5112

9-14

Dynamic Programming

Weighted Interval Scheduling

Join the Slack!

Section 6.1

Weighted Interval Scheduling

Input: $\{I_j = (s_j, f_j, v_j)\}$ $1 \le j \le n$

Call $S \subseteq \{1, \ldots, n\}$ compatible if $j, k \in S$ I_j doesn't overlap with I_k.

Output: Compatible S which maximizes $\sum_{j \in S} v_j$

Weighted Interval Scheduling

It's natural to order jobs by EFT

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_n$$

Say that job i comes before job j if $i < j$.

Define $p(j)$ to be the largest index i such that the intervals i and j don't overlap.
Say $p(j) = 0$ if no such i exist.

Weighted Interval Scheduling

Let's think about an optimal solution O.
Specifically, is $n \in O$ or not?

Case 1: $n \in O$.
Then we know $p(n)+1, p(n)+2, \ldots, n-1 \notin O$.
Because they overlap with n.
So which subset of $\{1, \ldots, p(n)\}$ is the rest of O?

Both reduce to smaller subproblems

Case 2: $n \notin O$.
Then we know O is an optimal solution on $\{1, \ldots, n-1\}$.

Weighted Interval Scheduling

the value of the

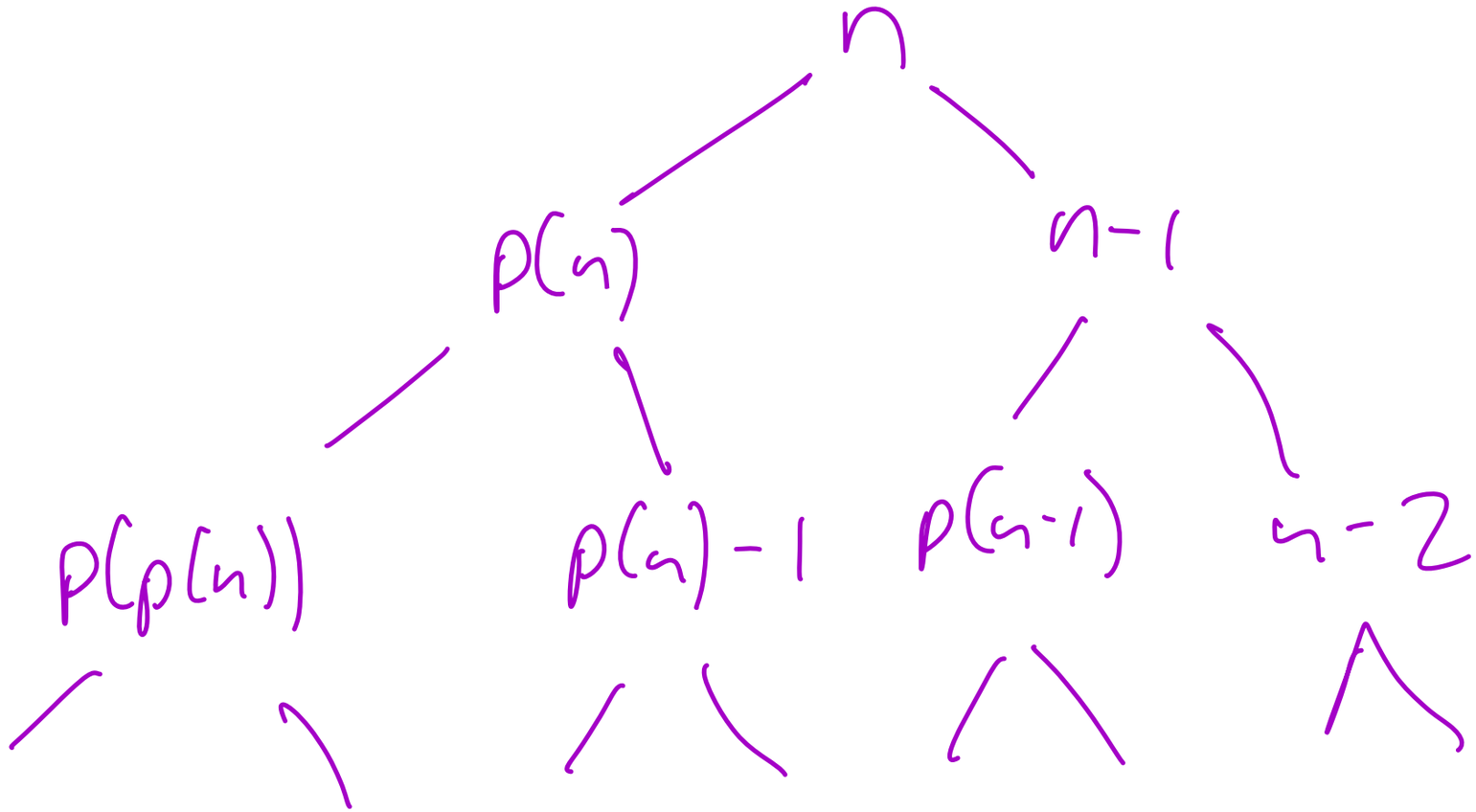
// computes optimal compatible
// subset on $\{1, \ldots, j\}$.

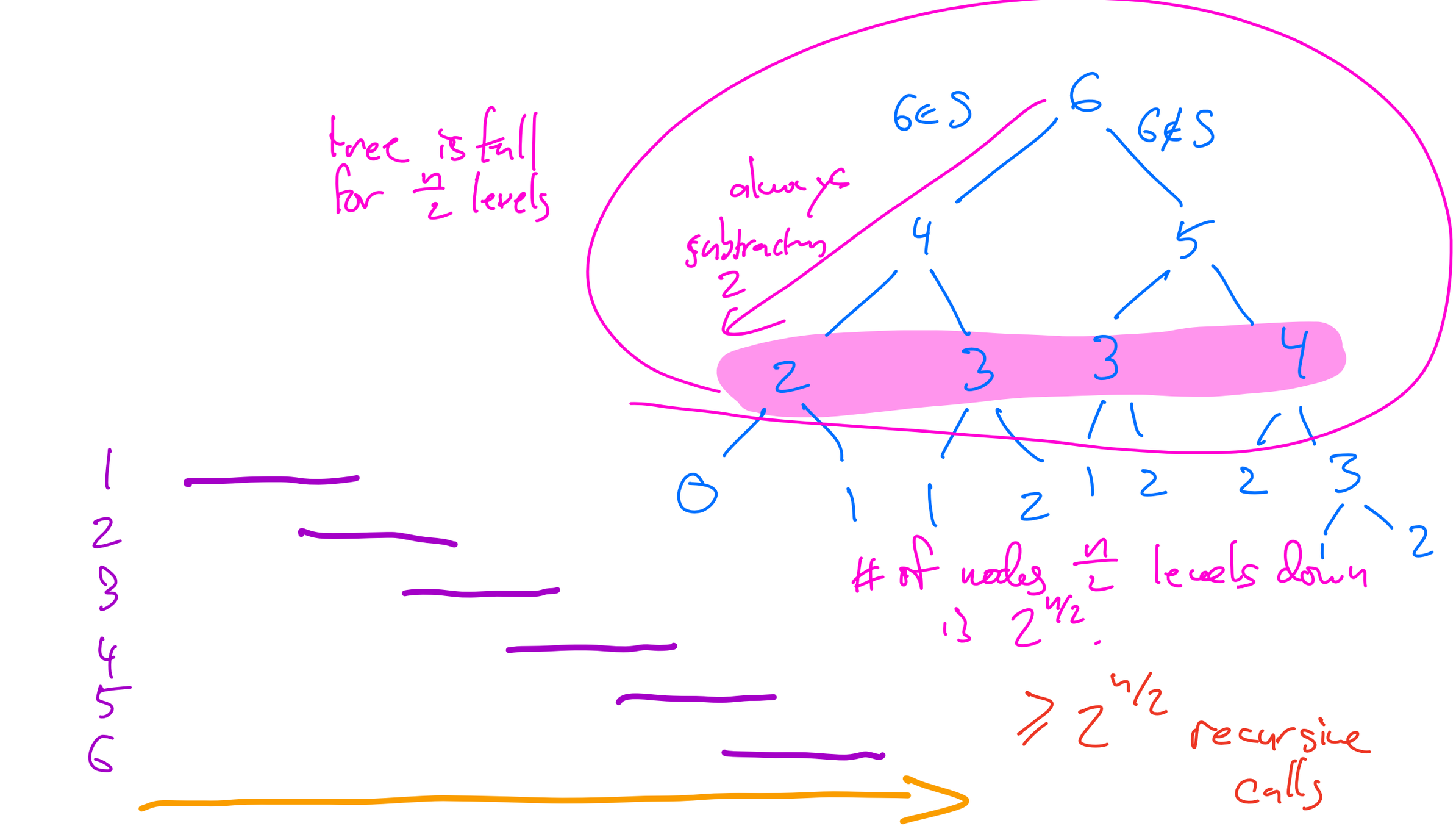
```
Opt-val(j)
  If j = 0
    Return 0
  Else
    Return Max(Opt-val(p(j)) + v_j, Opt-val(j-1))
```

case 1 ↙ Opt-val(p(j)) + v_j

case 2 ↘ Opt-val(j-1)

Running time?


n
p(n)
n-1
p(p(n))
p(n)-1
p(n-1)
n-2


tree is full for $\frac{n}{2}$ levels
6
6∈S
6∉S
always subtracting 2
4
5
2
3
3
4
0
1
1
2
1
2
2
3
1
2
1
2
3
4
5
6
# of nodes $\frac{n}{2}$ levels down is $2^{n/2}$.
$\geq 2^{n/2}$ recursive calls

Memoization

Idea: Remember the results of recursive calls.

Initialize an array $M[1, \ldots, n]$ $M[i] = -1$ $1 \le i \le n$ ↑ empty

```
M-Opt-Val(j)
  If j = 0
    Return 0
  Else if M[j] ≠ -1
    Return M[j]
  Else
    Set M[j] = max(M-Opt-Val(p(j)) + v_j, M-Opt-Val(j-1))
    Return M[j]
```

Same output as Opt-Val

2n calls, O(1) on each call
⇒ O(n) time

once per j

Iteration vs Recursion

Could do the following algorithm:

Set $M[0] = 0$
For $i = 1, i \leq n, i++$
  Set $M[i] = \max(M[p(i)] + v_i, M[i-1])$

Return $M[n]$

This is effectively the same algorithm.

Subset Sum / Knapsack Problem

Input: $\{w_i\}$ $1 \le i \le n$. Threshold T.

Output: $S \subseteq \{1, \ldots, n\}$ such that

$$\sum_{i \in S} w_i \le T.$$

and we want this to be maximal.

Try a greedy algorithm:

$\{\frac{1}{100}, \frac{1}{2}, \frac{1}{2}\}$ $T=1$

$\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}+\frac{1}{100}\}$ $T=1$