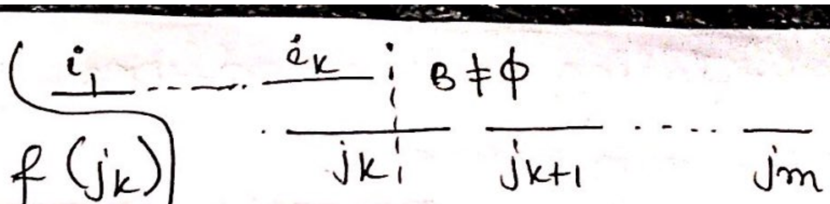
$r = 1$, our algorithm chooses booking i_1 with earliest overall finish time.

$r > 1$: Assume, by induction that $f(i_{r-1}) \leq f(j_{r-1})$

Then, it must be the case that $f(i_r) \leq f(j_r)$

If not, algorithm would choose j_r rather than i_r.





→ Suppose $m > k$.

→ We know that $f(i_k) \leq f(j_k)$

→ consider booking j_{k+1} is 0

→ Greedy algorithm terminates when B is empty.

→ Since $f(i_k) \leq f(j_k) \leq S(j_{k+1})$, this booking is compatible with $A = i_1, i_2 \ldots i_k$.

→ After selecting i_k, B still contains j_{k+1}. Contradiction

So it is correct.

Complexity.

Initially sort the n bookings by finish time, $O(n \log n)$

→ Booking are renumbered $1, 2, \ldots n$ in this order.

Setup an array $ST[1 \ldots n]$ so that $ST[i] = S(i)$

Start with booking 1.

After choosing booking j, scan $ST[j+1], ST[j+2], \ldots$, and choose first k such that $ST[k] > f(j)$

Second phase is $O(n)$, so $O(n \log n)$ is overall complexity.

Scheduling with deadlines.

Minimizing lateness.

→ A single resource, n requests to use this resource,

→ Request i ~~request~~ requires time $t(i)$ to complete and has a deadline $d(i)$

→ All requests will be scheduled.

Request j starts at $s(j)$ and ends at $f(j) = s(j) + t(j)$

· If $s(j) + t(j) > d(j)$ request j is late by

$l(j) = d(j) - f(j)$

Goal: Minimize maximum lateness.

→ Minimize the maximum value of $l(j)$ over all j.

Strategy 1.

choose jobs in increasing order of length $-t(j)$.

eg. $t(1) = 1$, $d(1) = 100$..

$t(2) = 10$, $d(2) = 10$.

1, 2 — 11 lateness 1 ⇒ $l(j) = d(j) - f(j)$.

1 — 10 — 11 lateness 0.

Picking shortest job has not given us correct optimal answer.

Strategy 2.

choose job with smaller slack times, $d(j) - t(j)$ first.

Task that can be delayed.

eg. 1- $t(1) = 1$ $d(1) = 2$. $d(j) - t(j) = 2 - 1 = 1$
$t(2) = 10$ $d(2) = 10$. Slack. $10 - 10 = 0$

pick $t(2), t(1)$. as slack is 0, 1 respectively,

latiness = $11 - 2 = 9$.

~~A~~ pick $t(1), t(2)$

latness = $11 - 10 = 1$.

This strategy also does not work.

strategy 3.

Choose job with earliest deadline $d(j)$ first

This strategy is correct,

How do we prove it?

Correctness: all jobs are sorted by deadline.

Assume all jobs are sorted by deadline.

- Renumber so that $d(1) \leq d(2) \leq \cdots \leq d(n)$.

schedule is simple: $1, 2 \ldots n$.

- Job 1 starts at $S(1) = 0$ and ends at $f(1) = t(1)$.
- Job 2 starts at $S(2) = f(1)$ and ends at $f(2) = S(2) + t(2)$

⋮

our schedule has no gap - idle time.

The resource is continuously in use from $S(1)$ to $f(n)$

claim

There is an optimum schedule with no idle time,

shifting jobs earlier to remove idle time can only reduce lateness.

Suppose O is some other optimal schedule.

Transform O step by step until it becomes identical to the schedule A found by the greedy algorithm.

→ A schedule O has an <u>inversion</u> if i appears before j in O but $d(j) < d(i)$

→ By constructing the greedy solution that has no inversion.

Any two Schedules with no inversions and no idle time produce the same lateness.

No inversions, no idle time means the only difference can be in order of jobs with same deadline.

Any reordering of jobs with the same deadline produces the same lateness
[suffeling of jobs makes it no difference in lateness].

Claim :-

There is an optimal schedule with no inversion and no idle time.

→ Let O be an optimal solution with no idle time.

→ (A) if O has an inversion, then there is a pair of jobs i and j such that j is scheduled immediately after i and $d(j) < d(i)$.

→ Find the first point where deadline decreases.

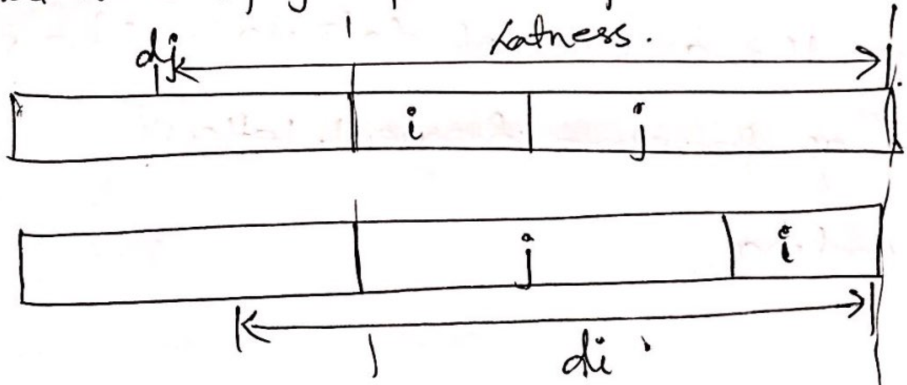
(B) After swapping i and j we get a solution with one less inversion.

obvious.

(C) After swapping i and j we get a solution whose maximum lateness is no larger than that of O.

Recall that $d(j) < d[j]$ i j

lateness of i after swap cannot be more than lateness of j before swap,



latness.

swap does not effect other jobs without increasing latness.

claim:- There is an optimal schedule with no inversions and no idle time.

From (C) we can remove each adjacent inversion without increasing lateness.

At most $n(n-1)/2$ inversions in O to begin with.

Repeatedly removing adjacent inversions to get an optimal. schedule with no inversions, no idle time.

complexity : Sort jobs by deadline – $O(n \log n)$
Reading off schedule – $O(n)$
in same order

$\therefore$ overall complexity $= O(n \log n)$.