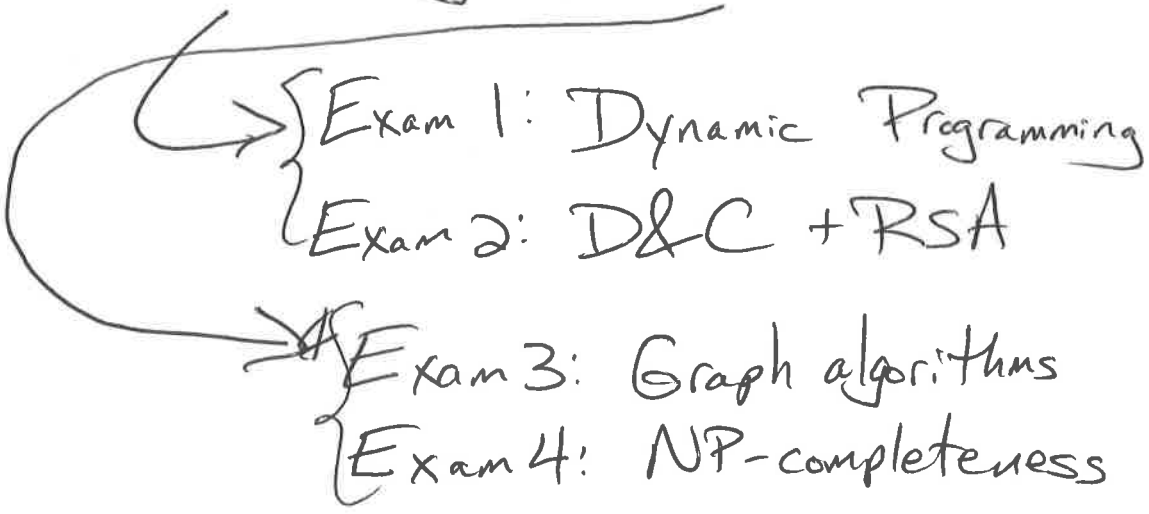
3510 overview:

Course webpage: www.cc.gatech.edu/~vigoda/3510

instructors: Eric Vigoda & Prasad Tetali



No laptops, no phones used in class (distracting to others)

Grading schemes:

Best of 2:

1. HW = 5%, best 3 of 4 midterms = 60%
   & final = 35%
2. HW = 5%, all 4 midterms = 95%
   No final

TA's will hold recitation sessions in evenings 5-7pm?
details will be announced over email.

Login to Piazza through T-Square.
- announcements through Piazza or T-Square.
- HW's submitted through Gradescope
  we will create an account for you using your GTid so it syncs with T-Square.
- No late HW's.
- HW's Due on Mondays before 3pm.

Textbook-
Algorithms by Dasgupta, Papadimitriou & Vazirani
(online version may have diff. numbering ± 1)

HW's worth little but very useful for exams.
OK to have study groups.
But cite collaborators & references
& write up on own (like in exam).

Review: $O(\ )$ notation.

For functions $f(n)$ & $g(n)$,

$f(n) = O(g(n))$ if there is a constant $c > 0$ where $f(n) \leq cg(n)$.

Examples:

a) $f(n) = 3n^2 + 10n - 5n^{2.5} + 1.7n^3$

$f(n) = O(n^3)$ and $f(n) = O(n^5)$ but want best possible.

b) $f(n) = 5(\log n)^{10} + 7\sqrt{n}$

$f(n) = O(\sqrt{n})$

c) $f(n) = n^2 \log n + 1000n^2 + 10^{10} n^{1.5}$

$= O(n^2 \log n)$

Review: Manipulating log's

$\log n$: base 2 if don't specify base $\Rightarrow 2^{\log n} = n$

$\ln n = \log_e n$

$$\log_{10} n = O(\log n)$$
$$\ln n = O(\log n)$$

Exercise: $f(n) = n^2$, $g(n) = 2^{4 \log n}$

a) $f(n) = O(g(n))$, b) $g(n) = O(f(n))$, c) both a & b hold.

rewrite $g(n) = 2^{4 \log n} = \left(2^{\log n}\right)^4 = n^4$

So $f(n) = O(g(n))$ but $g(n) \neq O(f(n))$

Dynamic Programming (DP): Toy example

Computing Fibonacci numbers,

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

defined by:

$F_0 = 0$, $F_1 = 1$

and for $n > 1$,

$F_n = F_{n-1} + F_{n-2}$

Natural recursive algorithm:

Fib1(n):

```
if n=0, return(0)
if n=1, return(1)
return (Fib1(n-1)+Fib1(n-2))
```

What's running time:

Look at running time as function of input size $n = n^{th}$ Fib. #

other examples:

n = # of numbers to sort

n = # of vertices in input graph

n = # of bits in input numbers to multiply.

Look at running time in $O(\ )$
so ignoring constant factors
& machine independent analysis

Model of computation:

- $O(1)$ time to add/subtract/multiply/divide
  & any basic arithmetic operation
- unlimited memory
  & $O(1)$ time read/write unit of memory.

Analyzing Fib1(n)

Let $T(n)$ = # of steps for computing n^{th} Fibonacci #

$T(0)=O(1)$ & $T(1)=O(1)$

for $n>1$, $T(n)=T(n-1)+T(n-2)+O(1)$

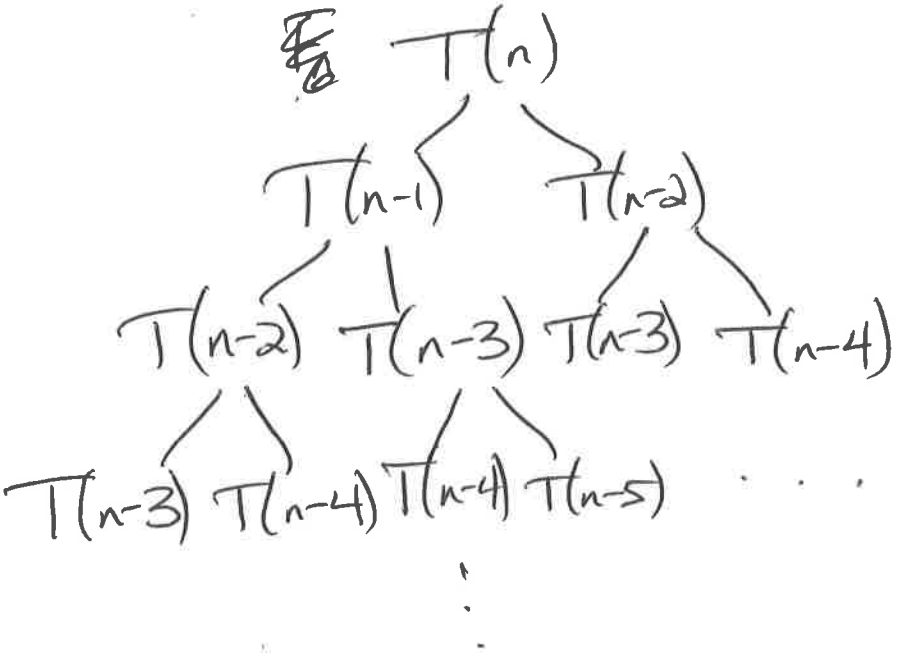
But $T(n) \geq F(n) = n^{th}$ Fibonacci number

$T(n-1)+T(n-2)+c$ $\quad$ $F(n-1)+F(n-2)$

$F_n \approx \frac{\phi^n}{\sqrt{5}}$ where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ is the golden ration

So exponential-time algorithm.

Why is Fib1( ) so slow?



Recomputing small subproblems many times.

Better approach:

Work bottom-up.

Start with smallest F(0), F(1)

& go to larger.

Only solve each subproblem once.

Fib2(n):

```
if n=0, return(0)
if n=1, return(1)
create an array F[0...n]
F[0]=0, F[1]=1
for i=2→n
    F[i] = F[i-1] + F[i-2]
return(F[n])
```

Running time:

$O(1)$ per i & $O(n)$ sized loop over i

$\Rightarrow O(n)$ total time.

Non-trivial example: Longest increasing subsequence (LIS)

input: n numbers $a_1, a_2, \ldots, a_n$

example: 5, 2, 8, 6, 3, 6, 9, 7

a subsequence is a subset in order

e.g., ~~a_2~~ ~~indices~~

using indices 2, 4, 5, 7 so a_2, a_4, a_5, a_7

gives: 2, 6, 3, 9

So a subsequence is a subset $a_{i_1}, a_{i_2}, \ldots, a_{i_\ell}$

where ~~a_2~~ $1 \le i_1 < i_2 < \cdots < i_\ell \le n$

(increasing indices)

Subsequence is increasing if:

$$a_{i_1} < a_{i_2} < \cdots < a_{i_\ell}$$

for example: $a_2, a_5, a_6, a_7 = 2, 3, 6, 9$

is an increasing subsequence.

LIS problem:

Given $a_1, \ldots, a_n$

find an increasing subsequence of max length

For now, just output the length not the actual subsequence.

Input: $a_1, \ldots, a_n$

Output: length of LIS

First step in DP design,

Define the subproblem in words

Natural idea:

let $S(j)$ = length of LIS in $a_1, \ldots, a_j$

Goal: compute $S(n)$.

2nd step: Write recurrence for $S(j)$ in terms of $S(1), S(2), \ldots, S(j-1)$.

Idea: $S(j)$ is max of $S(j-1)$ and $1 +$ best of $S(j-1)$ ending below a_j

But how to get → from $S(1), \ldots, S(j-1)$?

Example: $A = 5, 2, 8, 6, 3, 6, 9, 7$

$S = 1, 1, 2, 2, 2, ?$

$S[6] = 3$ but how do we know whether or not 6 can get added to optimal for $S(j-1)$?

Need to know longest in $a_1, \ldots, a_j$ for each possible ending number.

What are possible ending numbers?

$a_1, \ldots, a_j$

So look at LIS ending at a_j

Redefine subproblem to add extra condition
— forcing ending number.

Let $L(j)$ = length of LIS in $a_1, \ldots, a_j$ which ends at a_j & includes a_j

Goal: Compute $\max_j L(j)$.

Recurrence: (for a_j)

$$L(j) = 1 + \max_i \{ L(i) : i < j, a_i < a_j \}$$

LIS(A)

```
for j=1→n
    L(j)=1                          ← prev(j)=NULL
    for i=1→j-1
        if L(i)+1 > L(j) & a_i < a_j
            then L(j)=L(i)+1        ← prev(j)=i
Let max=1
for i=1→n
    if L(i)>L(max) then max=i
Return(L(max))
```

Running time:

$O(1)$ Per i

$O(n)$ sized loop over i } $O(1) \times O(n) \times O(n)$

$O(n)$ sized loop over j } $= O(n^2)$

$\Rightarrow O(n^2)$ total time

How to find the actual subsequence of max length?

Keep track of the penultimate index i that gives the max for $L(j)$.

Then backtrack.

Earlier example:

$A =$	5	2	8	6	3	6	9	7	2	4
$L =$	1	1	2	2	2	3	4	4	1	3
Prev =	NULL	NULL	1	1	2	5	6	6	NULL	5

To reconstruct $L(8)$ follow path $a_8 \to a_6 \to a_5 \to a_2$

$7 \to 6 \to 3 \to 2$

and reversing it's: 2, 3, 6, 7.