Objectives

- Divide and Conquer: Matrix Multiplication
- Introduction to Dynamic Programming
  - Weighted interval scheduling

Divide-and-Conquer Multiplication: Warmup

- To multiply 2 n-digit integers:
  - Multiply 4 (pairs of) ½n-digit integers
  - Add 2 ½n-digit integers and shift to obtain result

Higher order bits — Lower order bits — Shift

x=10001101

x_1=1000 x_0=1101

$$x = 2^{n/2} \cdot x_1 + x_0$$

$$y = 2^{n/2} \cdot y_1 + y_0$$

$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0$$

A B C D

What is the recurrence relation?
- How many subproblems?
- What is merge cost?
- What is its runtime?

Divide-and-Conquer Multiplication: Warmup

- To multiply 2 n-digit integers:
  - Multiply 4 (pairs of) ½n-digit integers
  - Add 2 ½n-digit integers and shift to obtain result

Higher order bits — Lower order bits — Shift

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = \underset{A}{2^n \cdot x_1 y_1} + 2^{n/2} \cdot (\underset{B}{x_1 y_0} + \underset{C}{x_0 y_1}) + \underset{D}{x_0 y_0}$$

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

assumes n is a power of 2

Not an improvement over brute force

Karatsuba Multiplication


Anatolii Alexeevich Karatsuba

- To multiply two n-digit integers:
  - Add 2 ½n digit integers
  - Multiply 3 ½n-digit integers
  - Add, subtract, and shift ½n-digit integers to obtain result

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0$$
$$= \underset{A}{2^n \cdot x_1 y_1} + 2^{n/2} \cdot \left(\underset{B}{(x_1 + x_0)(y_1 + y_0)} - \underset{A}{x_1 y_1} - \underset{C}{x_0 y_0} \right) + \underset{C}{x_0 y_0}$$

What is the recurrence relation? Runtime?

Karatsuba Multiplication

- Theorem. [Karatsuba-Ofman, 1962]
  Can multiply two n-digit integers in $O(n^{1.585})$ bit operations

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^{n} \cdot \underset{A}{x_1 y_1} + 2^{n/2} \cdot \big(\underset{B}{(x_1 + x_0)(y_1 + y_0)} - \underset{A}{x_1 y_1} - \underset{C}{x_0 y_0} \big) + \underset{C}{x_0 y_0}
\end{aligned}
$$

$$
\begin{aligned}
&T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \\
&\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})
\end{aligned}
$$

MATRIX MULTIPLICATION

Matrix Multiplication

- Given 2 n-by-n matrices A and B, compute C = AB

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

  - Ex: $c_{12} = a_{11} b_{12} + a_{12} b_{22} + a_{13} b_{32} + \ldots + a_{1n} b_{n2}$

Row 1 of a   Column 2 of b

Solve using brute force …

Matrix Multiplication

- Given 2 n-by-n matrices A and B, compute C = AB

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

  - Ex: $c_{12} = a_{11} b_{12} + a_{12} b_{22} + a_{13} b_{32} + \ldots + a_{1n} b_{n2}$

- Brute force. $\Theta(n^3)$ arithmetic operations
- Fundamental question: Can we improve upon brute force?

Matrix Multiplication: Warmup

- Divide: partition A and B into ½n-by-½n blocks
- Conquer: multiply 8 ½n-by-½n recursively
- Combine: add appropriate products using 4 matrix additions

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

Recurrence relation? Runtime?

Matrix Multiplication: Warmup

- Divide: partition A and B into ½n-by-½n blocks
- Conquer: multiply 8 ½n-by-½n recursively
- Combine: add appropriate products using 4 matrix additions

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$\mathrm{T}(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow \mathrm{T}(n) = \Theta(n^3)$$

Matrix Multiplication: Key Idea

Trade expensive multiplication for
less expensive addition/subtraction

- Multiply 2-by-2 block matrices with only 7 multiplications and 15 additions

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

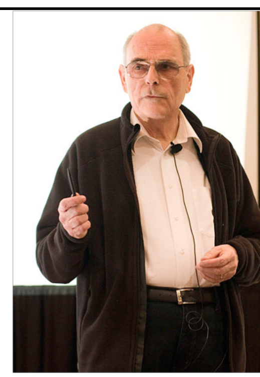
$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

Mar 15, 2019 CSCI211 - Sprenkle 11

Fast Matrix Multiplication

[Strassen, 1969]

- Divide: partition A and B into ½n-by-½n blocks
- Compute: 14 ½n-by-½n matrices via 10 matrix additions
- Conquer: multiply 7 ½n-by-½n matrices recursively
- Combine: 7 products into 4 terms using 8 matrix additions
- Analysis.
  - Assume n is a power of 2.
  - T(n) = # arithmetic operations.



Volker Strassen

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Mar 15, 2019 CSCI211 - Sprenkle 12

Fast Matrix Multiplication in Practice

- Implementation issues:
  problems putting theory into practice
  - Sparsity
  - Caching effects
  - Numerical stability
    - Theoretically correct but possible problems with round off errors, etc
  - Odd matrix dimensions
- Crossover to classical algorithm around
  n = 128

Fast Matrix Multiplication in Practice

- Common misperception:
  "Strassen is only a theoretical curiosity."
  - Advanced Computation Group at Apple Computer reports **8x** speedup on G4 Velocity Engine when n ~2,500
  - Range of instances where it's useful is a subject of controversy
- Can "Strassenize" Ax=b, determinant, eigenvalues, and other matrix ops

Fast Matrix Multiplication in Theory

- Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?
- A. Yes! [Strassen, 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$
- Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
- A. Impossible [Hopcroft and Kerr, 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$
- Q. Two 3-by-3 matrices with only 21 scalar multiplications?
- A. Also impossible $\Theta(n^{\log_3 21}) = O(n^{2.77})$
- Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?
- A. Yes! [Pan, 1980] $\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$

- Decimal wars.
  - December 1979: $O(n^{2.521813})$
  - January 1980: $O(n^{2.521801})$

Mar 15, 2019 CSCI211 - Sprenkle 15

Fast Matrix Multiplication in Theory

- Best known. $O(n^{2.376})$
  [Coppersmith-Winograd, 1987]
  - But *really* large constant
- Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

- Caveat. Theoretical improvements to Strassen are progressively less practical.

Mar 15, 2019 CSCI211 - Sprenkle 16

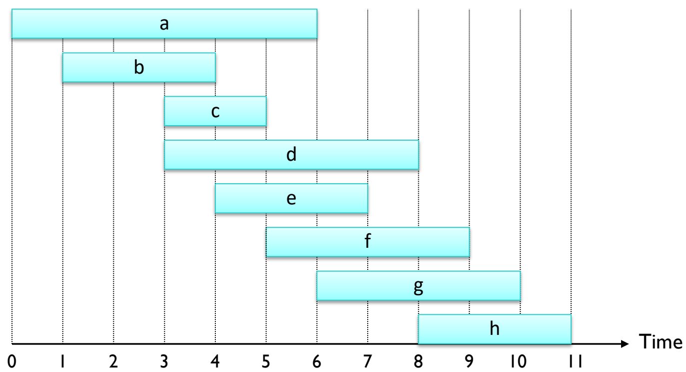
Algorithmic Paradigms

- Greedy. Build up a solution incrementally, myopically optimizing some local criterion
- Divide-and-conquer. Break up a problem into sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form solution to original problem
- **Dynamic programming.** Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems

WEIGHTED INTERVAL SCHEDULING

Weighted Interval Scheduling

- Job j starts at s_j, finishes at f_j, and **has weight or value v_j**
- Two jobs are compatible if they don't overlap
- **Goal**: find maximum weight subset of mutually compatible jobs



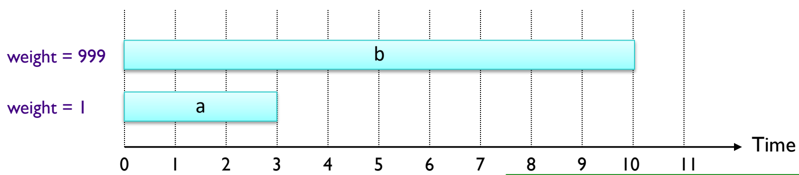
Unweighted Interval Scheduling Review

- Recall. Greedy algorithm works if all weights are 1 (or equivalent).
  - Consider jobs in ascending order of finish time
  - Add job to subset if it is compatible with previously chosen jobs

What happens to Greedy algorithm if we add weights to the problem?

Limitation of Greedy Algorithm

- Recall. Greedy algorithm works if all weights are 1 (or equivalent).
  - Consider jobs in ascending order of finish time
  - Add job to subset if it is compatible with previously chosen jobs
- Observation. Greedy algorithm can fail spectacularly if arbitrary weights are allowed



Any other greedy approaches?

Limitations of Greedy Algorithms

- Need to consider weight
  - No greedy algorithm works
- Need a more complex algorithm to solve problem

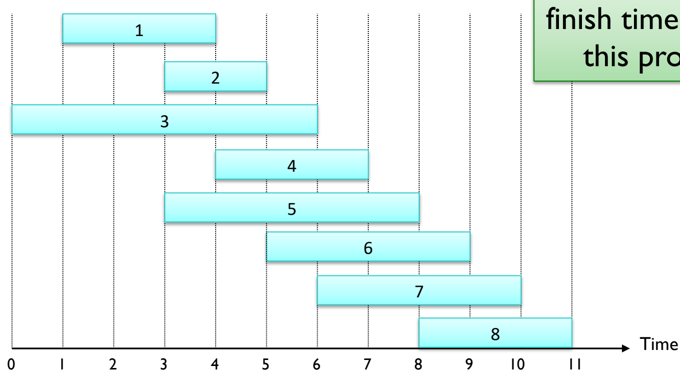
Weighted Interval Scheduling

Notation. Label jobs by finishing time: $f_1 \leq f_2 \leq . . . \leq f_n$

Def. p(j) = largest index *i* < *j* such that job *i* is compatible with *j*

Ex: p(8) = 5, p(7) = 3, p(2) = 0

Why is ordering by finish time useful in this problem?



Dynamic Programming

- Assume we have an optimal solution
- **OPT(j)** = **value** of optimal solution to the *problem* consisting of job requests 1, 2, ..., *j*

What is something *obvious/trivial* we can we say about the optimal solution with respect to job *j*?

Dynamic Programming: Binary Choice

- OPT(j) = **value** of optimal solution to the *problem* consisting of job requests 1, 2, ..., *j*
  - Case 1: OPT selects job *j*
  - Case 2: OPT does not select job *j*

Explore both of these cases…
- What jobs are in OPT? Which are not?

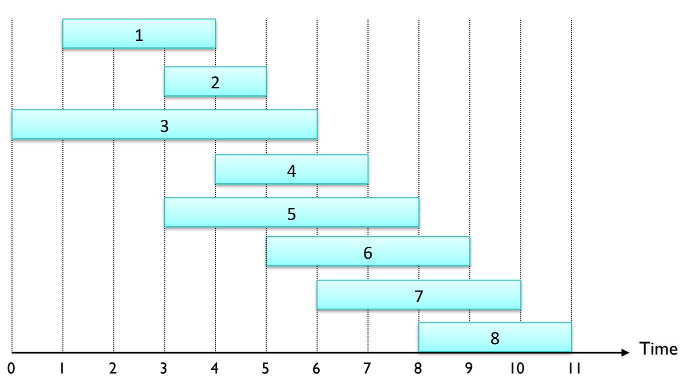
Keep in mind our definition of p

Weighted Interval Scheduling

Notation. Label jobs by finishing time: $f_1 \leq f_2 \leq . . . \leq f_n$

Def. p(j) = largest index *i* < *j* such that job *i* is compatible with *j*

Ex: p(8) = 5, p(7) = 3, p(2) = 0



Dynamic Programming: Binary Choice

- OPT(j) = value of optimal solution to the *problem* consisting of job requests 1, 2, ..., *j*
  - Case 1: OPT selects job *j*
    - can't use incompatible jobs { p(j) + 1, p(j) + 2, ..., j - 1 }
    - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., p(*j*)
  - Case 2: OPT does **not** select job *j*
    - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., *j*-1

optimal substructure

Formulate OPT(j) as a recurrence relation

Dynamic Programming: Binary Choice

- OPT(j) = value of optimal solution to the *problem* consisting of job requests 1, 2, ..., *j*
  - Case 1: OPT selects job *j*
    - can't use incompatible jobs { p(j) + 1, p(j) + 2, ..., j - 1 }
    - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., p(*j*)
  - Case 2: OPT does **not** select job *j*
    - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., *j*-1

Formulate OPT(j) in terms of smaller subproblems
Which should we choose?

Two options: Opt(j) = v_j + Opt(p(j))
Opt(j) = Opt(j-1)

Dynamic Programming: Binary Choice

- OPT(j) = value of optimal solution to the problem consisting of job requests 1, 2, ..., *j*
  - Case 1: OPT selects job *j*
    - can't use incompatible jobs { p(j) + 1, p(j) + 2, ..., j - 1 }
    - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., p(*j*)
  - Case 2: OPT does **not** select job *j*
    - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., *j*-1

$$Opt(j) = \begin{cases} 0 & j=0 \\ \max\{ v_j + Opt(p(j)), Opt(j-1) \} & \text{Otherwise} \end{cases}$$

Basecase

Choose the "better" of the two solutions

Weighted Interval Scheduling: Recursive Algorithm

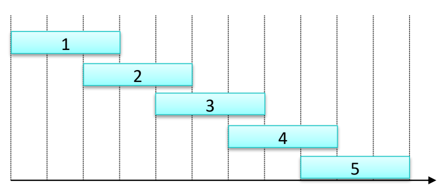
```
Input: n jobs (associated start time sj, finish time fj, and value vj)

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn

Compute p(1), p(2), …, p(n)

Compute-Opt(j):
   if j = 0
      return 0
   else
      return max(vj + Compute-Opt(p(j)), Compute-Opt(j-1))
```

Closest compatible job

Picks j

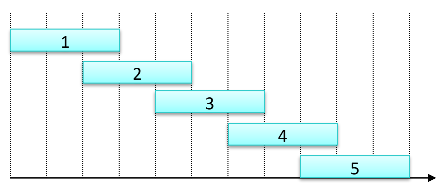
Doesn't pick j

What is the runtime?
(Trace for *n* = 5)

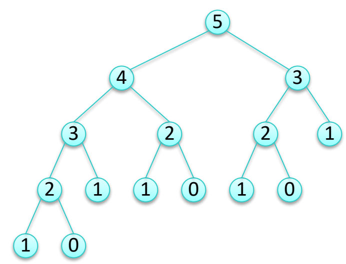


Weighted Interval Scheduling: Brute Force

- Observation. Redundant sub-problems ⇒ exponential algorithms
- Ex. Number of recursive calls for family of "layered" instances grows like Fibonacci sequence.



p(1) = 0, p(j) = j-2



Weighted Interval Scheduling: Memoization

- Store results of each sub-problem in a cache; lookup as needed.

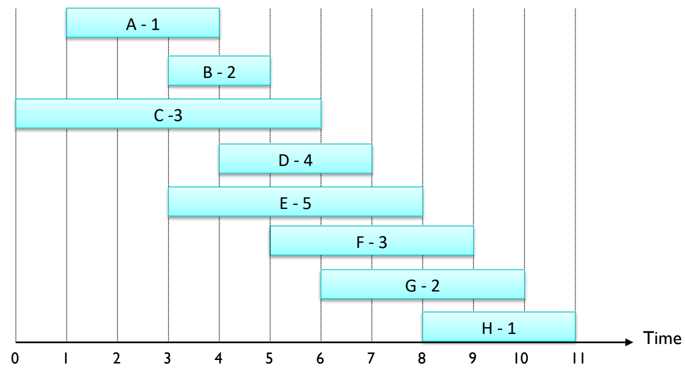
```
Input: n jobs (associated start time sj, finish time fj, and value vj)

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn
Compute p(1), p(2), …, p(n)

for j = 1 to n
   M[j] = empty
M[0] = 0

M-Compute-Opt(j):
   if M[j] is empty:
      M[j] = max(vj + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]

M-Compute-Opt(n)
```

global array (M[j] = empty)

Call function with initial input (M-Compute-Opt(n))

Example

- Jobs labeled as name – weight



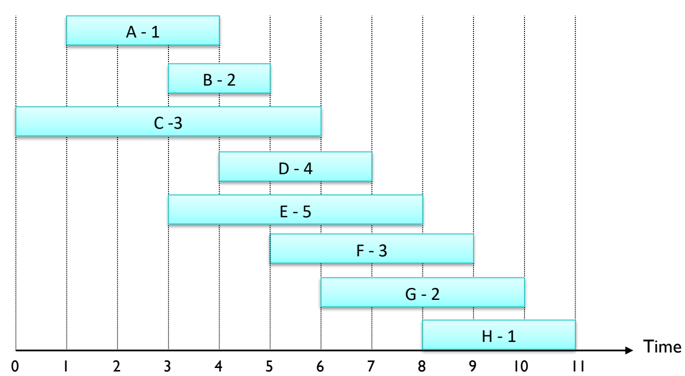
M

0	A	B	C	D	E	F	G	H
0								

Example

What is the value of p for each job?

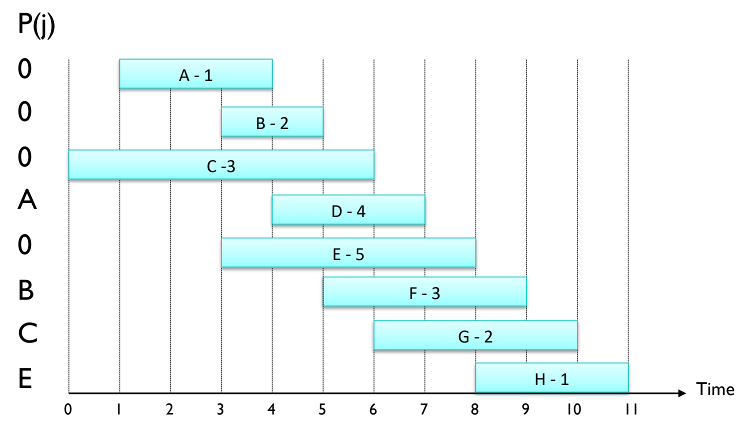
- Jobs labeled as name – weight



M

0	A	B	C	D	E	F	G	H
0								

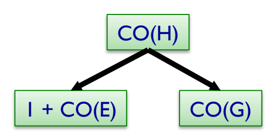
Example

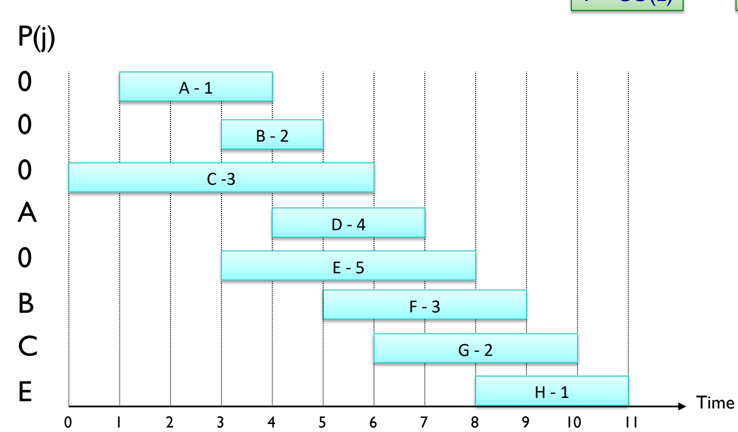


M	0	A	B	C	D	E	F	G	H
	0								

Mar 15, 2019 CSCI211 - Sprenkle 45

Example

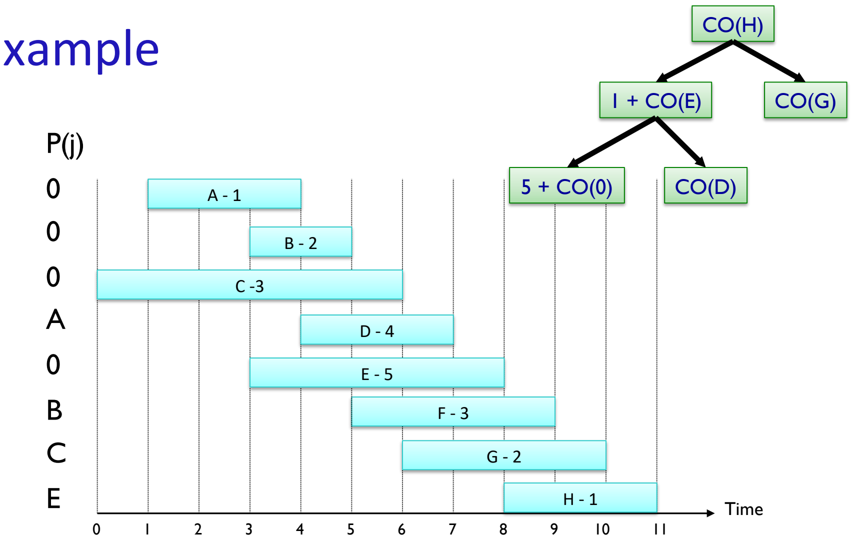




M	0	A	B	C	D	E	F	G	H
	0								

Mar 15, 2019 CSCI211 - Sprenkle 46

Example

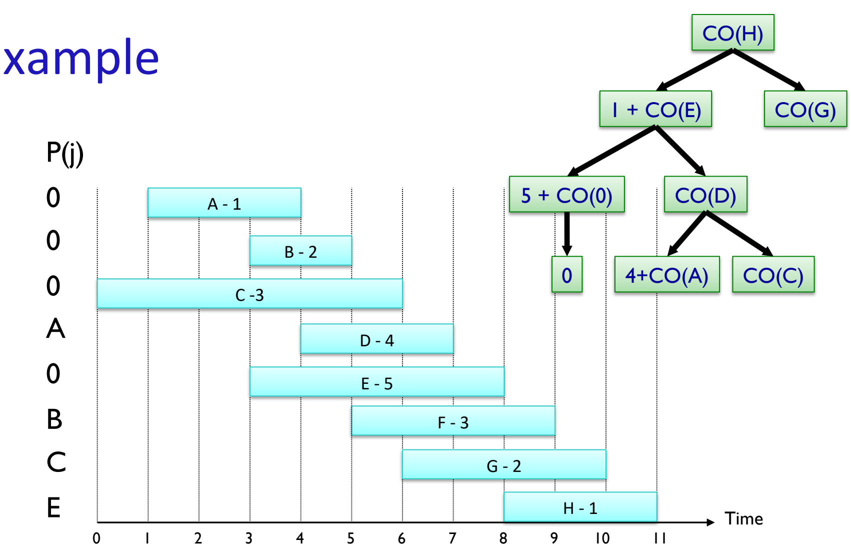


M

0	A	B	C	D	E	F	G	H
0								

Mar 15, 2019 CSCI211 - Sprenkle 47

Example

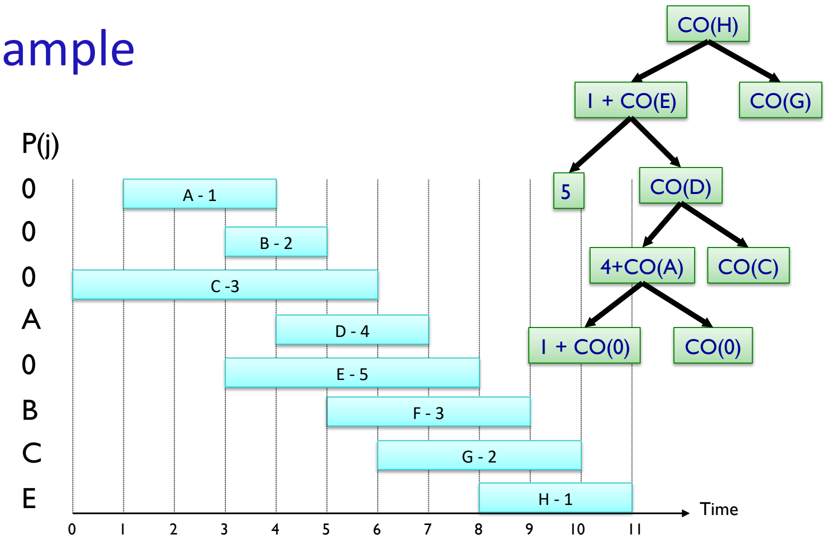


M

0	A	B	C	D	E	F	G	H
0								

Mar 15, 2019 CSCI211 - Sprenkle 48

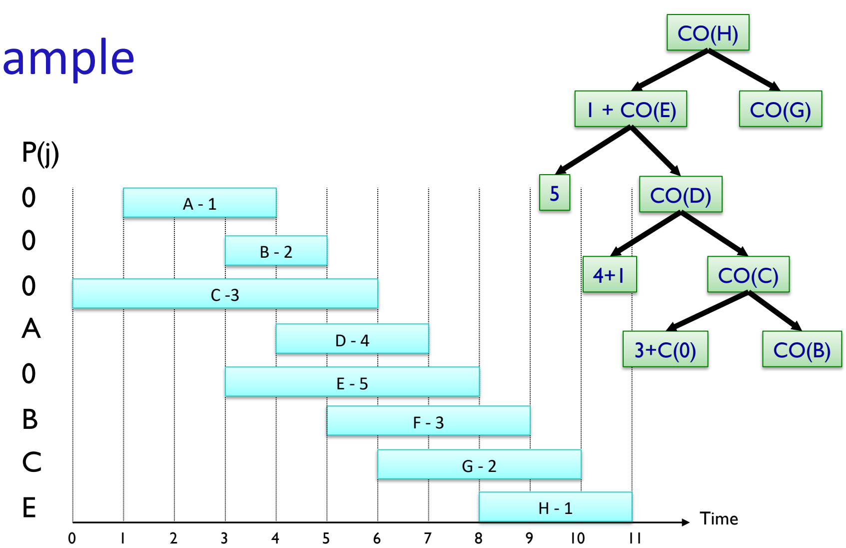
Example



M

0	A	B	C	D	E	F	G	H
0	1							

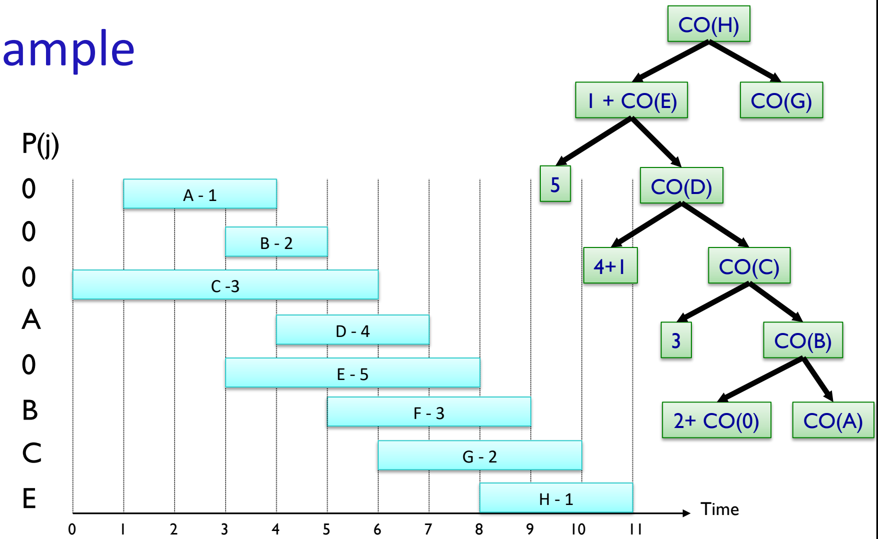
Example



M

0	A	B	C	D	E	F	G	H
0	1							

Example

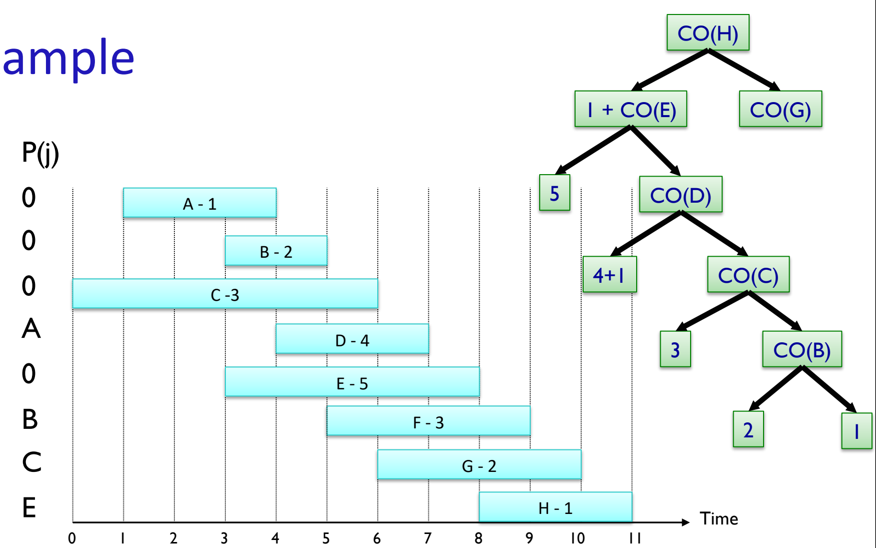


M

0	A	B	C	D	E	F	G	H
0	1							

Mar 15, 2019 L CSCI211 - Sprenkle 51

Example

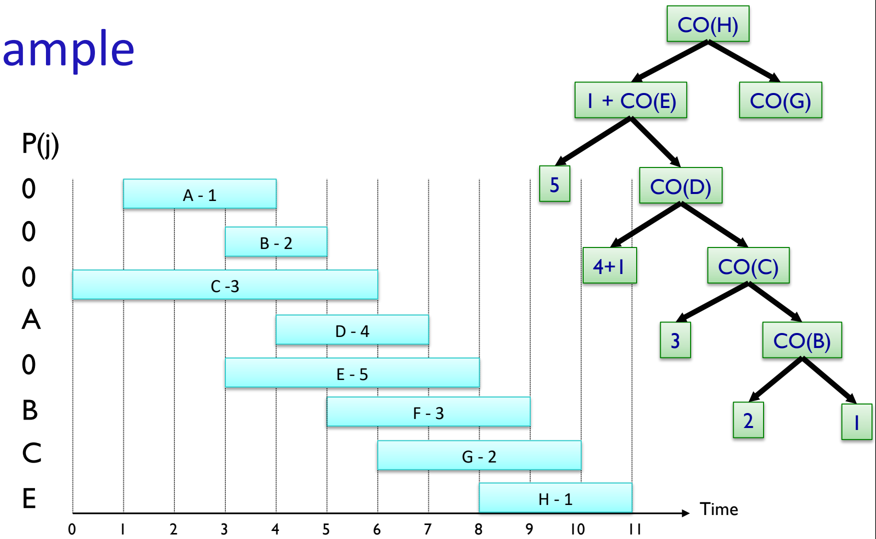


M

0	A	B	C	D	E	F	G	H
0	1							

Mar 15, 2019 L CSCI211 - Sprenkle 52

Example

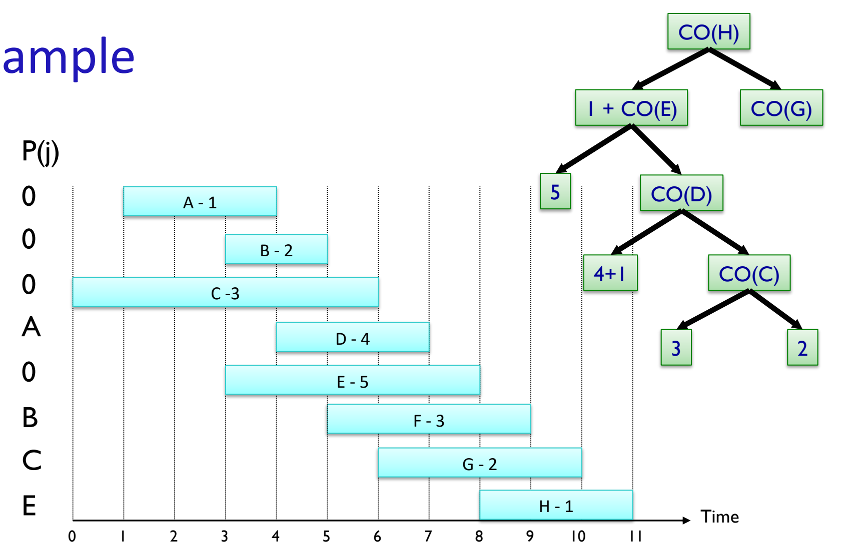


M

0	A	B	C	D	E	F	G	H
0	1	2						

L L

Example

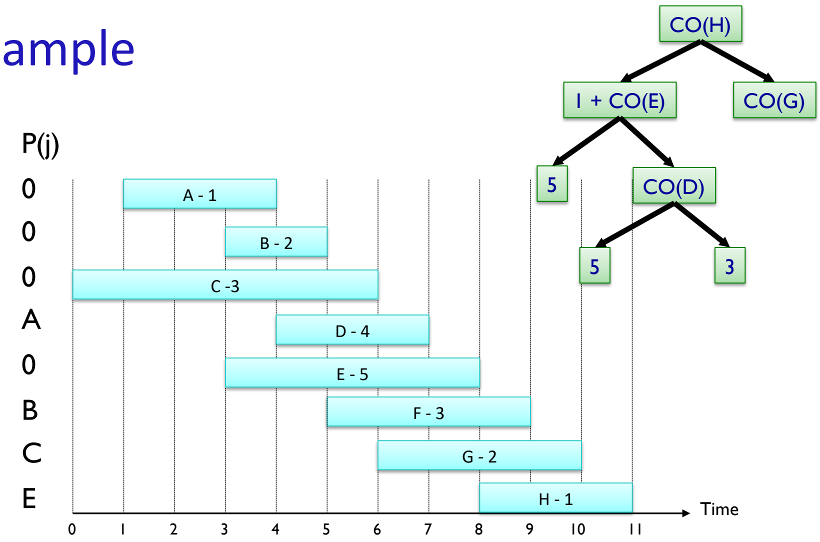


M

0	A	B	C	D	E	F	G	H
0	1	2	3					

L L L

Example

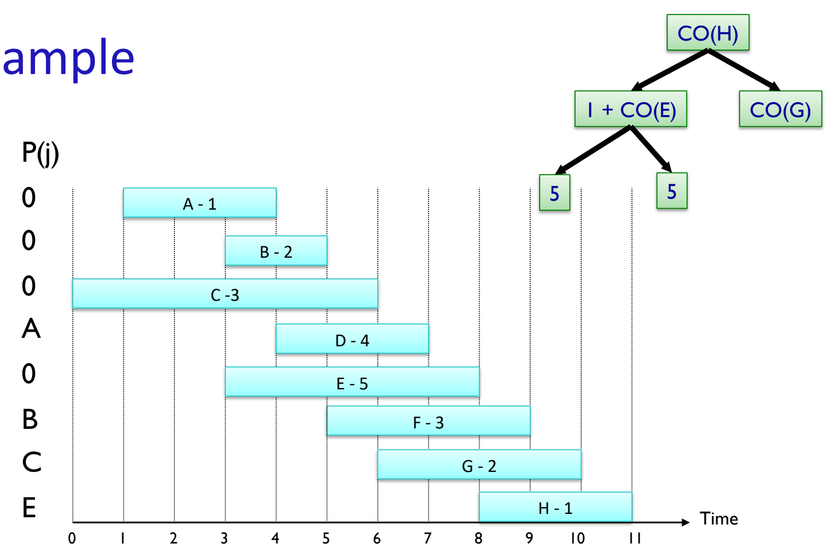


M

0	A	B	C	D	E	F	G	H
0	1	2	3	5				

Mar 15, 2019 L L L CSCI L - Sprenkle 55

Example

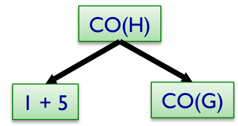


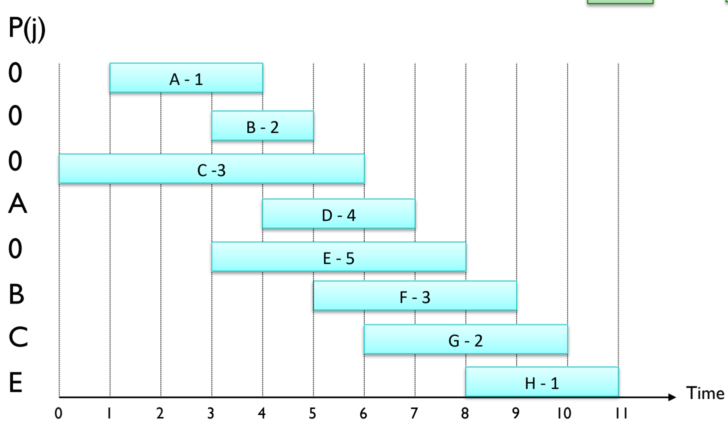
M

0	A	B	C	D	E	F	G	H
0	1	2	3	5	5			

Mar 15, 2019 L L L CSCI L - Sp L/R 56

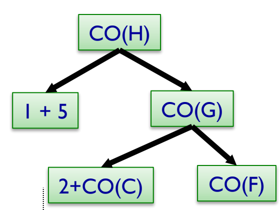
Example

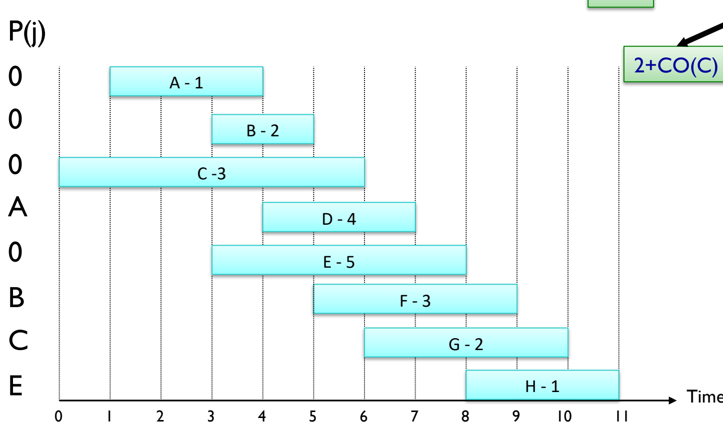




M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5			

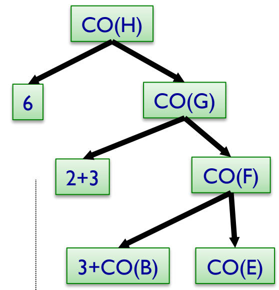
Example

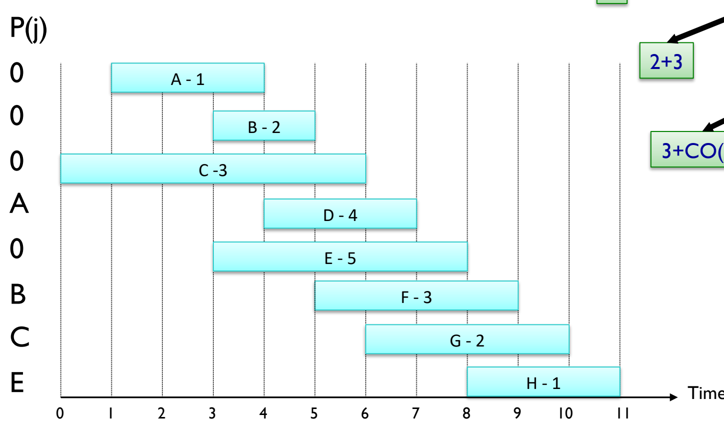




M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5			

Example



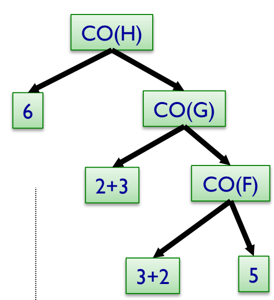


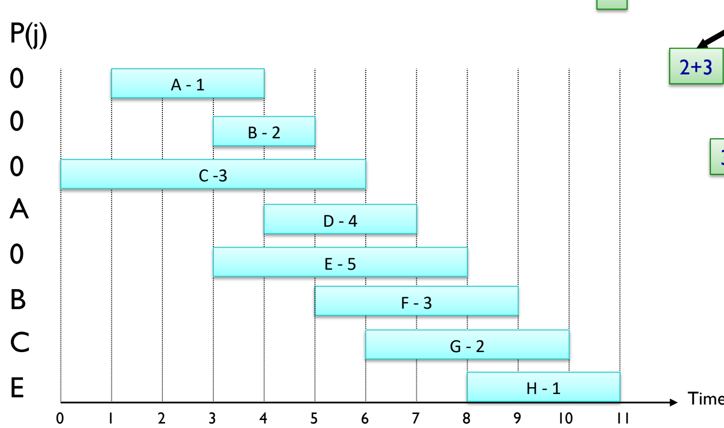
M

0	A	B	C	D	E	F	G	H
0	**1**	**2**	**3**	**5**	**5**			

L L L L L/R

Example



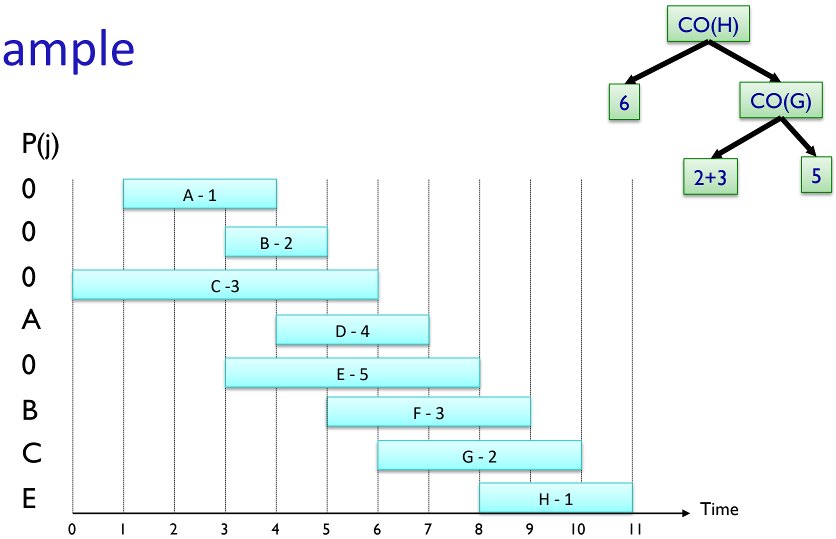


M

0	A	B	C	D	E	F	G	H
0	**1**	**2**	**3**	**5**	**5**	**5**		

L L L L L/R L/R

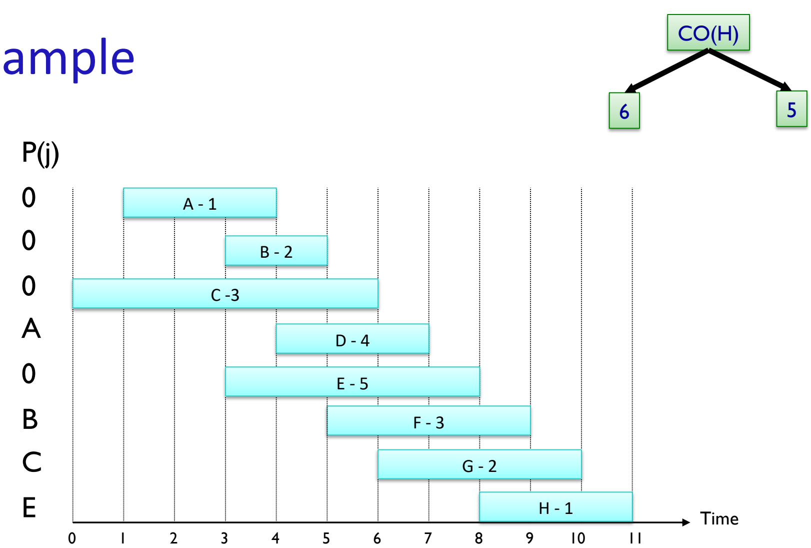
Example



M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5	5	5	

L L L L L/R L/R L/R

Example



M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5	5	5	6

L L L L L/R L/R L/R L

Exam

- Focused on greedy and divide and conquer
- Rules
  - Open brain notes, textbook, wiki, solutions on Sakai, my lecture notes
  - Limited me
  - Closed everything else
- Adjustments
  - No class on Monday – additional office hours during that time
  - No wiki for next week
    - May want to review D&C chapters not in the wiki
  - Office hours:
    - Monday: 9:45 – 10:45 (class) – noon, 2:35 – 5 p.m.
    - Wednesday: 2:35-5 p.m.
    - Thursday: 2:35 p.m.- 5 p.m.
    - Sign up in Box Note