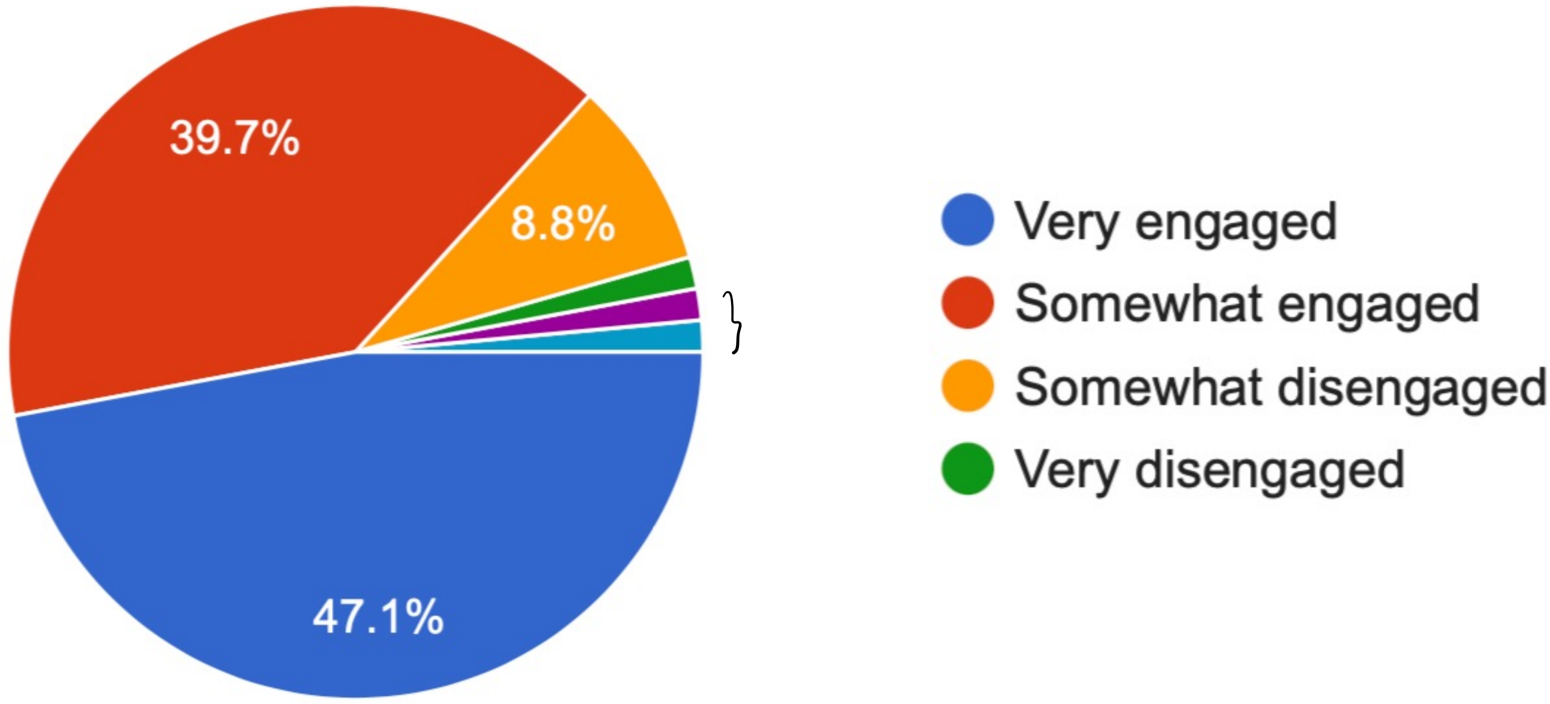
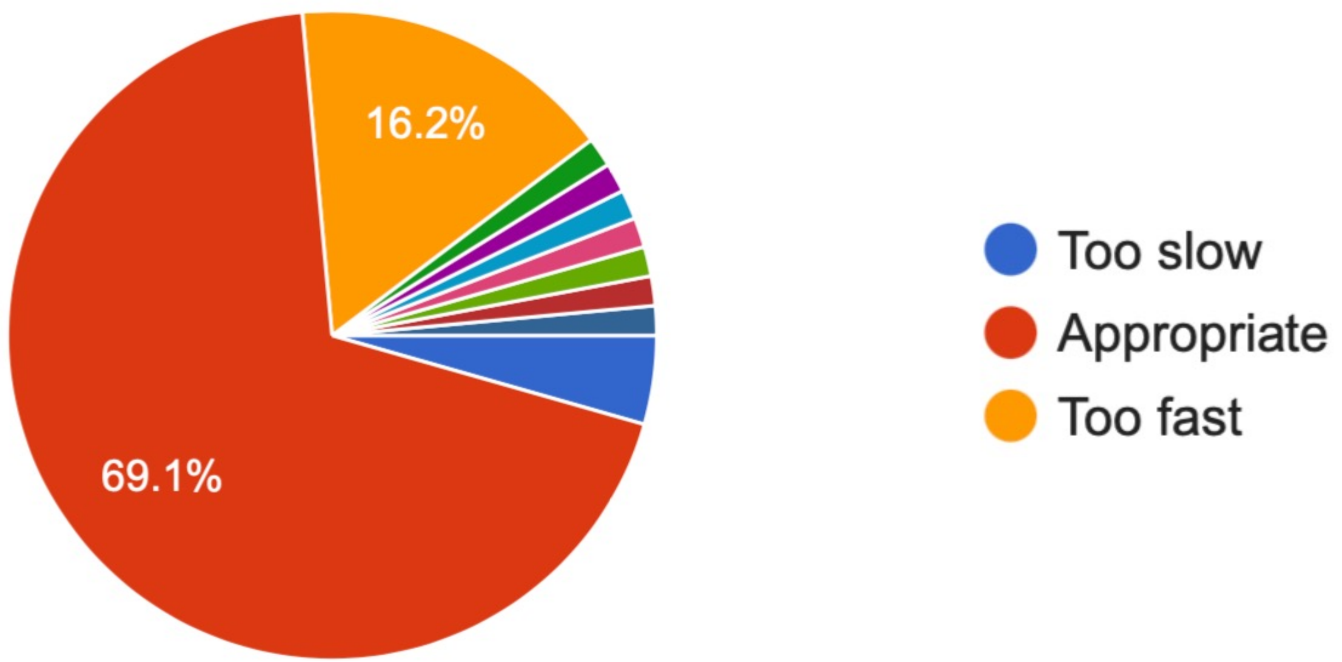
Survey Results

- How engaged do you feel in class?



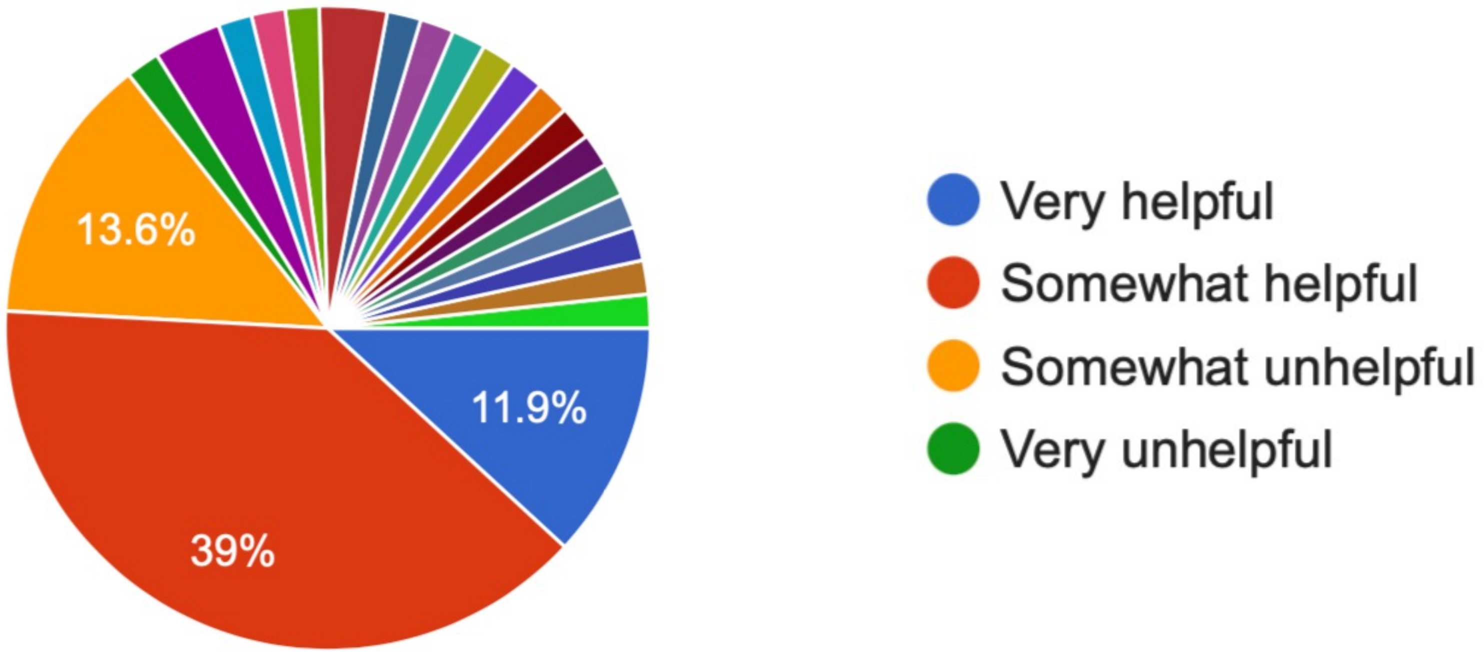
Survey Results

- What do you think about the pace of the lectures?



Survey Results

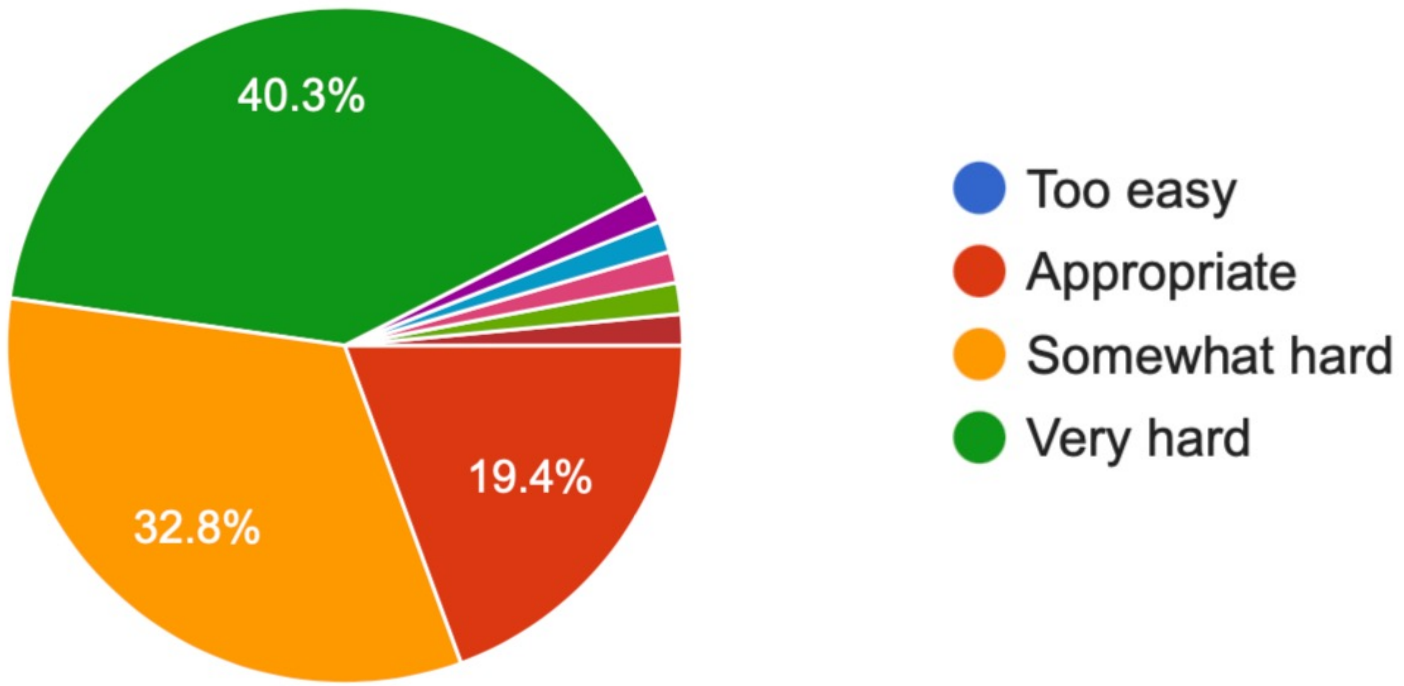
- Do you attend the office hours? If yes, have they been helpful?



GMT-04	SAT 30 8	SUN 1 9	MON 2 10	TUE 3 11	WED 4 12	THU 5 13	FRI 6 14
09:00			Hemasumanth's Office Hours 09:00 – 11:00 https://northeastern.zoom.us/j/4980432053	Hemasth's Off Hours 09:00 - Jamie's Office 09:00 - Mili's Office Hours 09:00 -	Hemasumanth's Office Hours 09:00 – 11:00	Hemasumanth's Office Hours 09:00 – 11:00	Hemasumanth's Office Hours 09:00 – 11:00
10:00							
11:00							
12:00							
13:00					Jamie's Office Hours 13:00 – 15:00 Cahners Hall		
14:00	Pravin's Office Hours 14:00 – 16:00		Soheil's Office Hours 14:00 – 16:00 https://northeastern.zoom.us/j/96544879274				Pravin's Office Hours 14:00 – 16:00
15:00				Ryan's Office Hours 15:00 – 18:00 271 Huntington Ave	Rishabh's Office Hours 15:00 – 17:00 Cahners Hall	Rishabh's Office Hours 15:00 – 17:00	
16:00			Mili's Office Hours 16:00 – 18:00	Amir's Office Hours 16:00 – 18:00			Ryan's Office Hours 16:00 – 17:00
17:00							
18:00							

Survey Results

- What do you think about the difficulty of homework 1?



Dynamic Programming Recipe

- **Recipe:**

  (1) identify a set of **subproblems**

  (2) relate the subproblems via a **recurrence**

  (3) find an **efficient implementation** of the recurrence (top down or bottom up)

  (4) **reconstruct the solution** from the DP table

Dynamic Programming

a. Fibonacci Series

b. Weighted Interval Scheduling

Weighted Interval Scheduling

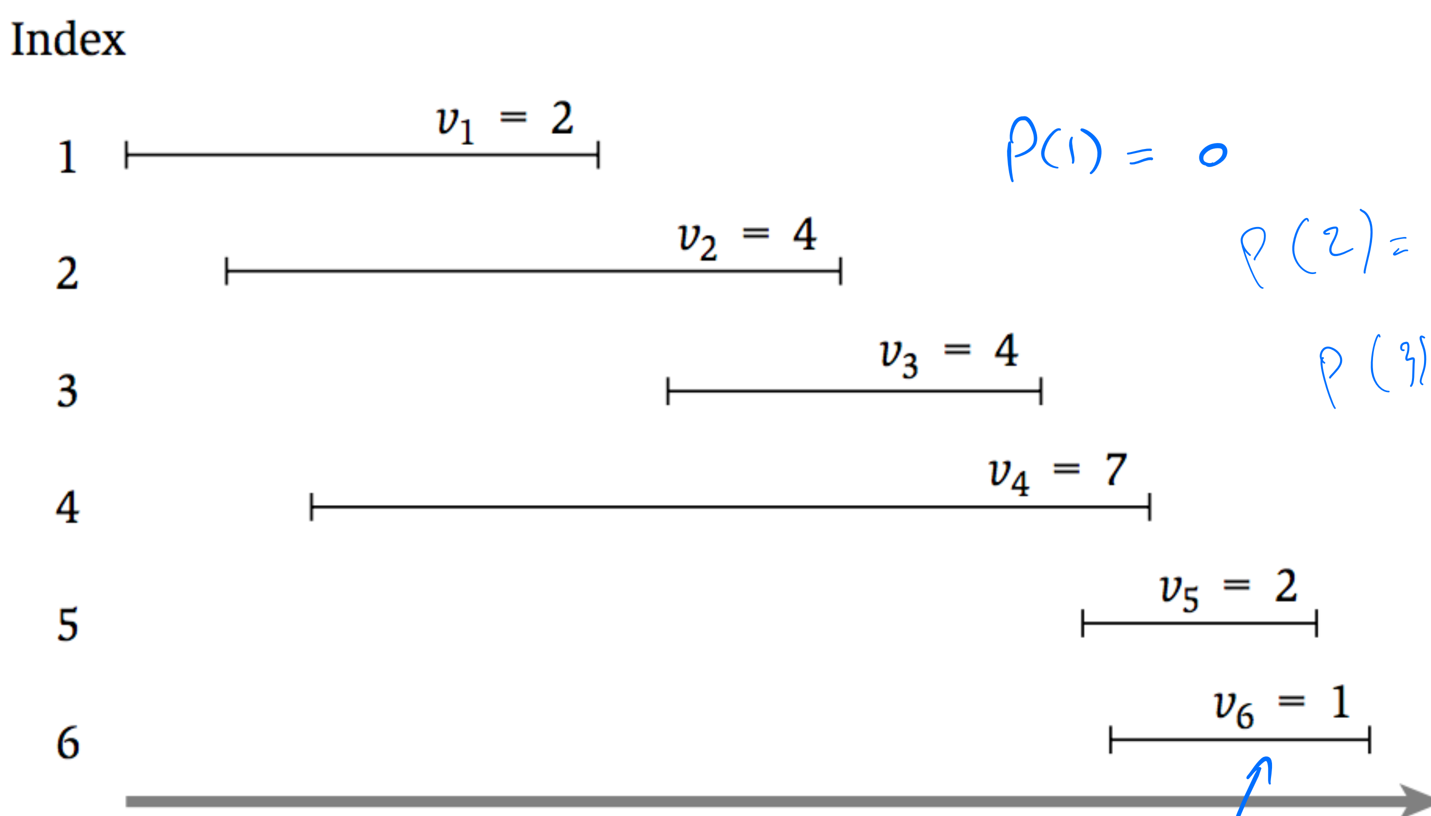
- How can we optimally schedule a resource?
  - This classroom, a computing cluster, …

- **Input:** n intervals (s_i, f_i) each with value v_i
  - Assume intervals are sorted so $f_1 < f_2 < \cdots < f_n$
- **Output:** a compatible schedule S **maximizing** the total value of all intervals
  - A **schedule** is a subset of intervals $S \subseteq \{1, \ldots, n\}$
  - A schedule S is **compatible** if no $i, j \in S$ overlap
  - The **total value** of S is $\sum_{i \in S} v_i$

Interval Scheduling

$2+4+1=7$

$p(i)$: largest j s.t. $f_j \leq s_i$.



$p(1) = 0$

$p(2) = 0$

$p(3) = 1$

$p(4) = 0$

$p(5) = 3$

$p(6) = 3$

$$O_6 = \begin{cases} O_5 \\ v_6 + val(O_{p(6)}) \end{cases}$$

A Recursive Formulation: Subproblems

- ***Subproblems:*** Let O_i be the **optimal schedule** using only the intervals $\{1, \dots, i\}$
- **Case 1:** Final interval is not in O_i ($i \notin O_i$)
  - Then O_i must be the optimal solution for $\{1, \dots, i-1\}$
    - $O_i = O_{i-1}$
- **Case 2:** Final interval is in O_i ($i \in O_i$)
  - Assume intervals are sorted so that $f_1 < f_2 < \cdots < f_n$
  - Let $p(i)$ be the largest j such that $f_j < s_i$
  - Then O_i must be i + the optimal solution for $\{1, \dots, p(i)\}$
    - $O_i = \{i\} + O_{p(i)}$

A Recursive Formulation: Subproblems & Recurrence

- **Subproblems:** Let $OPT(i)$ be the ***value* of the optimal schedule** using only the intervals $\{1, \dots, i\}$ $(OPT(i) = value(O_i))$

- **Case 1:** Final interval is not in O_i $(i \notin O_i)$
  - Then O_i must be the optimal solution for $\{1, \dots, i-1\}$
- **Case 2:** Final interval is in O_i $(i \in O_i)$
  - Assume intervals are sorted so that $f_1 < f_2 < \cdots < f_n$
  - Let $p(i)$ be the largest j such that $f_j < s_i$
  - Then O_i must be i + the optimal solution for $\{1, \dots, p(i)\}$

- $OPT(i) = \max\{OPT(i-1), v_i + OPT(p(i))\}$

- $OPT(0) = 0, OPT(1) = v_1$

Dynamic Programming Recipe

- **Recipe:**

  (1) identify a set of **subproblems**

  (2) relate the subproblems via a **recurrence**

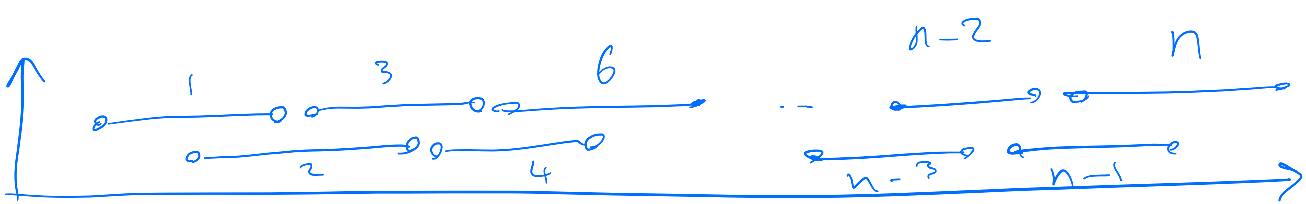
  (3) find an **efficient implementation** of the recurrence (top down or bottom up)

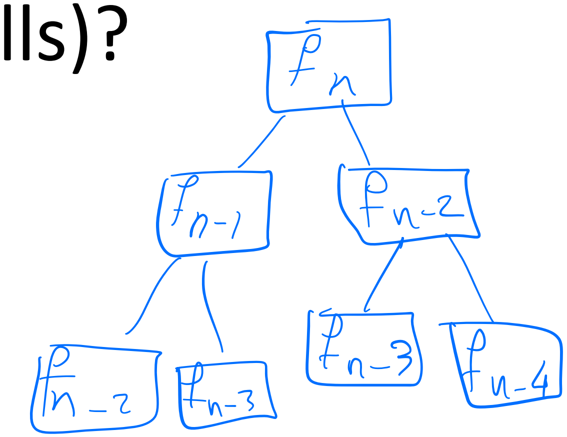
  (4) **reconstruct the solution** from the DP table

Interval Scheduling: Straight Recursion

```
FindOPT(n):
  if (n = 0): return 0
  elseif (n = 1): return v₁
  else:
    return max{FindOPT(n-1), vₙ + FindOPT(p(n))}
```

- What is the worst-case running time of **`FindOPT(n)`** (how many recursive calls)?





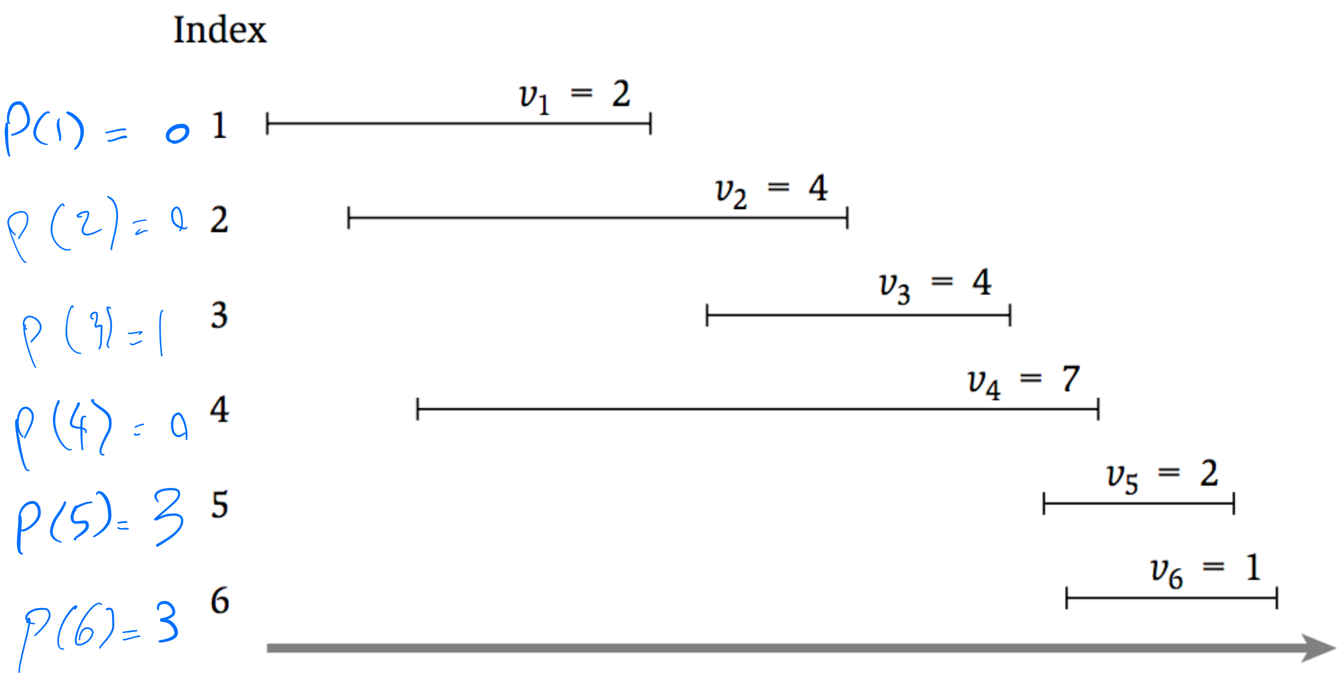
Interval Scheduling: Memoized

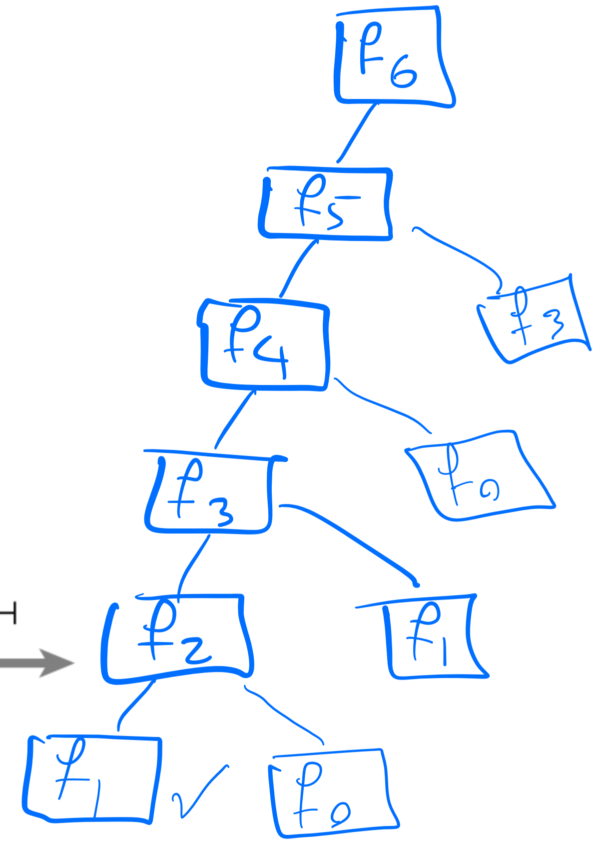
```
// All inputs are global vars
M ← empty array, M[0] ← 0, M[1] ← v₁
FindOPT(n):
  if (M[n] is not empty): return M[n]
  else:
    M[n] ← max{FindOPT(n-1), vₙ + FindOPT(p(n))}
    return M[n]
```

- What is the running time of **`FindOPT(n)`**?

$O(n)$

Interval Scheduling: Memoized





$$M[3] = \max\{\overset{4}{M[2]}, 4 + \overset{2}{M[1]}\}$$

$$M[4] = \max\{M[3], 7 + M[0]\}$$

$$M[5] = \max\{M[4], 2 + M[3]\}$$

$$M[6] = \max\{M[5], 1 + M[3]\}$$

M[0]	M[1]	M[2]	M[3]	M[4]	M[5]	M[6]
0	2	4	6	7	8	8

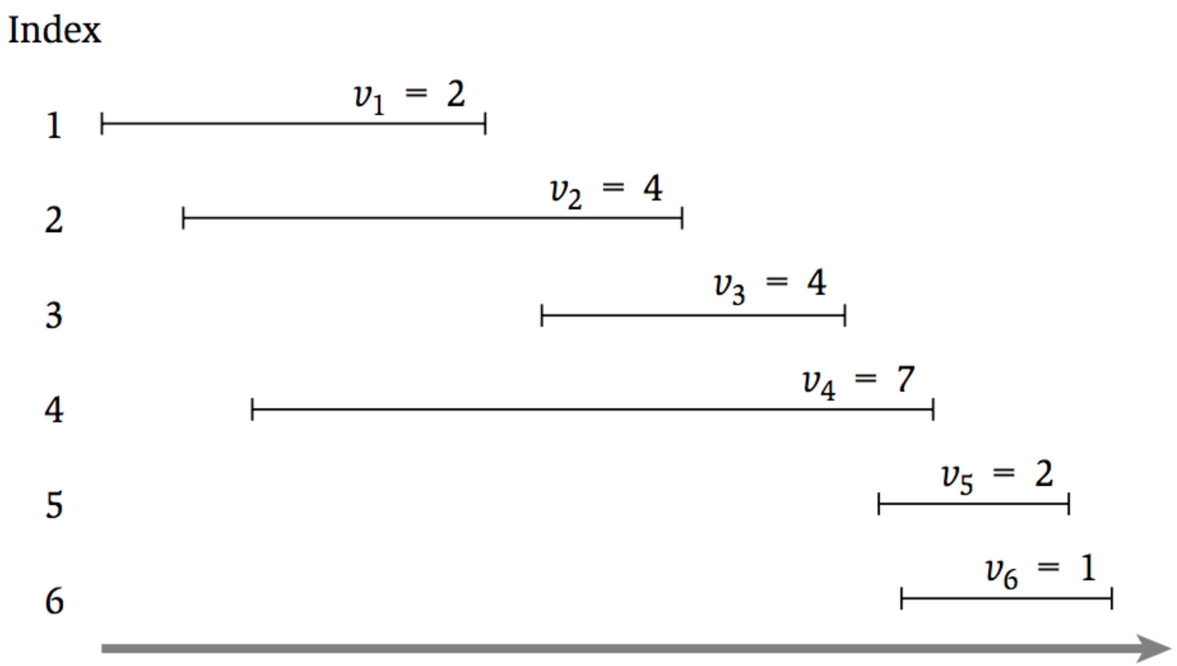
Interval Scheduling: Bottom Up

```
FindOPT(n):
  M[0] ← 0, M[1] ← v_1
  for (i = 2,…,n):
    M[i] ← max{M[i-1], v_i + M[p(i)]}
  return M[n]
```

$O(n)$

- What is the running time of **`FindOPT(n)`**?

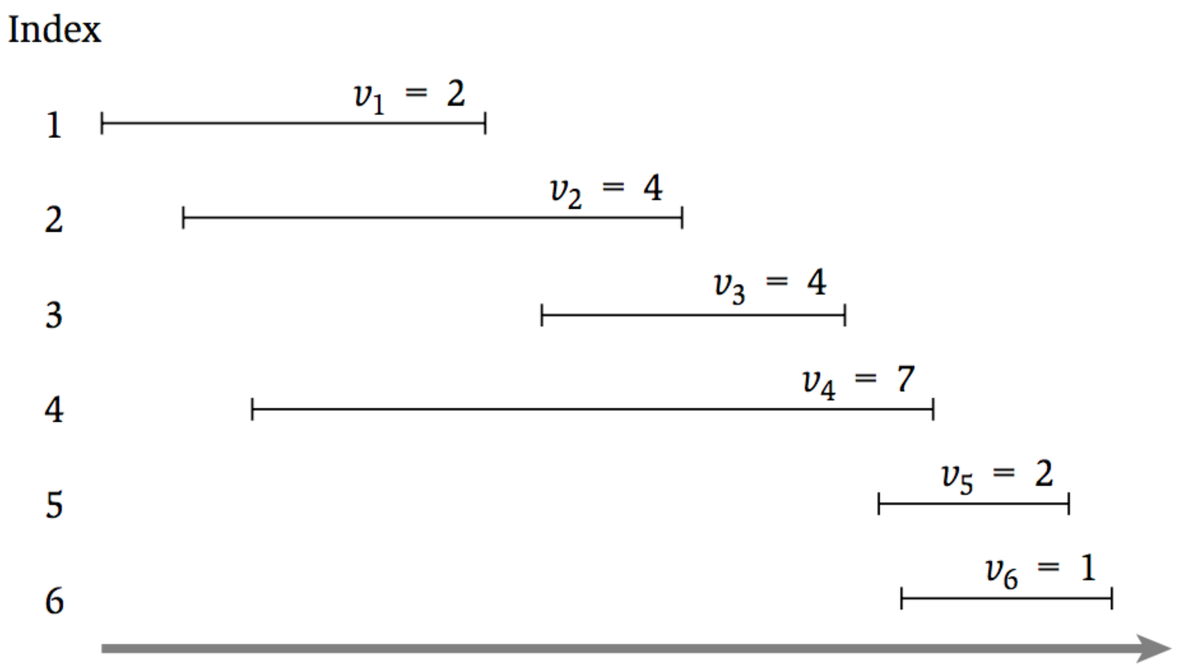
Interval Scheduling: Bottom Up



M[0]	M[1]	M[2]	M[3]	M[4]	M[5]	M[6]
0	2	4				

Finding the Optimal Solution

- But we want a schedule, not a value!



M[0]	M[1]	M[2]	M[3]	M[4]	M[5]	M[6]
0	2	4	6	7	8	8

Dynamic Programming Recipe

- **Recipe:**

  (1) identify a set of **subproblems**

  (2) relate the subproblems via a **recurrence**

  (3) find an **efficient implementation** of the recurrence (top down or bottom up)

  (4) **reconstruct the solution** from the DP table

Finding the Optimal Solution

DP Table which we assume is already filled up

```
FindSched(M,n):
  if (n = 0): return ∅
  elseif (n = 1): return {1}
  elseif (vn + M[p(n)] > M[n-1]):
    return {n} + FindSched(M,p(n))
  else:
    return FindSched(M,n-1)
```

F_n

$\downarrow$

F_{n-1}

$\downarrow$

$F_{p(n-1)}$

$\downarrow$

$\vdots$

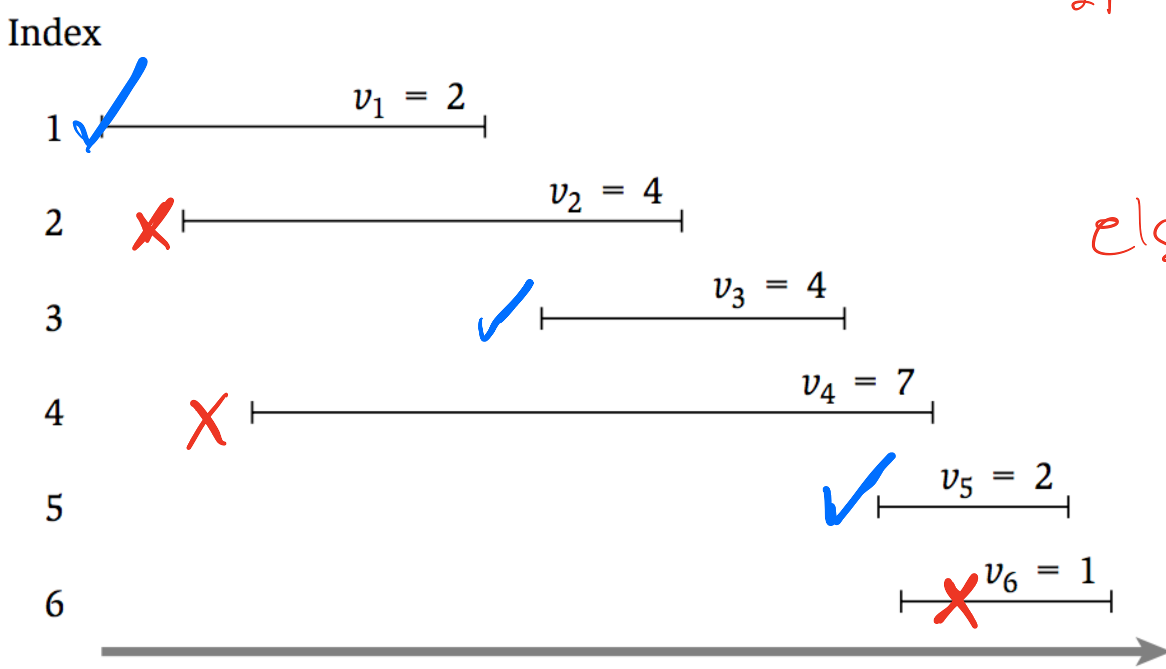
$F[0]$

$\leq O(n)$

- What is the running time of **FindSched(n)**?

$O(n)$

Finding the Optimal Solution



If $v_6 + M[p[6]] > M[5]$

1 6 8

return $\{6\} + FO(p(6))$

else

return $FO(6-1)$

$v_5 + M[p[5]] > M[4]$

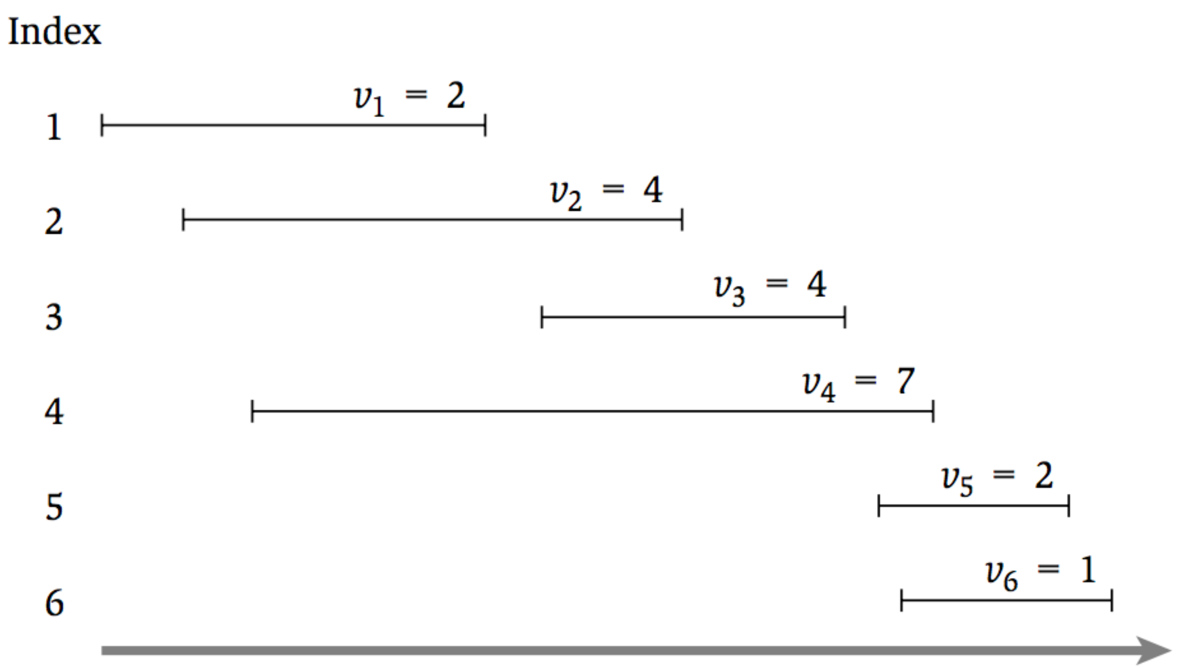
2 6 7

$v_3 + M[p[3]] > M[2]$

4 2 4

M[0]	M[1]	M[2]	M[3]	M[4]	M[5]	M[6]
0	2	4	6	7	8	8

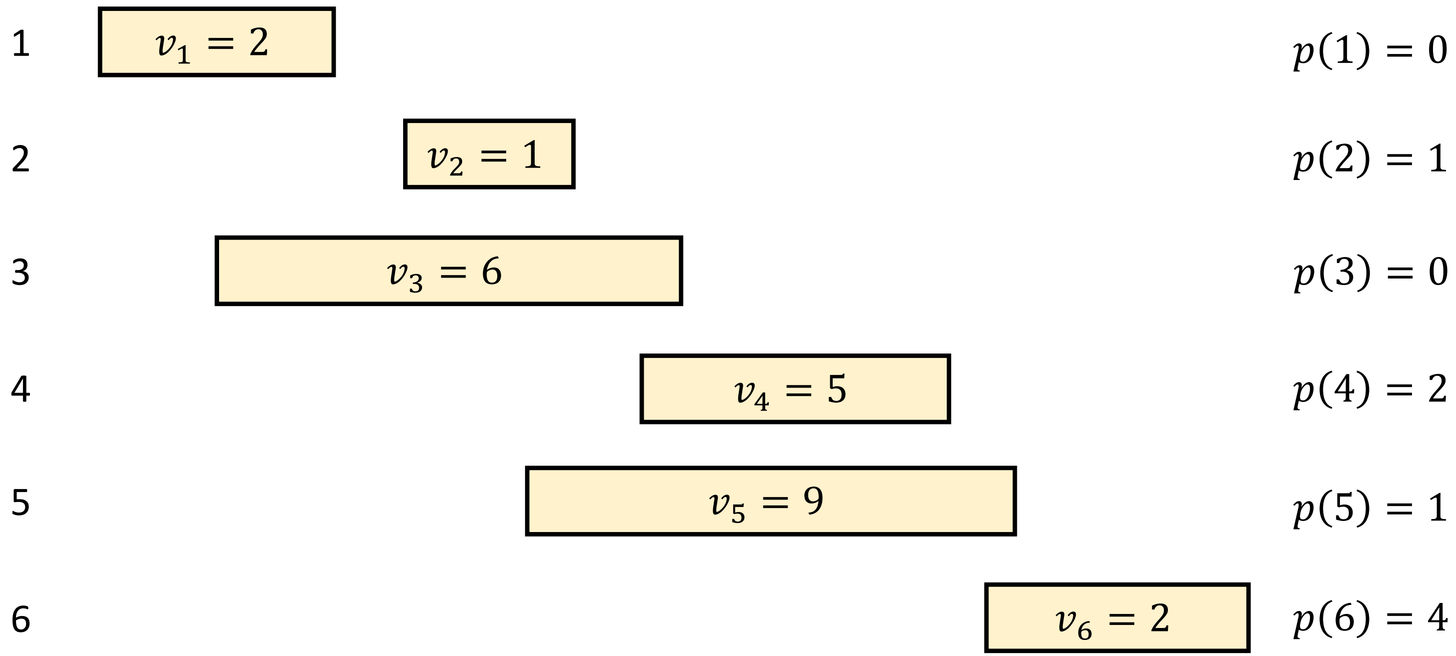
How much space is used?



$O(n)$

M[0]	M[1]	M[2]	M[3]	M[4]	M[5]	M[6]
0	2	4	6	7	8	8

Now You Try



M[0]	M[1]	M[2]	M[3]	M[4]	M[5]	M[6]

Dynamic Programming Recap

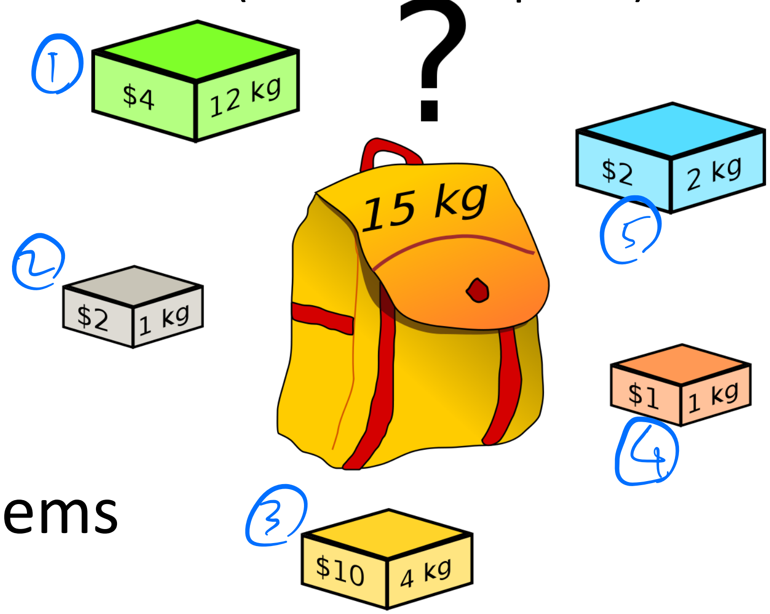
- Express the optimal solution as a **recurrence**
  - Identify a small number of subproblems
  - Relate the optimal solution on subproblems
- Efficiently solve for the **value** of the optimum
  - Simple implementation is exponential time, but top-down and bottom-up are linear time
    - Top-Down: recursive, store solution to subproblems
    - Bottom-Up: iterate through subproblems in order
- Find the **solution** using the table of **values**

Dynamic Programming

a. Fibonacci Series
b. Weighted Interval Scheduling
c. Knapsack

The Knapsack Problem

(source: Wikipedia)



- **Input:** n items for your knapsack
  - value v_i and a weight $w_i \in \mathbb{N}$ for n items
  - capacity of your knapsack $T \in \mathbb{N}$
- **Output:** the most valuable subset of items that fits in the knapsack
  - Subset $S \subseteq \{1, \dots, n\}$
  - Value $V_S = \sum_{i \in S} v_i$ as large as possible
  - Weight $W_S = \sum_{i \in S} w_i$ at most T
- **Want:** $\mathbf{argmax}_{S \subseteq \{1,\dots,n\}} V_S$ s.t. $W_S \le T$
- **SubsetSum:** $v_i = w_i$,
- **TugOfWar:** $v_i = w_i, T = \frac{1}{2} \sum_i v_i$

$n = 5$ $T = 15$

$v_1 = 4$ $w_1 = 12$

$v_2 = 2$ $w_2 = 1$

$v_3 = 10$ $w_3 = 4$

$v_4 = 1$ $w_4 = 1$

$v_5 = 2$ $w_5 = 2$

Do we really need DP?

Items with large $\frac{v_i}{w_i}$ seem like good choices…

Ex. *T = 8,* $(v_1 = 6, w_1 = 5), (v_2 = 4, w_2 = 4), (v_3 = 4, w_3 = 4)$ ✓

- Strategy 1: Repeatedly pick items that fit with largest $\frac{v_i}{w_i}$

we only pick item 1 and gain a value of 6

- Is this optimal?

opt is $\{2, 3\}$ which gives value 8.