CS3000: Algorithms & Data
Paul Hand

Lecture 9:

- Dynamic Programming
- Interval Scheduling

Feb 6, 2019

Dynamic Programming

Dynamic Programming

Dynamic programming is careful recursion

- Break the problem up into small pieces
- Recursively solve the smaller pieces
- Store outcomes of smaller pieces that get called multiple times
- **Key Challenge:** identifying the pieces

Dynamic Programming: Interval Scheduling

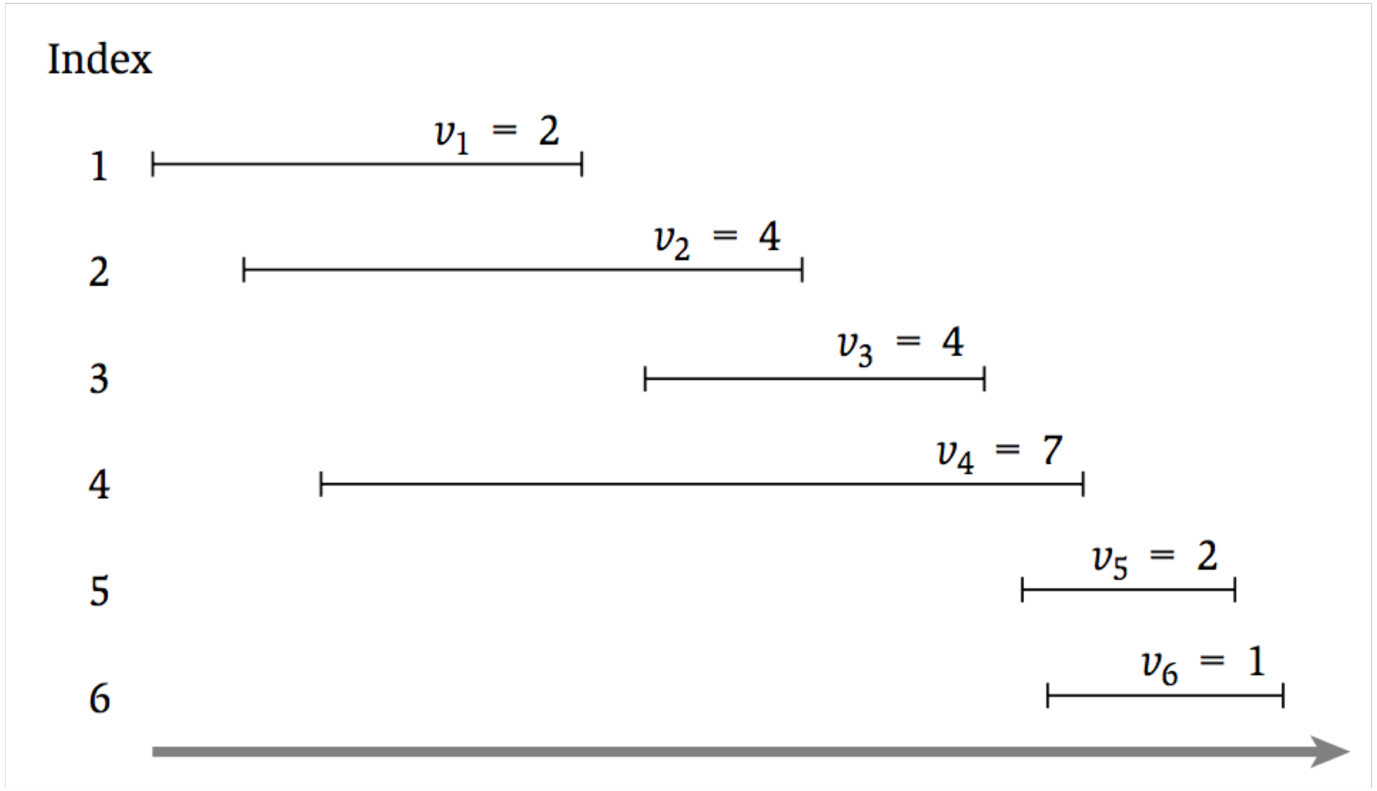
Interval Scheduling

- 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:

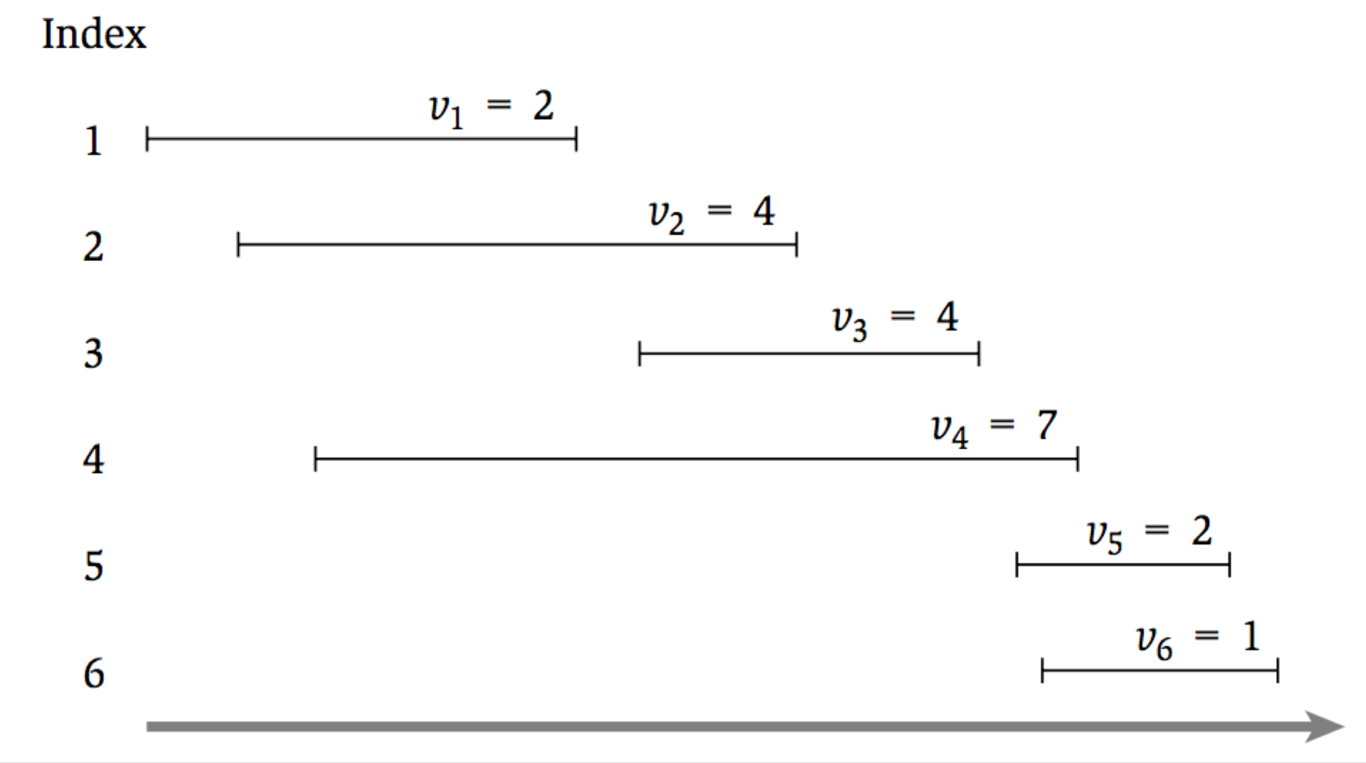
A **schedule** is a subset of intervals $S \subseteq \{1, \ldots, n\}$
A schedule S is **compatible** if no $i, j \in S$ overlap
The **total value** of S is $\sum_{i \in S} v_i$



Activity: Find the schedule that maximizes the total value of the intervals.

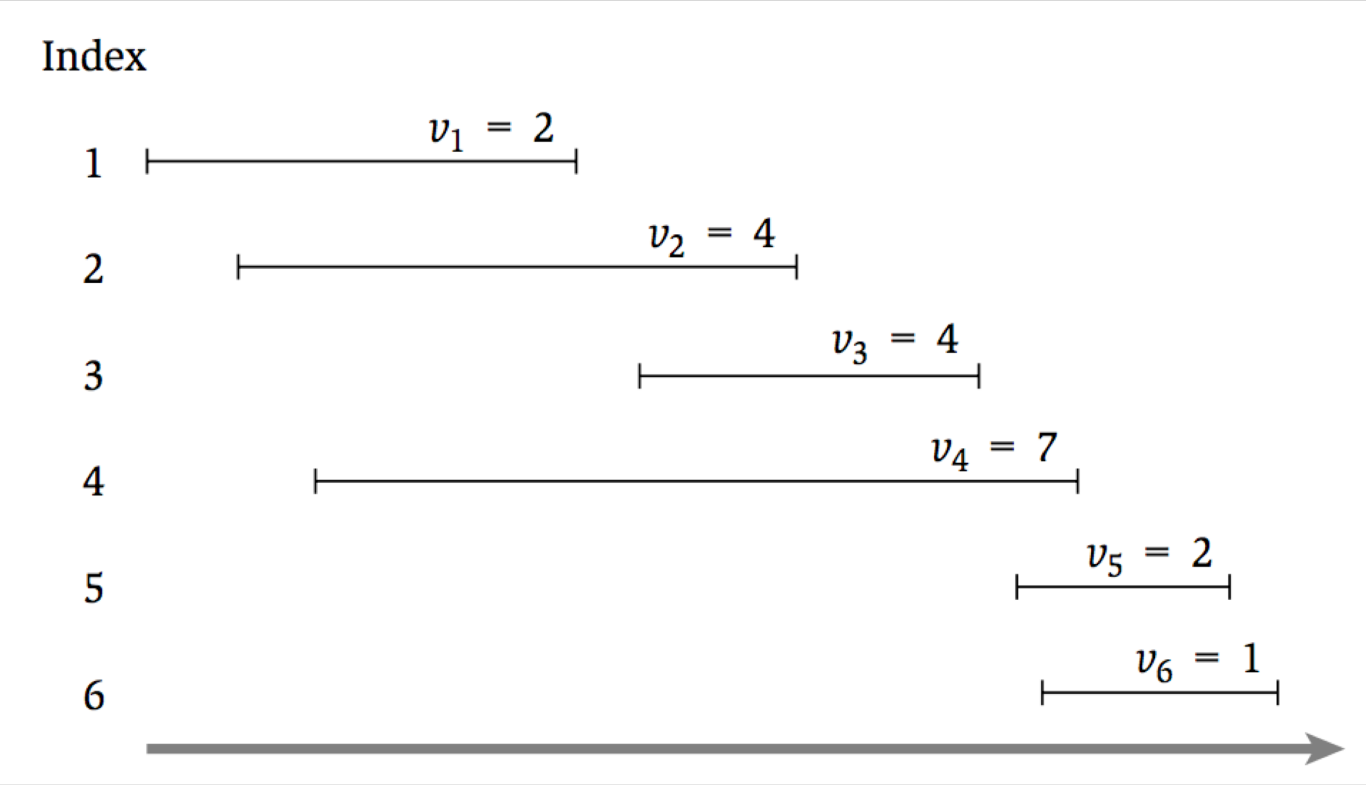
Possible Algorithms

- Choose intervals in decreasing order of v_i



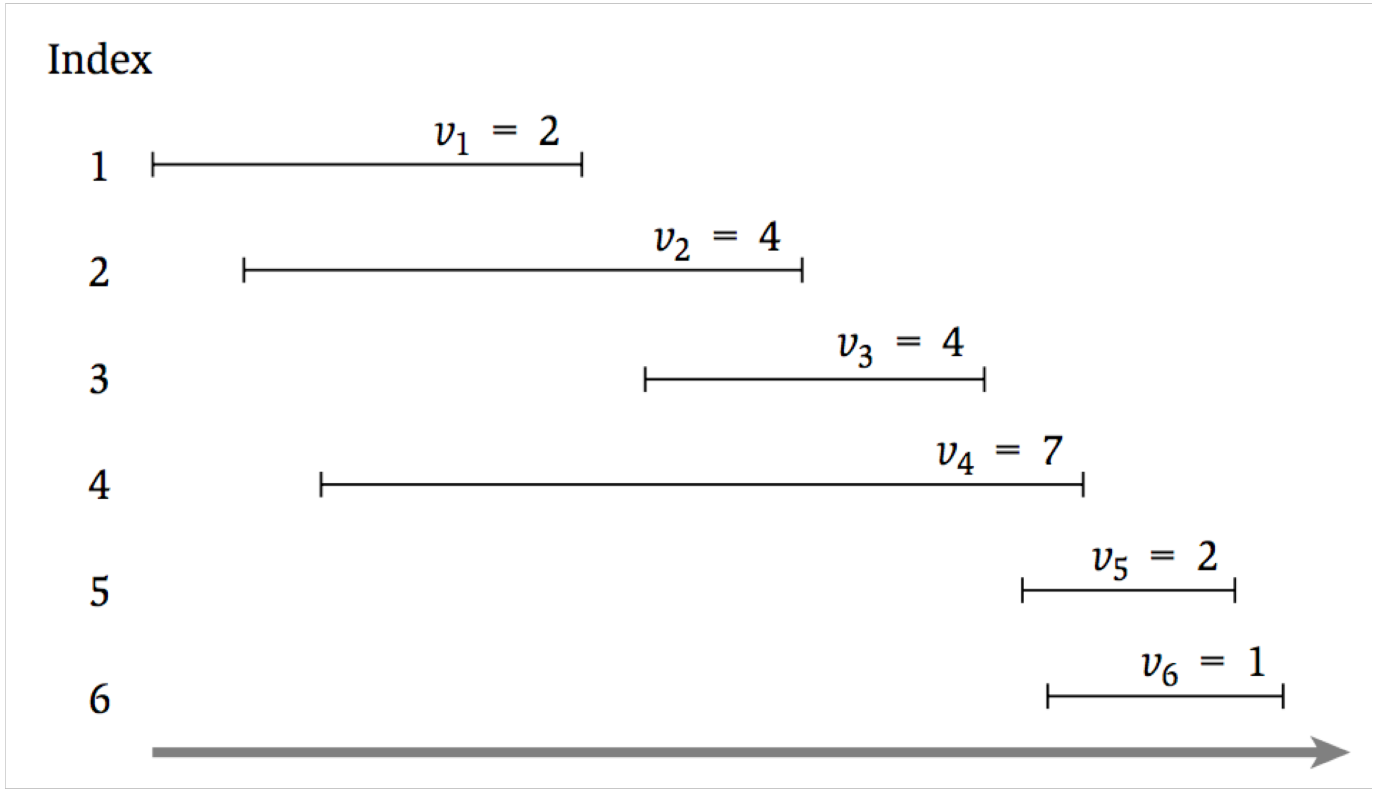
Possible Algorithms

- Choose intervals in increasing order of s_i



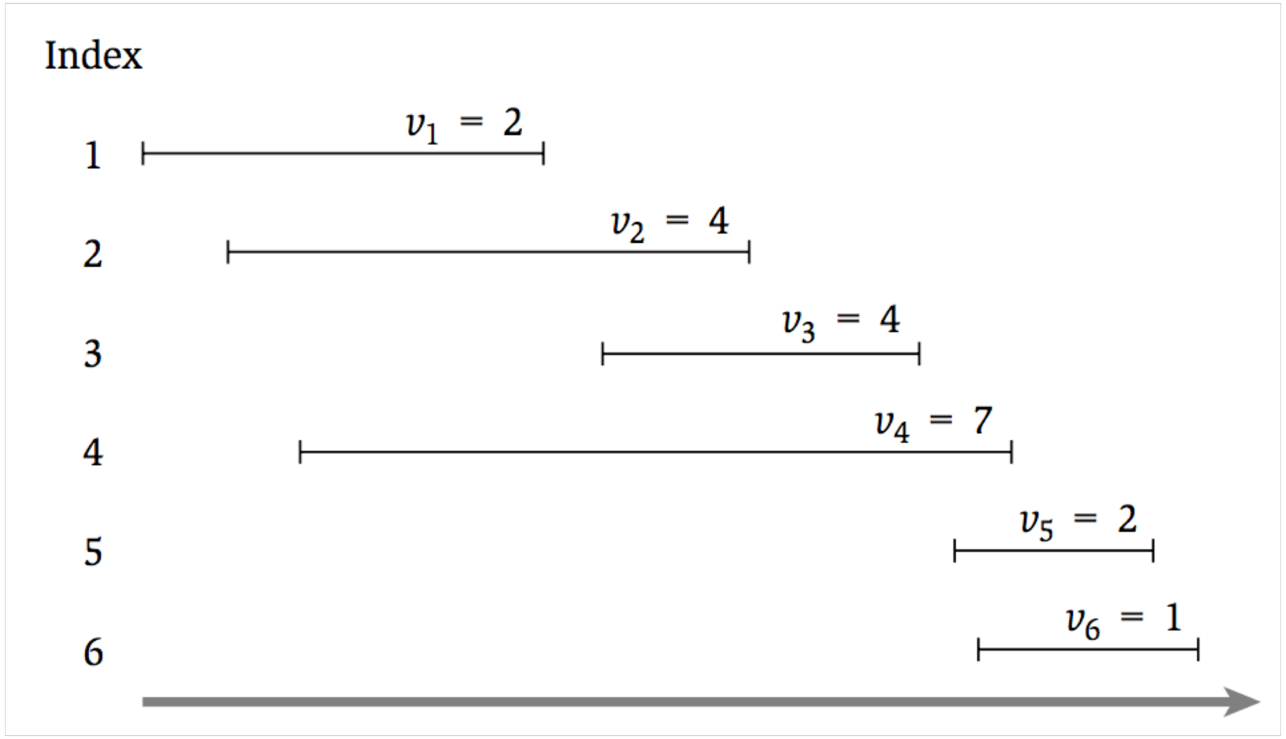
Possible Algorithms

- Choose intervals in increasing order of $f_i - s_i$



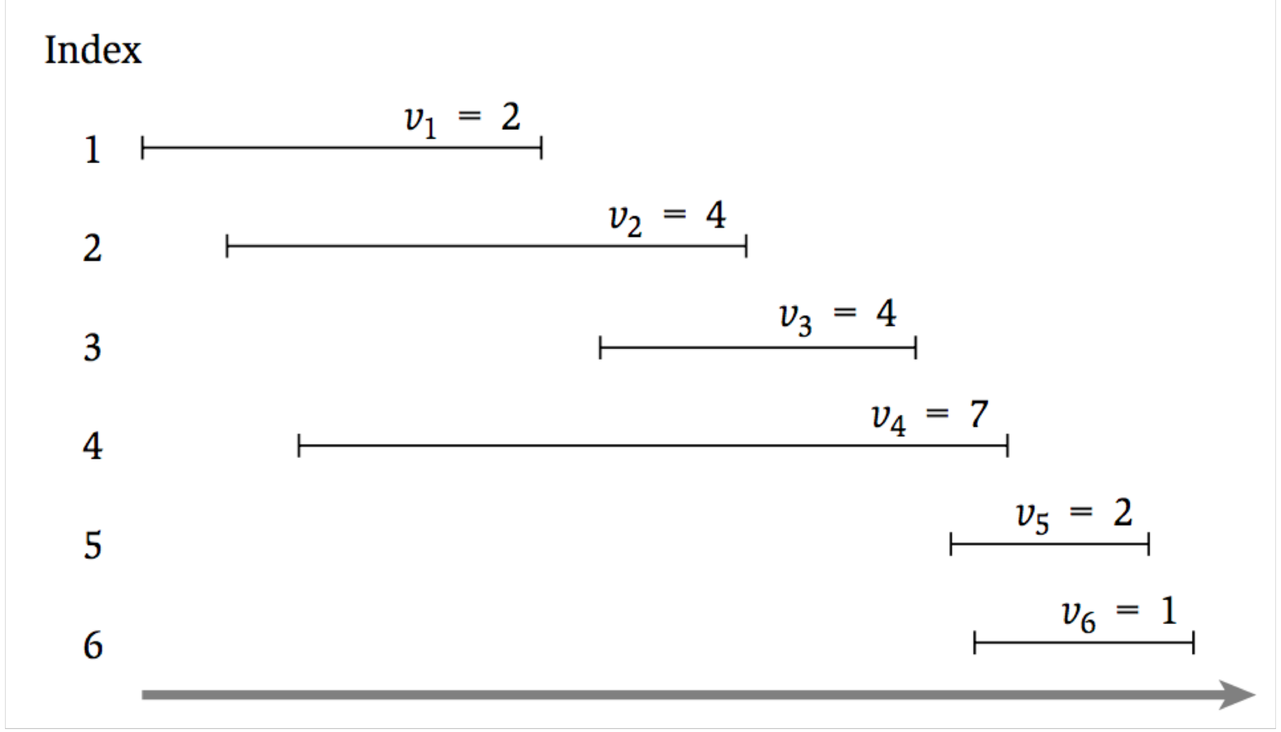
A Recursive Formulation

- Let O be the **optimal** schedule
- **Case 1:** Final interval is not in O (i.e. $6 \notin O$)
  - Then O must be the optimal solution for $\{1, \ldots, 5\}$



A Recursive Formulation

- Let O be the **optimal** schedule
- **Case 2:** Final interval is in O (i.e. $6 \in O$)
  - Then O must be 6 + the optimal schedule for $\{1, \dots, 3\}$



A Recursive Formulation

- Let O_i be the **optimal schedule** using only the intervals $\{1, \ldots, i\}$
- **Case 1:** Final interval is not in O $(i \notin O)$
  - Then O must be the optimal solution for $\{1, \ldots, i-1\}$
- **Case 2:** Final interval is in O $(i \in O)$
  - 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 must be i + the optimal solution for $\{1, \ldots, p(i)\}$

A Recursive Formulation

- Let $OPT(i)$ be the **value of the optimal schedule** using only the intervals $\{1, \dots, i\}$
- **Case 1:** Final interval is not in O ($i \notin O$)
  - Then O must be the optimal solution for $\{1, \dots, i-1\}$
- **Case 2:** Final interval is in O ($i \in O$)
  - 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 must be i + the optimal schedule for $\{1, \dots, p(i)\}$

- $OPT(i) = \max\{OPT(i-1), v_i + OPT\big(p(i)\big)\}$
- $OPT(0) = 0, OPT(1) = v_1$

Interval Scheduling: Take I

```
// All inputs are global vars
FindOPT(n):
  if (n = 0): return 0
  elseif (n = 1): return $v_1$
  else:
    return max{FindOPT(n-1), $v_n$ + FindOPT(p(n))}
```

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

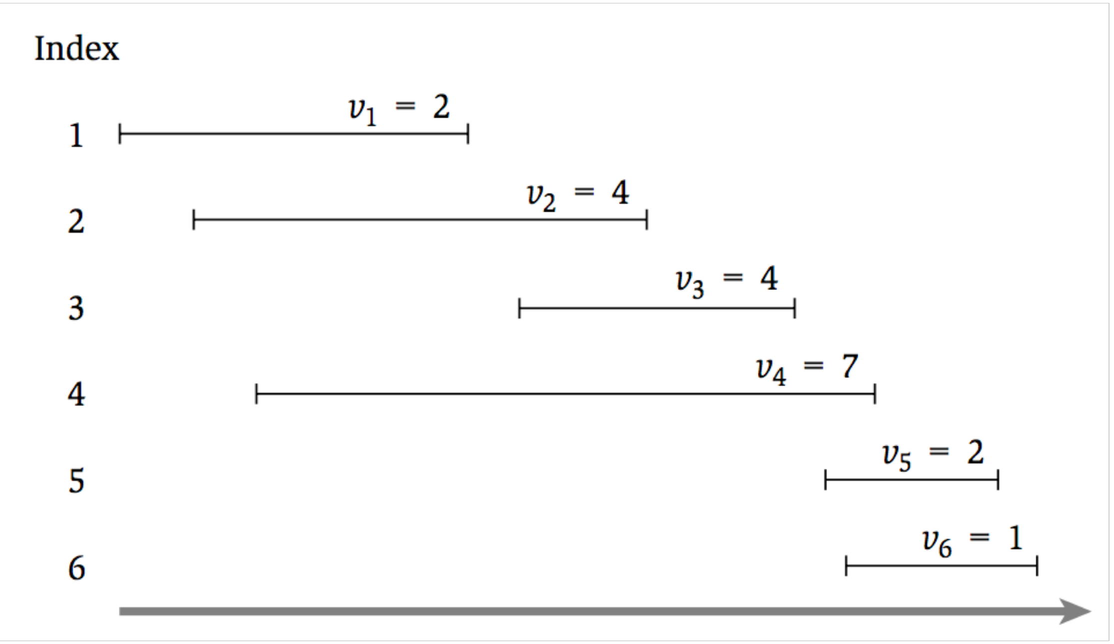
Interval Scheduling: Take II

```
// All inputs are global vars
M ← empty array, M[0] ← 0, M[1] ← $v_1$

FindOPT(n):
  if (M[n] is not empty): return M[n]
  else:
    M[n] ← max{FindOPT(n-1), $v_n$ + FindOPT(p(n))}
    return M[n]
```

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

Interval Scheduling: Take II



```
// All inputs are global vars
M ← empty array, M[0] ← 0, M[1] ← $v_1$

FindOPT(n):
  if (M[n] is not empty): return M[n]
  else:
    M[n] ← max{FindOPT(n-1), $v_n$ + FindOPT(p(n))}
    return M[n]
```

	P[1]	P[2]	P[3]	P[4]	P[5]	P[6]

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

Interval Scheduling: Take III

```
// All inputs are global vars
FindOPT(n):
  M[0] ← 0, M[1] ← 1
  for (i = 2,…,n):
    M[i] ← max{M[i-1], $v_i$ + M[p(i)]}
  return M[n]
```

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

Finding the Optimal Solution

- Let $OPT(i)$ be the **value of the optimal schedule** using only the intervals $\{1, \ldots, i\}$
- **Case 1:** Final interval is not in O ($i \notin O$)
- **Case 2:** Final interval is in O ($i \in O$)

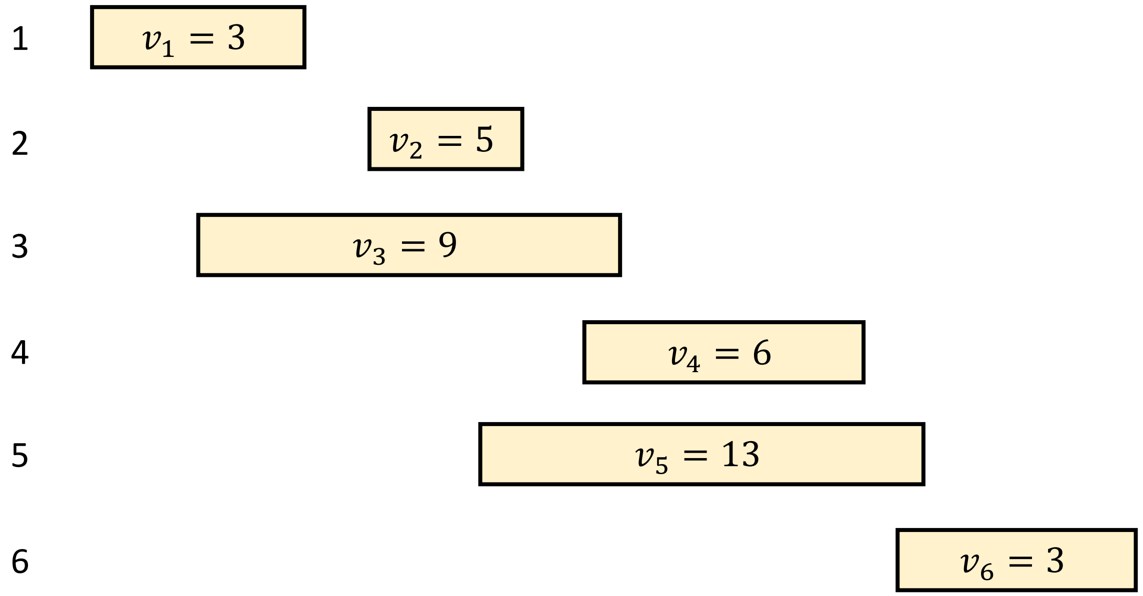
- $OPT(i) = \max\{OPT(i-1), v_n + OPT\big(p(i)\big)\}$

Interval Scheduling: Take III

```
// All inputs are global vars
FindSched(M,n):
  if (n = 0): return ∅
  elseif (n = 1): return {1}
  elseif ($v_n$ + M[p(n)] > M[n-1]):
    return {n} + FindSched(M,p(n))
  else:
    return FindSched(M,n-1)
```

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

Now You Try



1 $v_1 = 3$ $p(1) = 0$

2 $v_2 = 5$ $p(2) = 1$

3 $v_3 = 9$ $p(3) = 0$

4 $v_4 = 6$ $p(4) = 2$

5 $v_5 = 13$ $p(5) = 1$

6 $v_6 = 3$ $p(6) = 4$

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
  - Top-Down: store solution to subproblems
  - Bottom-Up: iterate through subproblems in order

- Find the **solution** using the table of **values**