Transition: DS + Algorithms

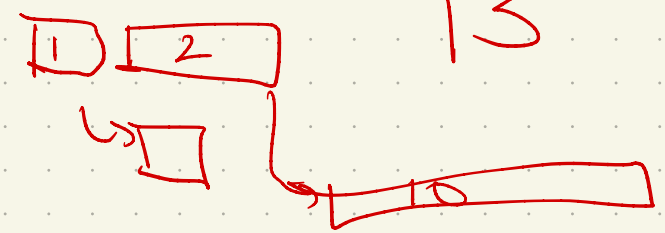
Greed

Worksheet

a) smallest → largest bake

A : 1, 1

B : 2, 10



13 ✓

B, then A : 12

b) largest → smallest cool

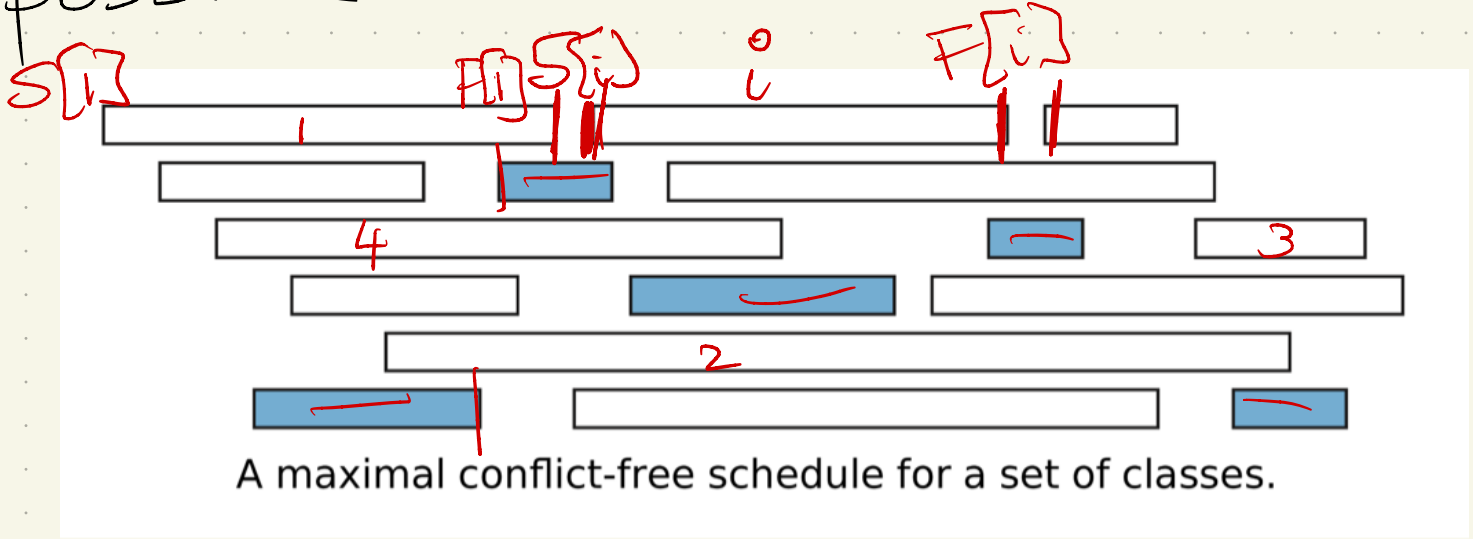
Swap { Spps A shorter cool, & before B w/ longer B finishes earlier & A finishes before B used to

c) $b_i + c_i$, small to large

same example!

Greedy scheduling

Given a set of events (ie intervals, with a start and end time), select as many as possible so that no 2 overlap.



A maximal conflict-free schedule for a set of classes.

More formally:

Two arrays:

S[1..n]

F[1..n]:

Goal:

Subset

→ $X \subseteq [1..n]$

st no 2 overlap $F[i] < S[i+1]$

How would we formalize a dynamic programing approach?

Recursive structure:

Consider job i:

either include or don't

$1 + DP[i+1]$ $\quad$ $DP[i+1]$

$\rightarrow O(n^2)$

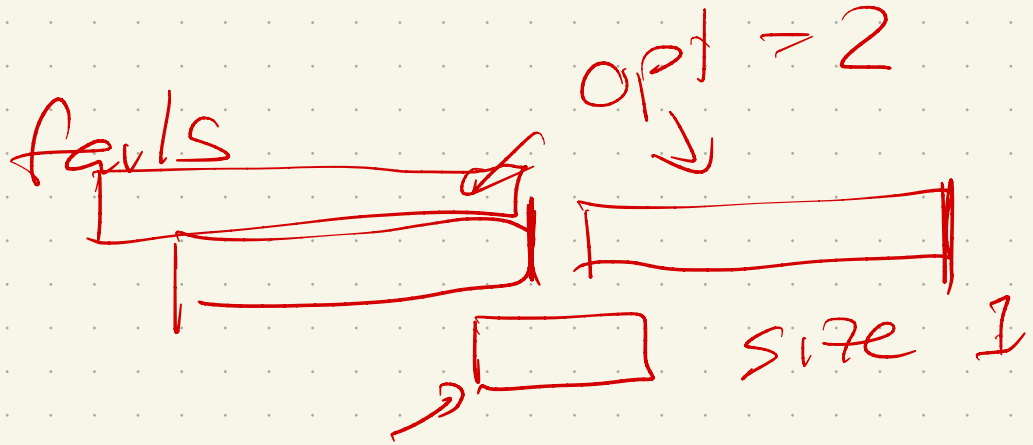
let $DP[i]$ = best schedule among $i+1$ to n jobs

Intuition for greedy:

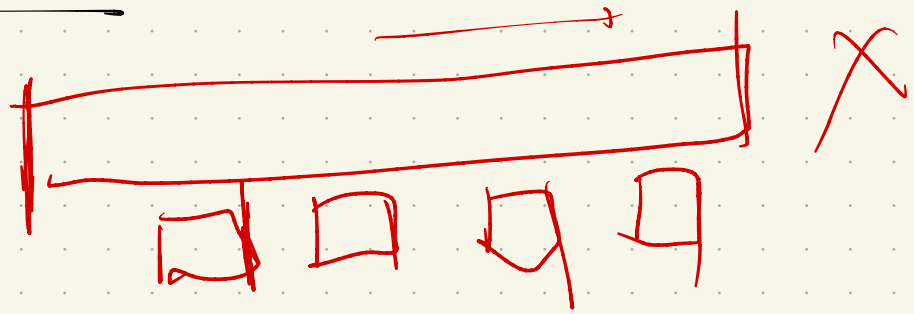
Consider what might be a good first one to choose.

Ideas?

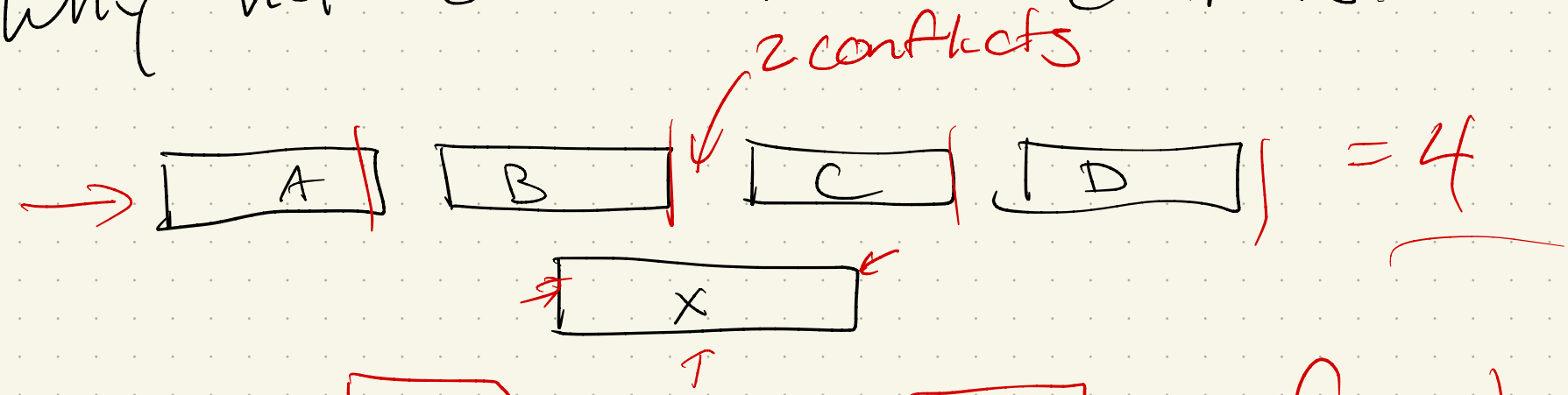
Shortest:

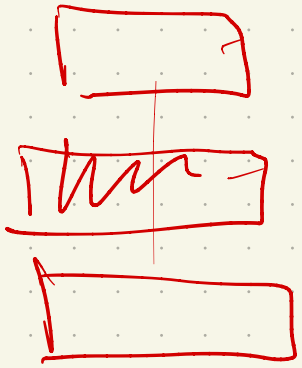


Earliest start:



Why not choose fewest conflicts?

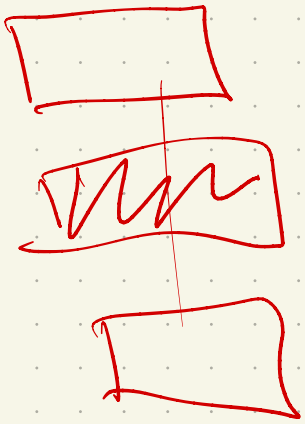




A: 3 conflicts

B: 4

X: 2



C: 4

D: 3

fewest conflicts gets 3

Add jobs that conflict with A+B, & with C+D, but not X

↪ s.t. A-B-C-D is best

Key intuition: sort by finish time

If it finishes as early as possible, we can fit more things in!

So - strategy:

sort by finish time
take earlies, remove any conflicts
& keep going

Code:

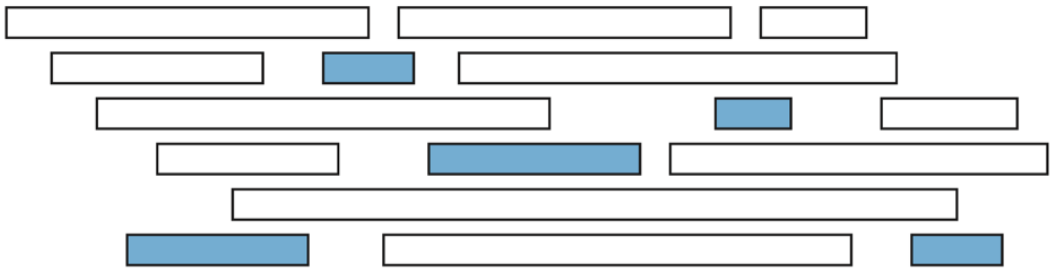
```
GREEDYSCHEDULE(S[1..n], F[1..n]):
  sort F and permute S to match
  count ← 1
  X[count] ← 1
  for i ← 2 to n
      if S[i] > F[X[count]]
          count ← count + 1
          X[count] ← i
  return X[1..count]
```

Runtime?

$O(n \log n)$

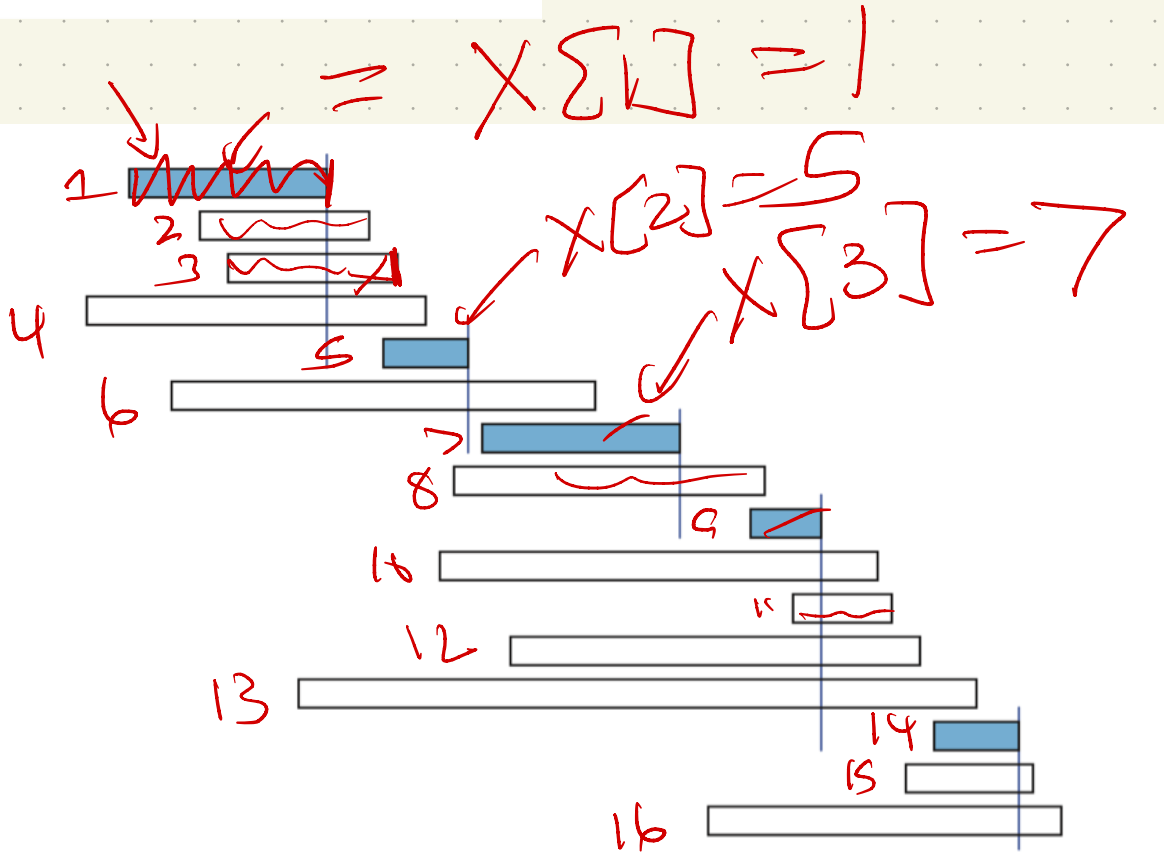
(b/c sort)

Picture:



A maximal conflict-free schedule for a set of classes.

```
GREEDYSCHEDULE(S[1..n], F[1..n]):
  sort F and permute S to match
  count ← 1
  X[count] ← 1
  for i ← 2 to n
      if S[i] > F[X[count]]
          count ← count + 1
          X[count] ← i
  return X[1..count]
```



The same classes sorted by finish times and the greedy schedule.

Correctness:

Why does this work?

Note: No longer trying all possibilities or relying on optimal substructure!

So we need to be very careful on our proofs.

(Clearly, intuition can be wrong!)

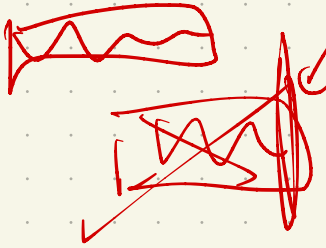
Lemma: We may assume the optimal schedule includes the class that finishes first.

pf: Suppose not: $O_1, \dots O_k$

look at the interval that end first in opt: O_1

Since it's a valid schedule,

$S[O_i] > F[O_1]$



But then replace O_1 with class finish first.

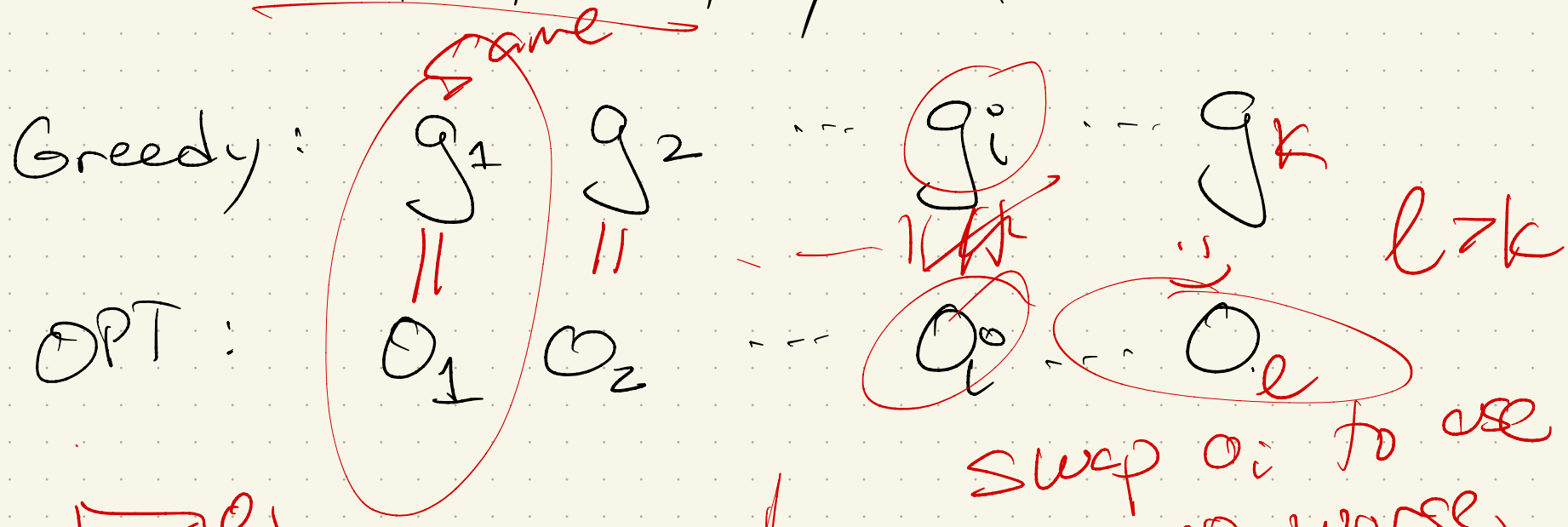
$F[1] < F[O_1] < S[O_i]$ still stuck!

Thm: The greedy schedule is an optimal.

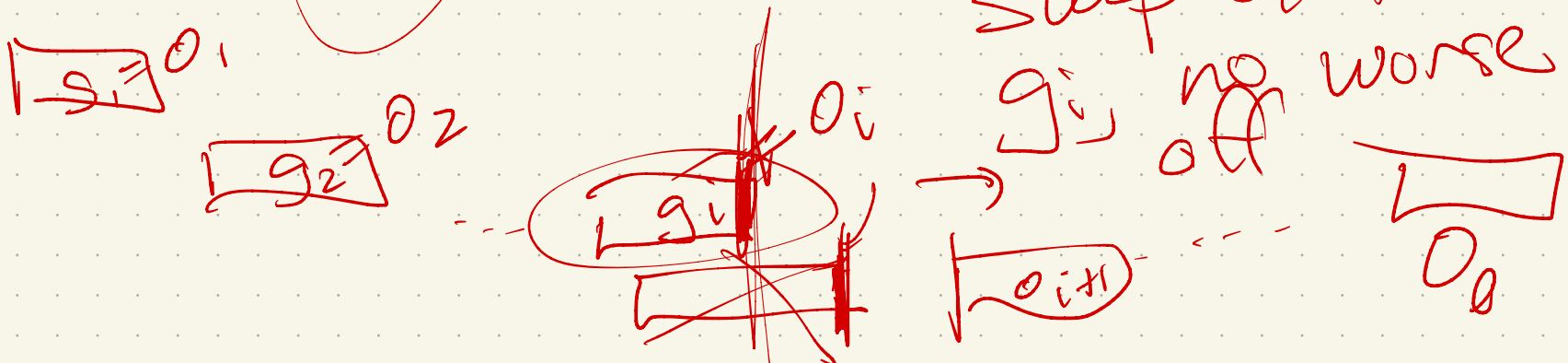
Pf: Suppose not.

Then ∃ an optimal schedule that has more intervals than the greedy one.

Consider first time they differ: call this i



swap o_i to g_i, no worse



Conclusion:

no first place they differ

greedy is optimal!

Takeaway

(often)

- Greedy is faster.
- Won't always work!
  Need to think adversarially,
  + ideally work out an
  "exange" proof:

★ If greedy ≠ opt, find where they differ, + show a local swap doesn't hurt.

Next reading:

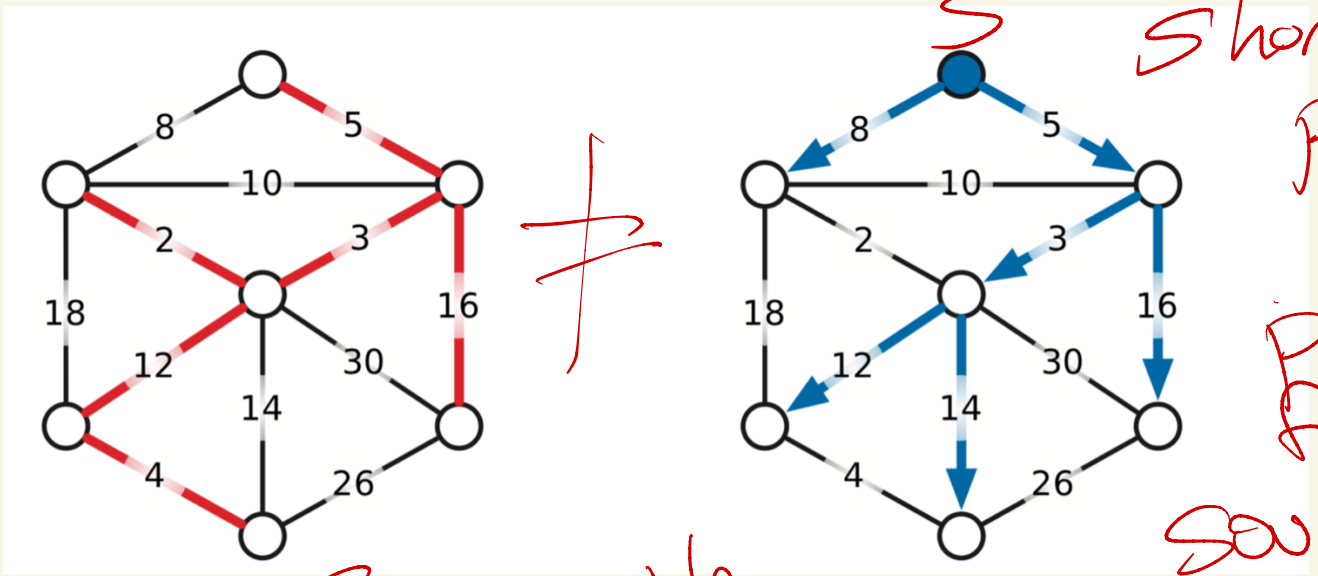
DFS & BFS

linked lists adj lists

- First part is a reminder of graph data structures

- Then: MST and shortest paths

vertices edges w/ weights



MST - tree w/ min possible weights

shortest path tree: paths from some source vertex to all others