# Scientific Computing

Mon, Feb 23

## Announcements

* HW 2 scores and feedback are posted on D2L.
  Overall very good!

* HW 3 is due Friday!
  covers brute force, search spaces,
  divide + conquer

* Midterm exam
  Monday, March 2
  in class portion + takehome portion
  due Friday, March 6

* I will post a "HW 4 preview" today to help you study BT and B+B

**Office Hours:**

Mon, 9:30-10:30
Fri, 2:00-3:00

Cudahy 307

# 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? (Has to be easier than actually solving the whole problem.)

| workers \ tasks | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 8 | 5 | 2 | 2 |
| B | 6 | 8 | 10 | 8 |
| C | 2 | 6 | 4 | 9 |
| D | 16 | 4 | 7 | 5 |

minimizing = lower bound to do B+B

(8, ∞)

(10, ∞)

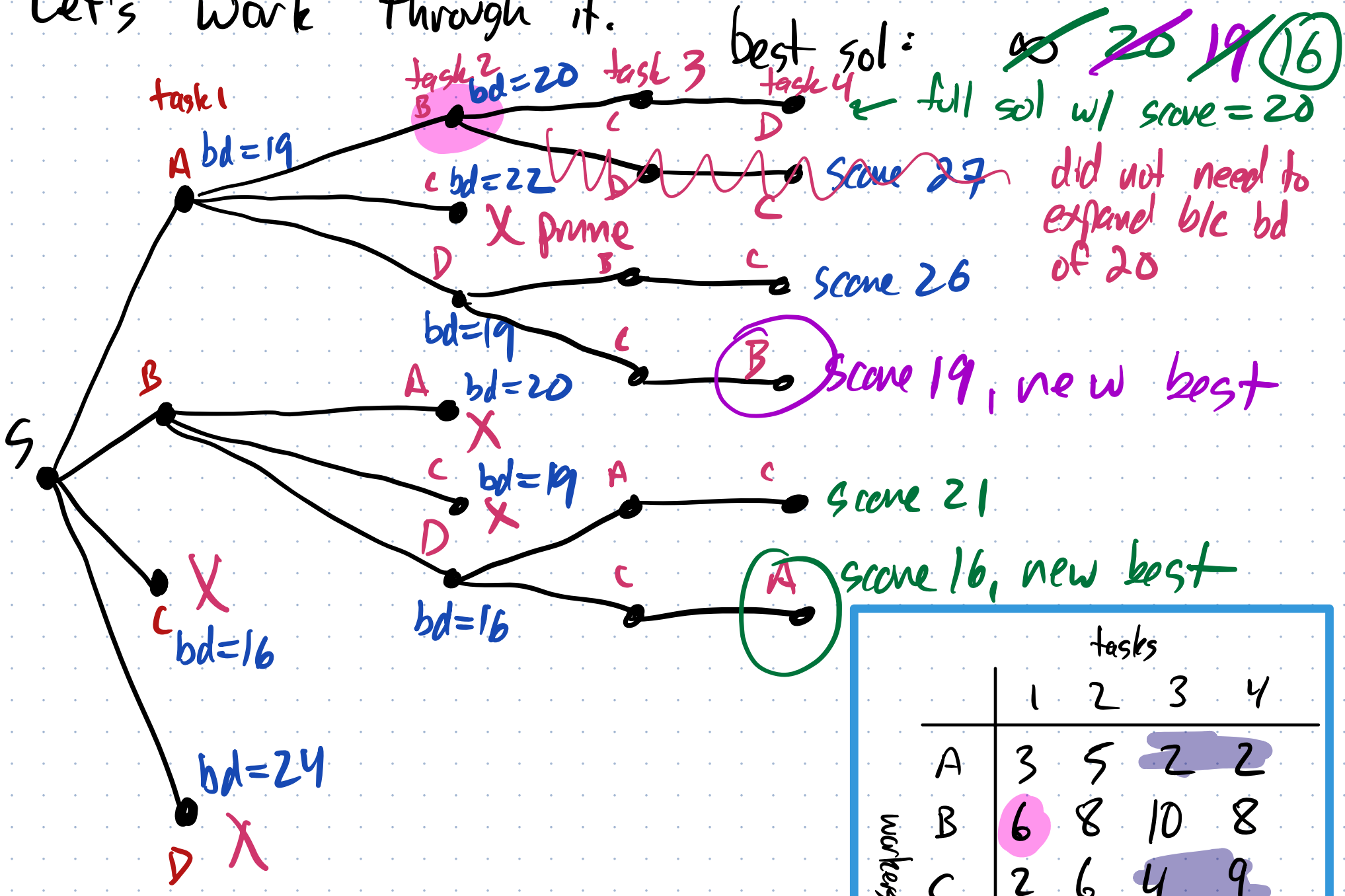
We don't know how cheaply we can finish the other jobs, but it can't be cheaper than X.

LB: $1 per remaining job ($3)

LB: $2 per job ($6)

LB: $8 => LB: $10

Let's work through it.




| 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 |

## General Procedure:

search space

```
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
```

Maximizing

setting best_score

(only has a single complete candidate)

base case

## General Procedure:

```
function bb( , 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
```

else assuming branch ↓ into 2

```
S1, S2 = branch(S)
if bound(S1) > best_score:
    best_sol = bb(S1, best_sol)
    best_score = score(best_sol)
if bound(S2) > best_score
    best_sol = bb(S2, best_sol)
    best_score = score(best_sol)
return best_sol
```

# Relaxation

Let's try to figure this out with the knapsack problem.

Capacity: 14

| item | weight | value |
|---|---|---|
| 1 | 8 | 13 |
| 2 | 3 | 7 |
| 3 | 5 | 10 |
| 4 | 5 | 10 |
| 5 | 2 | 1 |
| 6 | 2 | 1 |
| 7 | 2 | 1 |

maximizing

Branching: just like before
- → item 1 in or out
- → item 2 in or out
- ⋮

Bounding: Suppose we have put item 1 out and item 2 in. How can we find an upper bound on the best we could possibly do with the rest?

Remember:

* Greedy sol gives the wrong bound.
* "Add up the value of all remaining items" is technically an upper bound, but a useless one.
* Computing the UB can't be too slow.

Ideas?

The trick is called relaxation: sometimes it's easier to find an UB if you adjust the problem to be more permissible.

Fractional Knapsack: You are allowed to take fractions of items

Capacity: 14

| item | weight | value | | |
|---|---|---|---|---|
| 1 | 8 | 13 | 1.0 | capacity left of 6 |
| 2 | 3 | $7_{3.5}$ | 0.5 | capacity left of 4.5 |
| 3 | 5 | $10_{2}$ | 0.2 | capacity left of 3.5 |
| 4 | 5 | 10 | | |
| 5 | 2 | $1_{1}$ | 1.0 | |
| 6 | 2 | $1_{0.75}$ | 0.75 | |
| 7 | 2 | 1 | | |

20.25

A solution

**Theorem**: A greedy and optimal solution can be found by: (1) ordering the items by value density (value/weight)

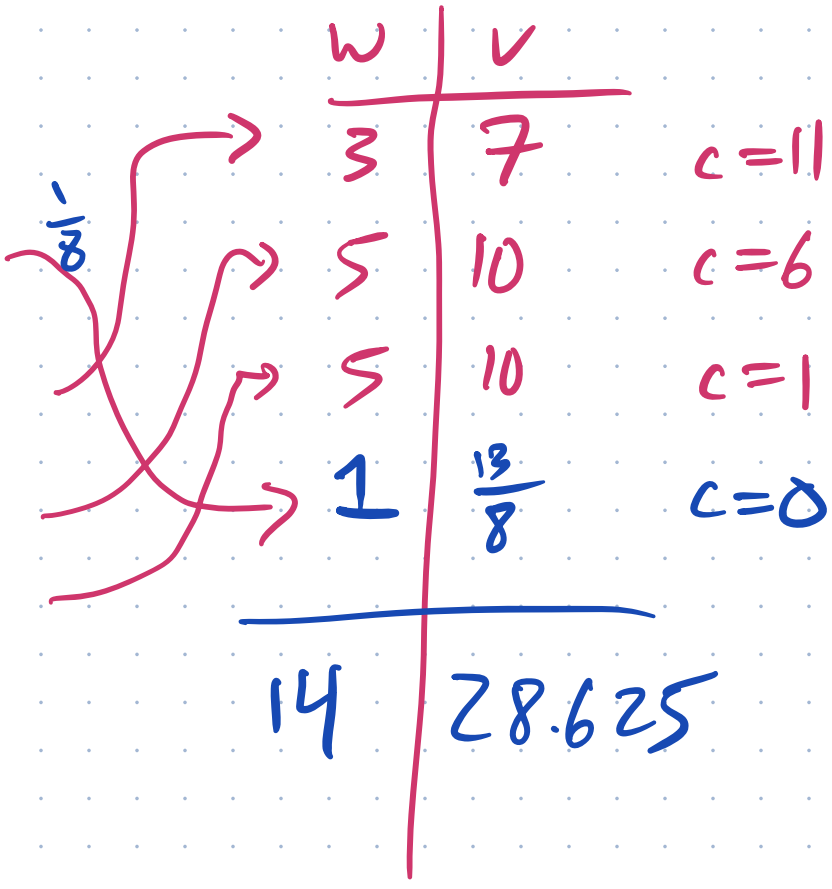
(2) taking items from the top in full until you can't anymore

(3) taking whatever fraction you can of the next item

We won't prove this, but you should think about it until you believe it.

Capacity: 14

| item | weight | value | density | |
|---|---|---|---|---|
| 1 | 8 | 13 | 1.625 | (4) |
| 2 | 3 | 7 | 2.333 | (1) |
| 3 | 5 | 10 | 2 | (2) |
| 4 | 5 | 10 | 2 | (3) |
| 5 | 2 | 1 | 0.5 | (5) |
| 6 | 2 | 1 | 0.5 | (6) |
| 7 | 2 | 1 | 0.5 | (7) |




(If capacity = 10, we get value 21, a better solution than the regular knapsack.)

So, Fractional Greedy = Fractional Optimal
≥ Regular Optimal.

more permissive

Thus, we can get an UB for B+B by computing the greedy fractional solution on remaining items.

# Upper Bound for item 1 out, item 2 in:

Capacity: 10 * Back to 10 to be easier

| item | weight | value | density | |
|---|---|---|---|---|
| ~~1~~ | ~~8~~ | ~~13~~ | ~~1.625~~ | ~~1~~ |
| 2 | 3 | 7 | 2.333 | 1 ✓ |
| 3 | 5 | 10 | 2 | 2 |
| 4 | 5 | 10 (4) | 2 | 3 |
| 5 | 2 | 1 | 0.5 | 5 |
| 6 | 2 | 1 | 0.5 | 6 |
| 7 | 2 | 1 | 0.5 | 7 |

UB = 21

- 1.0 c = 2
- 0.4 c = 0

Let's work out the B+B tree.

Greedy start!

Capacity = 10

| item | weight | value | density | |
|---|---|---|---|---|
| 1 | 8 | 13 | 1.625 | (4) |
| 2 | 3 | 7 | 2.333 | (1) |
| 3 | 5 | 10 | 2 | (2) |
| 4 | 5 | 10 | 2 | (3) |
| 5 | 2 | 1 | 0.5 | (5) |
| 6 | 2 | 1 | 0.5 | (6) |
| 7 | 2 | 1 | 0.5 | (7) |

Three greedy solutions:

most value = 14

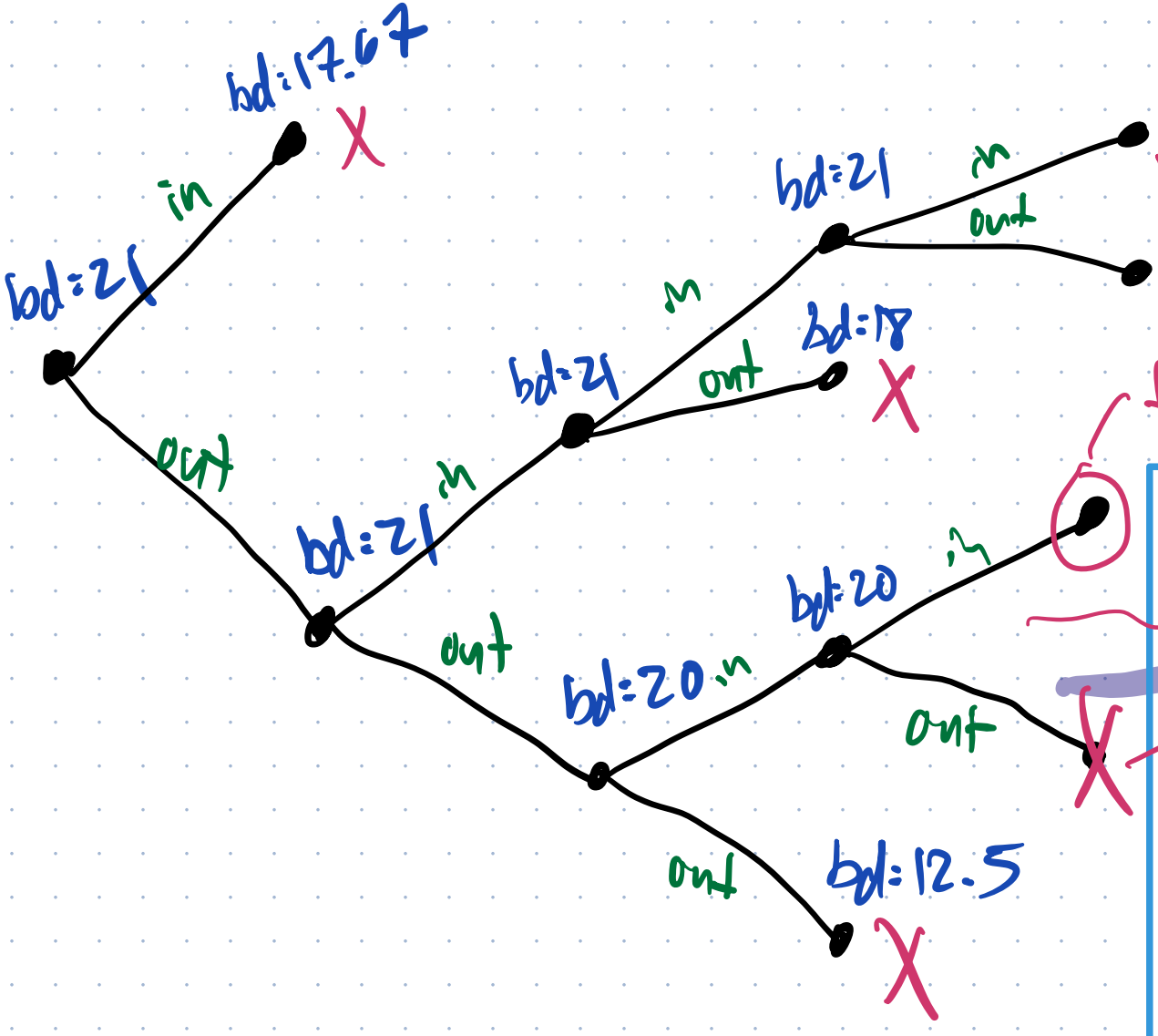
lightest = 10

value dence : 18

# B+B tree

Best sol: ~~18~~ 20

item 1 item 2 item 3 item 4




Capacity: 10

| item | weight | value | density | |
|---|---|---|---|---|
| 1 | 8 | 13 | 1.625 | 4 |
| 2 | 3 | 7 | 2.333 | 1 |
| 3 | 5 | 10 | 2 | 2 |
| 4 | 5 | 10 | 2 | 3 |
| 5 | 2 | 1 | 0.5 | 5 |
| 6 | 2 | 1 | 0.5 | 6 |
| 7 | 2 | 1 | 0.5 | 7 |

# Compare to backtracking alone!