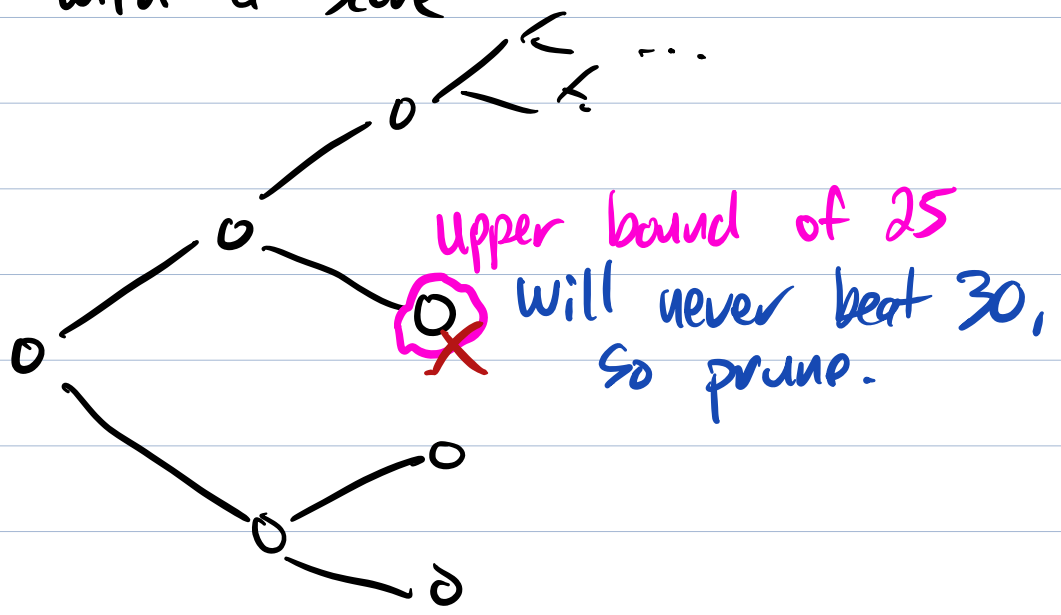
Topic 8 - Branch and Bound (continued)

"I don't know how good I can do when completing this partial solution, but I know for sure that I can't do better than Y."

Have a complete solution with a score of 30. (maximization)



Hard part: how to compute an upper bound
→ we'll come back to that

Ex. Job Assignment Problem.
You have n tasks that need to be done and n workers. Each task

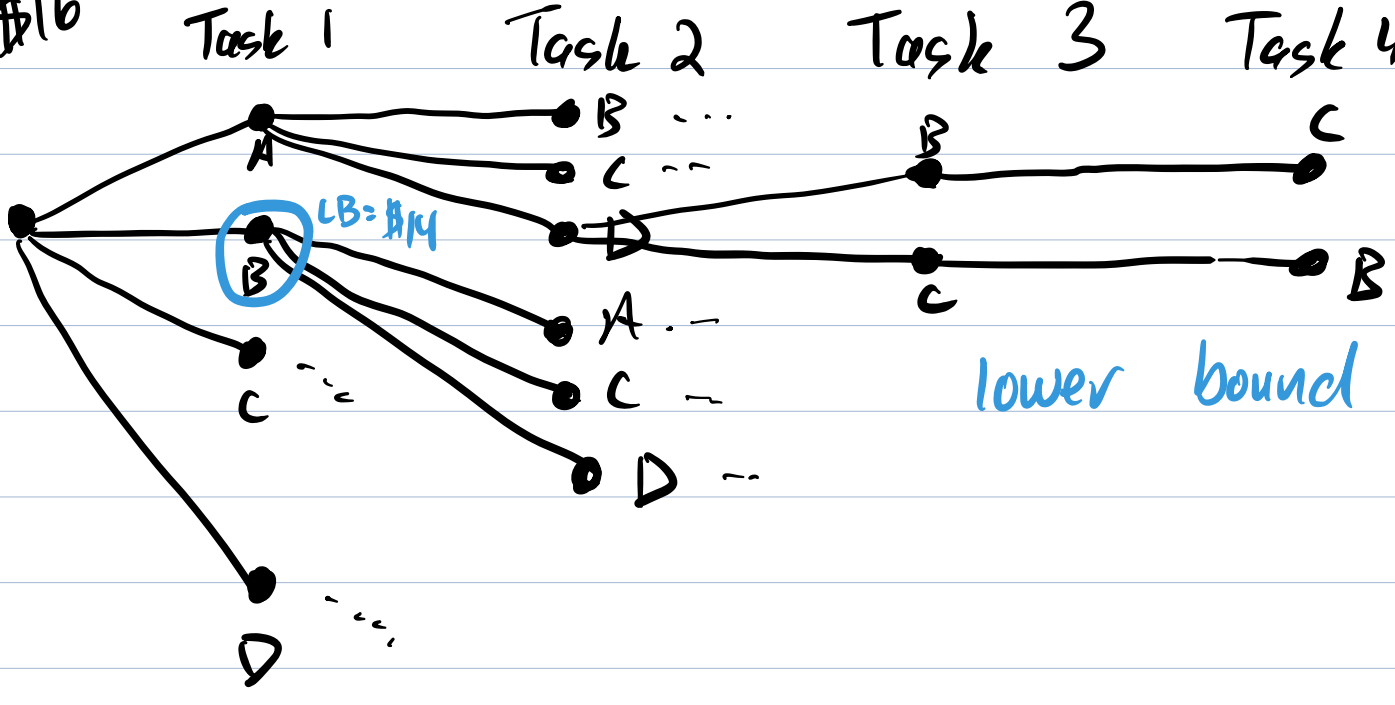
has a different cost to complete depending on which worker does it.
Goal: Minimize total cost.

workers \ tasks	1	2	3	4
A	3	5	2	2
B	6	8	10	8
C	2	6	4	9
D	10	4	7	5

Assign 1 task to each worker.

* Search space: All assignments of workers to tasks. How big?
$n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1$
* No constraints, so backtracking alone is = brute force.

#16



Bounding:

* Suppose you've already picked Worker B to do Task 1.
* What is a lower bound on the best you can do to finish?

	tasks 1	2	3	4
A	~~3~~	5	2	2
B	6	~~8~~	~~10~~	~~8~~
C	~~2~~	6	4	9
D	~~10~~	4	7	5

- I don't know how cheaply I can finish, but I know <u>for sure</u> I can't finish cheaper than X.
  * Has to be much faster than brute force / just solving the problem.

One possibility:
every remaining task is going to cost us <u>at least</u> its minimum remaining cost.

	tasks 1	2	3	4
A	~~3~~	5	2	2
B	6	~~8~~	~~10~~	~~8~~
C	~~2~~	6	4	9
D	~~10~~	4	7	5

Task 2: Will cost at least $4
Task 3: $2
Task 4: $2

The whole assignment: ≥ $14 $8

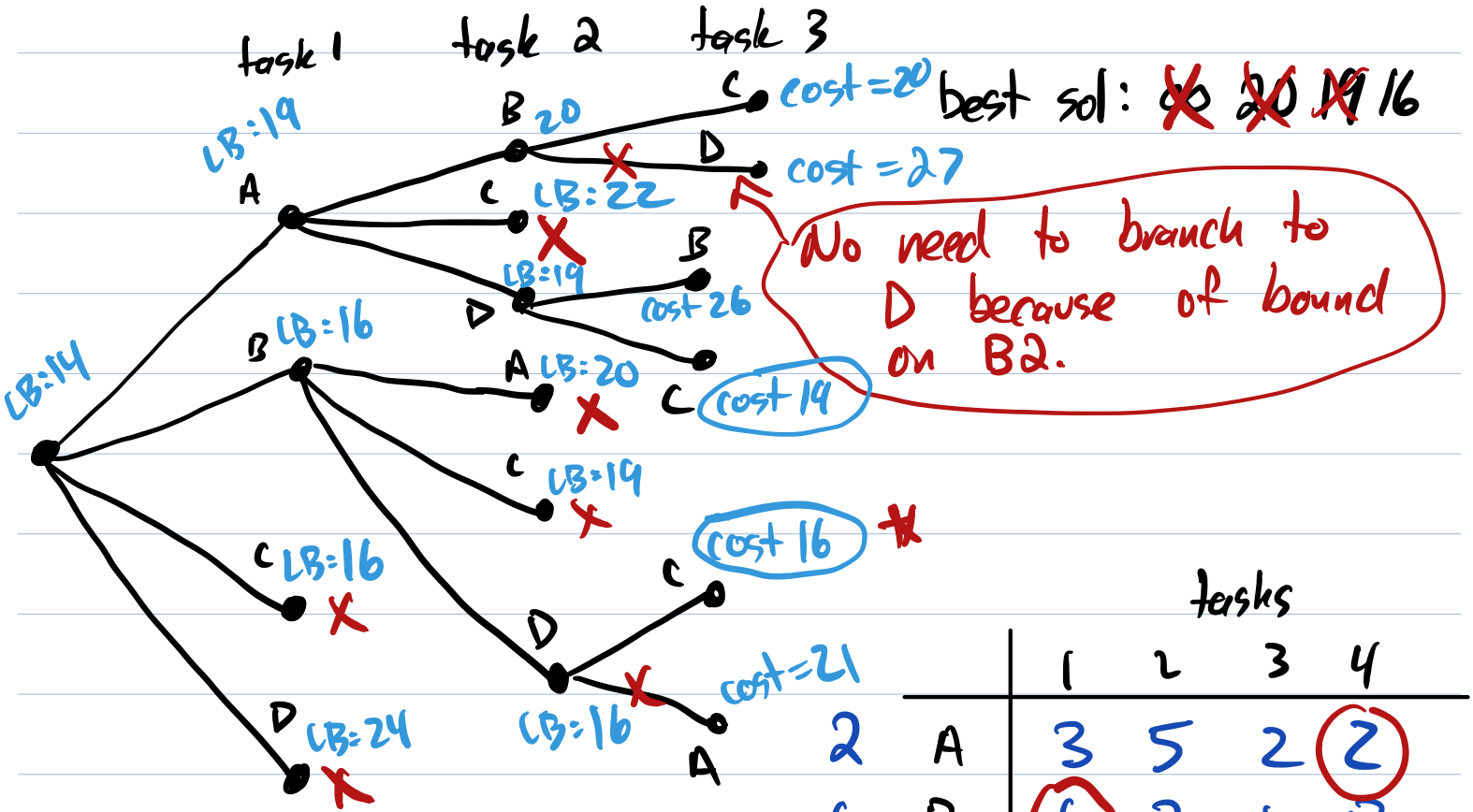
Another possibility:
every worker will incur a cost at least the minimum of the tasks remaining.

	tasks 1	2	3	4	
A	~~3~~	5	2	2	$2
B	6	~~8~~	~~10~~	~~8~~	
C	~~2~~	6	4	9	$4
D	~~10~~	4	7	5	$4
					$10

Lower bound: $6 + 10 = 16$

stronger lower bound for this particular example.

LB: Compute both (the sum of the min of each remaining colum) and (the sum of the min of each remaining row) and take the bigger one.
(+ the cost of whatever selections you've already made)



optimal:

		tasks 1	2	3	4
2	A	3	5	2	(2)
6	B	(6)	8	10	8
4	C	2	6	(4)	9
4	D	10	(4)	7	5
16					

(16)

Notes:

* Hardest part is finding a good bound. The stronger the better, as long as it's not too slow.
* At the start we had no best solution, and couldn't prune until we had a full solution.
  - Pick a few random solutions

or

- Pick a greedy solution to start with.

General Procedure:

```
function bb(S, best_sol = None)
    if best_sol is None:
        best_score = -∞
    else:
        best_score = score(best_sol)

    if |S| = 1:
        candidate = the one thing in S
        value = score(candidate)
        if value > best_score:
            return candidate
        else:
            return best_sol
```

the search space or some subspace (maximizing)

↳ returns best sol in the subspace S

(the subspace only has a single sol, so we check it!)

base case