

# Lecture 34

CSE 331

Nov 17, 2014

# HW grading schedule

Sorry for the delays!

note ☆ 4 views Actions ▾

## Returning HW 7 next week

Hi Everyone,

Returning HW 7 will be delayed just a bit.

HW 7 will be returned **next Monday 11/ 24**

HW 8 & HW 9 will be returned on **Monday 12/1** (after Thanksgiving break).

Thanks,  
Frank

[homework7](#) [homework8](#) [homework9](#)

[edit](#) good note | 0 13 minutes ago by Frank Schoeneman

# Weighted Interval Scheduling

Input:  $n$  jobs  $(s_i, f_i, v_i)$

Output: A schedule  $S$  s.t. no two jobs in  $S$  have a conflict

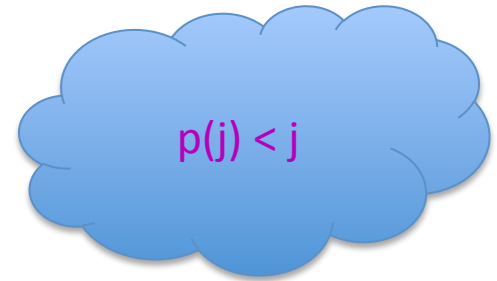
Goal:  $\max \sum_{i \in S} v_j$

Assume: jobs are sorted by their finish time

# Couple more definitions

$p(j)$  = largest  $i < j$  s.t.  $i$  does not conflict with  $j$

= 0 if no such  $i$  exists



$OPT(j)$  = optimal value on instance  $1, \dots, j$

# Property of OPT

$j$  in  $\text{OPT}(j)$

$j$  not in  $\text{OPT}(j)$

$$\text{OPT}(j) = \max \{ v_j + \text{OPT}(p(j)), \text{OPT}(j-1) \}$$

Given  $\text{OPT}(1), \dots, \text{OPT}(j-1)$ ,  
how can one figure out if  $j$   
is in optimal solution or not?



# A recursive algorithm

Compute-Opt( $j$ )

Correct for  $j=0$

Proof of correctness by induction on  $j$

If  $j = 0$  then return 0

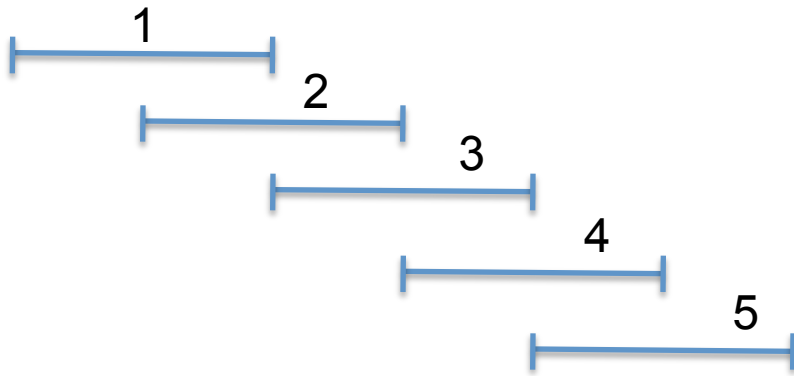
return  $\max \{ v_j + \text{Compute-Opt}(p(j)), \text{Compute-Opt}(j-1) \}$

$= \text{OPT}(p(j))$

$= \text{OPT}(j-1)$

$$\text{OPT}(j) = \max \{ v_j + \text{OPT}(p(j)), \text{OPT}(j-1) \}$$

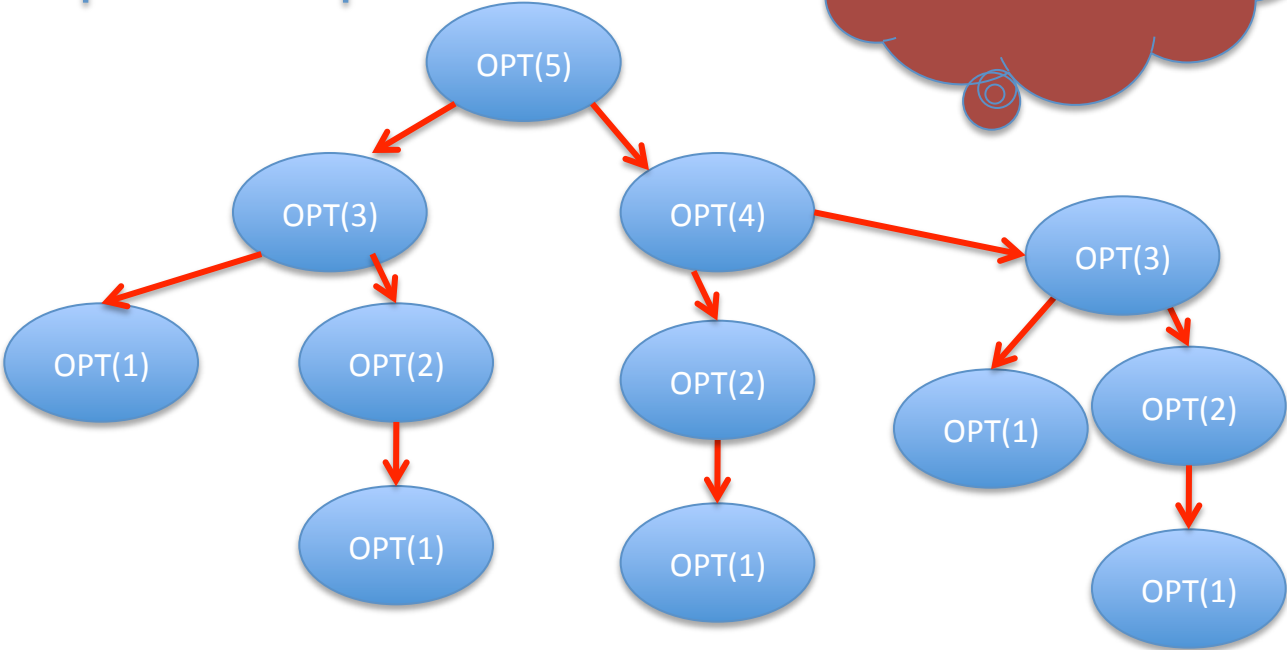
# Exponential Running Time



$$p(j) = j - 2$$

Only 5 OPT values!

Formal proof: Ex.







# Using Memory to be smarter

```
Pow (a,n)
```

```
⋮
```

```
// n is even and  $\geq 2$ 
```

```
return Pow(a,n/2) * Pow(a, n/2)
```

```
⋮
```

$O(n)$  as we recompute!

```
Pow (a,n)
```

```
⋮
```

```
// n is even and  $\geq 2$ 
```

```
t= Pow(a,n/2)
```

```
return t * t
```

```
⋮
```

$O(\log n)$  as we compute only once

How many distinct OPT values?

# A recursive algorithm

M-Compute-Opt(j)

If  $j = 0$  then return 0

If  $M[j]$  is not null then return  $M[j]$

$M[j] = \max \{ v_j + \text{M-Compute-Opt}(p(j)), \text{M-Compute-Opt}(j-1) \}$

return  $M[j]$

M-Compute-Opt(j)  
= OPT(j)

Run time =  $O(\# \text{ recursive calls})$

# Bounding # recursions

M-Compute-Opt(j)

If  $j = 0$  then return 0

If  $M[j]$  is not null then return  $M[j]$

$M[j] = \max \{ v_j + \text{M-Compute-Opt}(p(j)), \text{M-Compute-Opt}(j-1) \}$

return  $M[j]$

$O(n)$  overall

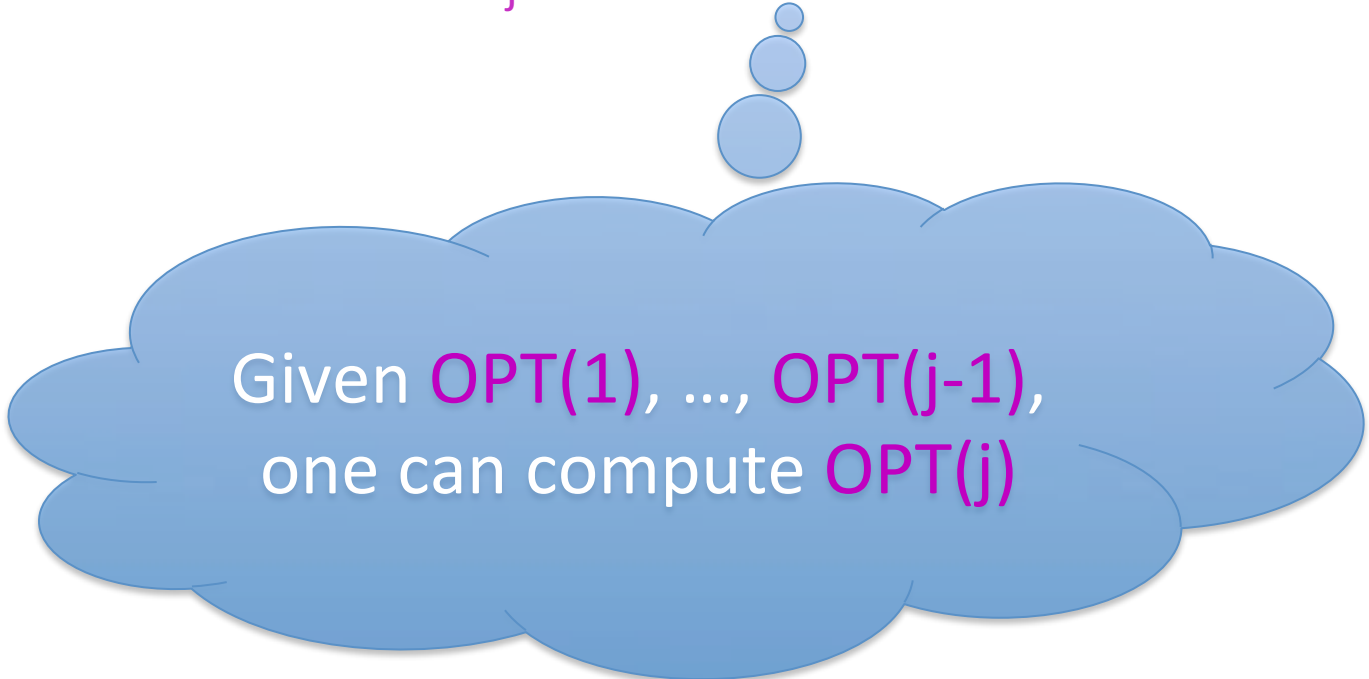
Whenever a recursive call is made an  $M$  value is assigned

At most  $n$  values of  $M$  can be assigned



# Property of OPT

$$\text{OPT}(j) = \max \{ v_j + \text{OPT}(p(j)), \text{OPT}(j-1) \}$$



Given  $\text{OPT}(1), \dots, \text{OPT}(j-1)$ ,  
one can compute  $\text{OPT}(j)$

# Recursion+ memory = Iteration

Iteratively compute the OPT(j) values

Iterative-Compute-Opt

$M[0] = 0$

For  $j=1, \dots, n$

$M[j] = \max \{ v_j + M[p(j)], M[j-1] \}$

$M[j] = \text{OPT}(j)$

$O(n)$  run time





# Reading Assignment

Sec 6.1, 6.2 of [KT]



# When to use Dynamic Programming

There are polynomially many sub-problems



Richard Bellman

Optimal solution can be computed from solutions to sub-problems

There is an ordering among sub-problem that allows for iterative solution