This course:
- Methods
- Examples

The Plan:
- First month: algorithmic methods
- Second month: analysis methods
- Third month: advanced topics
Linear-time Median Finding
(Blum, Floyd, Pratt, Rivest, Tarjan, 1973)

Observation #1 Given $n$ numbers $x_1, \ldots, x_n$ and a number $x$, can compute $x$'s "rank"/"position"/"order"
$$rk(x) = \left| \{ 1 \leq i \leq n \mid x_i \leq x \} \right|$$
in linear time.
(Can compare $x$ to each of $x_1, \ldots, x_n$)

Observation #2 If we can find in linear time an element $x$ in $x_1, \ldots, x_n$ such that
$$\frac{3n}{10} \leq rk(x) \leq \frac{7n}{10},$$
we can also find the median in linear time.

Why? If $rk(x) < \frac{n}{2}$ → throw away $x_i$'s s.t. $x_i < x$
If $rk(x) > \frac{n}{2}$ → throw away $x_i$'s s.t. $x_i > x$
If $rk(x) = \frac{n}{2}$, $x$ is the median.

Running-time
$$\leq C \cdot n + C \cdot \frac{3n}{10} + C \cdot \frac{7n}{10}^2 + \ldots = \Theta(n).$$
Throw away at least $\frac{1}{3}$ fraction of elements
The BIG IDEA:

Here is how to find an element with
\[ \frac{3n}{10} \leq rk \leq \frac{7n}{10} \cdot \]

Take the median of medians of constant-size (say, 5) sets of elements:

* takes only linear time:

need to find median on \( \frac{n}{5} \) elements.

Running time:

\[ T(n) = T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + \Theta(n) \]

\[ = cn + C \cdot \frac{9n}{10} + C \cdot \left(\frac{9}{10}\right)^2 n \cdot \ldots = \Theta(n) \]

* The median of medians is at least as large as \( \frac{n}{5} \cdot \frac{1}{2} \cdot 3 \) elements at least as small as \( \geq \frac{3n}{10} \) elements.
Time Management

So many things to do - so little time...

9am lecture 9pm lecture 9th sports
9am lunch 10am workgroup 10pm game
10am meeting 11am class
11am seminar 12pm party
1pm my day

Is there an efficient algorithm that finds a schedule?

Attempt at Formulation #1

Input (Si, fi), ..., (S_n, f_n), where i'th request for i=1, ..., n has start time Si; finish time fi; Si < fi;

Output A schedule (S_{i_1}, f_{i_1}), ..., (S_{i_k}, f_{i_k}) such that

Si_1 < f_{i_1} <= Si_2 < f_{i_2} <= ... <= Si_k < f_{i_k}

where k is maximized.

("I want to do as many things as possible")
Strategies that people try in their everyday life:

1. Pick an activity that takes the least time. (min \( f_i - s_i \))
   Caveat can preclude two other activities.

2. Pick an activity that is most urgent. (min \( s_i \))
   Caveat Less urgent activities can be more beneficial.

These are examples of greedy strategies.
Perhaps surprisingly, a different greedy strategy yields an optimal solution.

Algorithm

While there are still activities to consider,
- Pick \((s_i, f_i)\) with smallest \(f_i\)
- Remove all activities that intersect \((s_i, f_i)\).
Claim: Algō outputs \((s_i, f_i) \rightarrow (s_i, f_i)\) s.t.
\[s_i < f_i \leq s_i < \cdots \leq s_i < f_i \]

Pf: \(f_i \leq s_j \forall ij < k\) because after \((s_j, f_j)\) was picked, activities with \(s_i \leq f_j\) were removed.

Claim: If optimal schedule has \(k^*\) activities,
then greedy outputs a schedule with \(k^*\) activities.

Pf: By induction on \(k^*\).

\(k^*=1\) is obvious.

Assume for \(k^*\), and prove for \(k^*+1\): let the optimal schedule be \((s_{k^*}, f_{k^*})\)...

\(s_{k^*}, f_{k^*}\)

By definition, \(f_i \leq f_{k^*}\)

\(\Rightarrow\) Without loss of generality, \(f_i = f_{k^*}\)

because \((s_i, f_i), (s_{k^*}, f_{k^*})\) is optimal too.

Consider the collection of activities with \(s_i > f_i\).
The optimal schedule for it has \(k^*\) activities
(\(0/w\) \((s_{k^*}, f_{k^*})\) wouldn't have been opt)

By hypothesis, greedy finds a schedule with \(k^*\) activities
\(\Rightarrow\) The total \# of activities of greedy: \(k^*+1\).
Running time of greedy algo: $O(n\log n)$.

- Sort activities according to $f_i$, and scan them one by one.
- Maintain current finish time $f$ and skip activities with $s_i < f$. 

```
1 | 2 | 3 | 4 | 5
```
Second Attempt at Formulation

Input \((s_1, f_1), \ldots, (s_n, f_n)\) as before.

Output

A schedule \((s_1, f_1), \ldots, (s_k, f_k)\)

that maximizes \(\sum_{j=1}^{k} w_{ij}\)

("I have priorities!"")

Observe The greedy algo. fails!

What's better??

1. get less, free fast
2. get more, free later

Depends...

are there exciting opportunities in the near future?
Idea: Have an array with an entry per endpoint.

- Sort the endpoints $s_1, f_1, s_2, f_2$.

- The value at the entry $t$ is the maximal weight of a schedule that starts at point $t$.

- Fill the array by going from the latest endpoint to the earliest endpoint.

1. Value at finish point = value of prev start point.
2. Value at start point $s_i = \max\{\text{value of prev start, } W_i + \text{value at } f_i\}$
3. Update value of prev start to this max.

Running Time: $O(n \log n)$

Algorithmic paradigm: dynamic programming.

Note: We can find the schedule, and not just its weight, by holding per array entry the index of the next endpoint.
Conclusions

+ Formulate the algo. problem you want to solve—different problems have diff. solutions.

+ There are several approaches you may want to try: greedy, dynamic prog., divide-&-conquer, etc. —

more on these during the coming month.