

# Lecture 33

CSE 331

Nov 14, 2014

# HW 8 due today

Place Q1, Q2 and Q3 in separate piles

I will not accept HWs after 1:15pm

**DO NOT FORGET TO WRITE DOWN YOUR SOURCES**

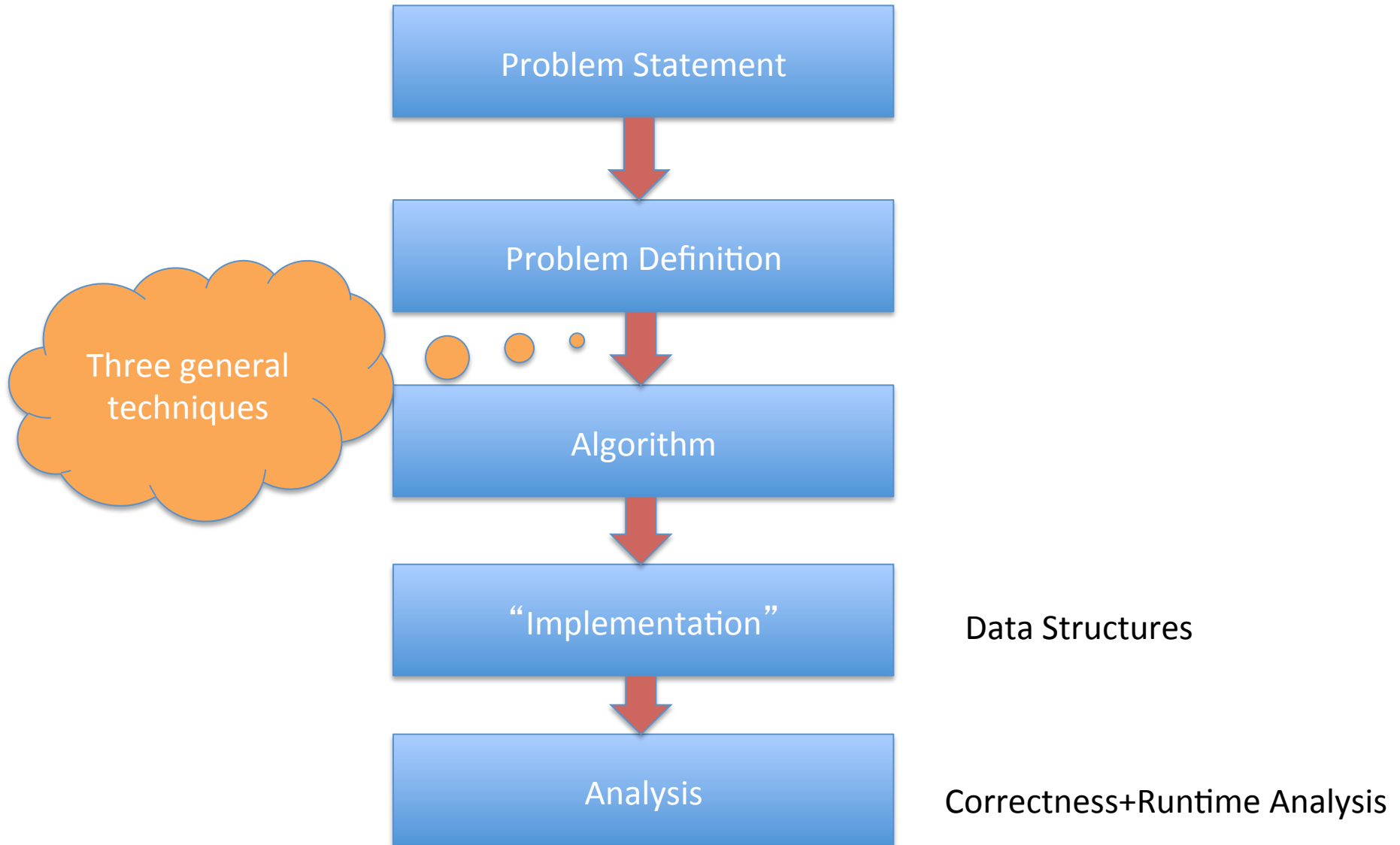
# Related HW stuff

HW 9 has been posted on piazza

Solutions to HW 8 at the END of the lecture

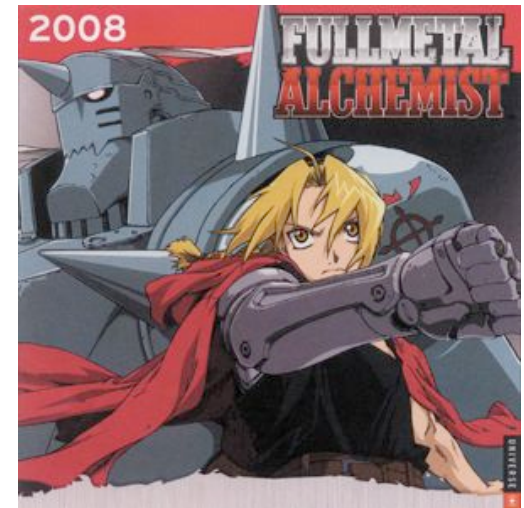
Can pick up graded HW 7 from Monday

# High level view of CSE 331



# Greedy Algorithms

Natural algorithms



Reduced exponential running time to polynomial

# Divide and Conquer

Recursive algorithmic paradigm



Reduced large polynomial time to smaller polynomial time

# A new algorithmic technique

## Dynamic Programming

# Dynamic programming vs. Divide & Conquer





# Same same because

Both design recursive algorithms



# Different because

Dynamic programming is smarter about solving recursive sub-problems



# End of Semester blues

Can only do one thing at any day: what is the optimal schedule to obtain maximum value?



Write up a term paper (10)

Party! (2)

Exam study (5)

331 HW (3)

Project (30)

Monday

Tuesday

Wednesday

Thursday

Friday

# Previous Greedy algorithm

Order by end time and pick jobs greedily

Greedy value =  $5+2+3=10$

Write up a term paper (10)

Party! (2)

Exam study (5)

331 HW (3)

Project (30)

OPT = 30

Monday

Tuesday

Wednesday

Thursday

Friday



# HW 8 due today

Place Q1, Q2 and Q3 in separate piles

I will not accept HWs after 1:15pm

**DO NOT FORGET TO WRITE DOWN YOUR SOURCES**

# Today's agenda

Formal definition of the problem

Start designing a recursive algorithm for the problem



# 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

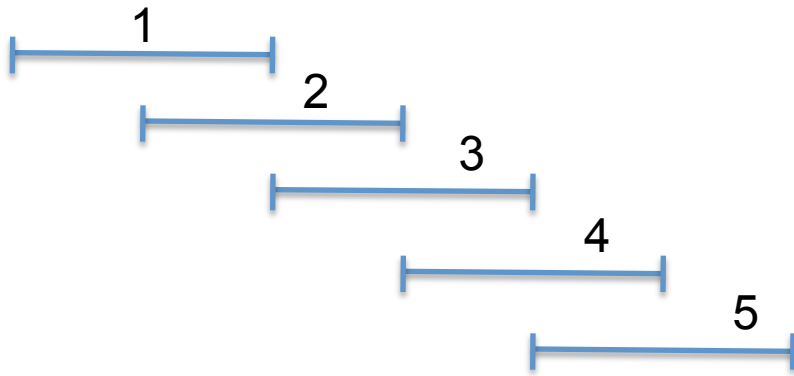
return  $\max \{ v_j + \text{Compute-Opt}(p(j)), \text{Compute-Opt}(j-1) \}$

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

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

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

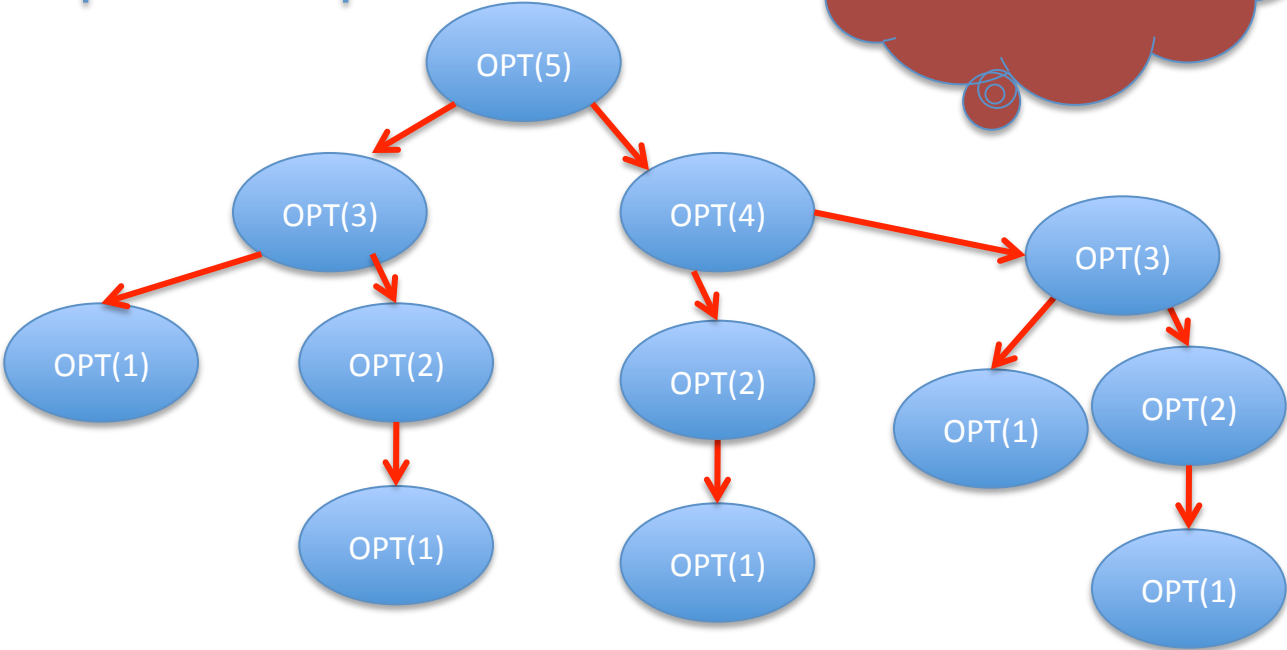
# Exponential Running Time



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

Only 5 OPT values!

Formal proof: Ex.





How many distinct OPT values?

# 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

# 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 of assigned

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





# 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