Midterm Review

Midterm Topics

- Fundamentals:
  - Asymptotics
  - Recurrences
- Divide and Conquer
  - Binary Search, MergeSort, Karatsuba's
- Dynamic Programming
  - WIS, Knapsack, LCS, LIS, Edit Distance

Topics: Asymptotics

Notation	… means …	Think…	E.g.
f(n)=O(n)	$\exists c > 0, n_0 > 0, \forall n \geq n_0:$ $0 \leq f(n) \leq cg(n)$	At most "≤"	$100n^2 = O(n^3)$
f(n)=Ω(g(n))	$\exists c > 0, n_0 > 0, \forall n \geq n_0:$ $0 \leq cg(n) \leq f(n)$	At least "≥"	$2^n = \Omega(n^{100})$
f(n)=Θ(g(n))	$f(n) = O\big(g(n)\big)$ and $f(n) = \Omega(g(n))$	Equals "="	$\log(n!) = \Theta(n \log n)$
f(n)=o(g(n))	$\mathrm{Lim}_{n\to\infty} \frac{f(n)}{g(n)} = 0$	Less than "<"	$n^2 = o(2^n)$
f(n)=ω(g(n))	$\mathrm{Lim}_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	Greater than ">"	$n^2 = \omega(\log n)$

Topics: Asymptotics

- **Constant factors can be ignored**
  - $\forall C > 0 \quad Cn = O(n)$
- **Smaller exponents are Big-Oh of larger exponents**
  - $\forall a > b \quad n^b = O(n^a)$
- **Any logarithm is Big-Oh of any polynomial**
  - $\forall a, \varepsilon > 0 \quad \log_2^a n = O(n^\varepsilon)$

$\log^{100} n = O(n^{0.001})$

- **Any polynomial is Big-Oh of any exponential**
  - $\forall\ a > 0, b > 1 \quad n^a = O(b^n)$
- **Lower order terms can be dropped**
  - $n^2 + n^{3/2} + n = O(n^2)$

Practice Question: Asymptotics

- Put these functions in order so that $f_i = O(f_{i+1})$

f_4 • $n^{\log_2 7}$ $\rightarrow \left(2^{\log_2 n}\right)^{\log_2 7} = 2^{(\log n) \times (\log_2 7)}$ $(2.\ )^{\log_2 n} = 2$

f_5 • $8^{\log_2 n}$ $2^{3 \log_2 n}$

f_1 • $2^{3 \log_2 \log_2 n}$

$\frac{n}{2} \times \frac{n}{2} \le \sum_{i=1}^{n} i \le n^2$

f_6 • $2^{(\log_2 n)^2}$

f_2 • $\sum_{i=1}^{n} i$ $= \frac{n(n+1)}{2} = \Theta(n^2) = \left(2^{\log n}\right)^2 = 2^{2 \log_2 n}$

f_3 • $n^2 \log_2 n$

$n^2 \log n = 2^{2 \log_2 n} \times 2^{\log \log n} = 2^{2 \log_2 n + \log \log n}$

Practice Question: Asymptotics

- Suppose $f_1 = O(g)$ and $f_2 = O(g)$.
  Prove that $f_1 + f_2 = O(g)$.

The fact that $f_1 = O(g)$ implies there are $c, n_0 > 0$ s.t.

$f_1(n) \le c\,g(n)$ for all $n \ge n_0$. For the same reason

there are c', n_0' s.t. $f_2(n) \le c'\,g(n)$ for all $n \ge n_0'$.

Let $n_0'' = \max\{n_0, n_0'\}$ and let $c'' = c + c'$. Then for

all $n \ge n_0''$, $f_1 + f_2 \le c\,g(n) + c'g(n) = c''g(n)$.

Practice Question: Asymptotics

- Suppose $f = O(g)$ and $h = O(g)$.
  Prove or disprove that $f = \Theta(h)$.

False. Take $f(n) = \lg n$, $h(n) = n$, $g(n) = n^2$.

We have $f(n) = O(g(n))$ and $h(n) = O(g(n))$ yet

$f(n)$ is not $\Theta(n)$ because $f(n) = o(h(n))$.

Practice Question: Asymptotics

$g \geq h$

$f \geq g$, $\cancel{g = h}$

$h \overset{?}{\leq} f$

- Suppose $f = \Omega(g)$ and g $= \Theta(h)$.
  Prove or disprove that h$= O(f)$.

The fact that $f_1 = \Omega(g)$ implies there are $c, n_0 > 0$ s.t.
$f(n) \geq c\,g(n)$ for all $n \geq n_0$. The fact that $g = \Theta(h)$
implies $g = \Omega(h)$ which implies there are $c', n_0' > 0$ s.t.
$g(n) \geq c' h(n)$ for $n \geq n_0'$. Therefore, for all $n \geq \max\{n_0, n_0'\}$

$f(n) \geq c\,g(n) \geq c \cdot c' h(n)$ So suffices to take $c'' = c \cdot c'$!

Practice Question: Asymptotics

Problem 1 (25 Points). For each pair of functions, f, g determine whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. Provide a proof for the correct relationship.

1. $f(n) = n^3 - 2n, g(n) = 100n^2$
2. $f(n) = 2^{\log_3 n}, g(n) = 3^{\log_2 n}$
3. $f(n) = (n!)^{1/\log n}, g(n) = 2^{\sqrt{n}}$

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$f(n) = (n!)^{1/\lg n} = \left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)^{1/\lg n} = 2^{\lg\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)/\lg n}$$

$$= 2^{\Theta(n \lg n)/\lg n} = 2^{\Theta(n)}$$

$$\lg\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right) \geq \lg\left(\left(\frac{n}{e}\right)^n\right) = n \lg\left(\frac{n}{e}\right) = \Omega(n \lg n)$$

1. We claim $f(n) = \Omega(g(n))$. To prove this, we show that there exist positive constants c and n_0 such that for all $n \geq n_0$, the inequality $n^3 - 2n \geq c \cdot 100n^2$ holds.

   First, let's simplify the left side of the inequality:

   $$n^3 - 2n = n(n^2 - 2)$$

   Now, we need to find c and n_0 such that:

   $$n(n^2 - 2) \geq c \cdot 100n^2$$

   We can simplify further:

   $$n^2 - 2 \geq c \cdot 100n$$

   Now, we need to find a value for c and n_0 that makes this inequality true for all $n \geq n_0$. Let's choose $c = \frac{1}{100}$. Then, our inequality becomes:

   $$n^2 - 2 \geq \frac{1}{100} \cdot 100n$$

   Simplifying further, we get

   $$n^2 - 2 \geq n$$

   which holds for all $n \geq 2$.

2. Rewriting, we get $f(n) = n^{\log_3 2}, g(n) = n^{\log_2 3}$. For $c = 1, n_0 = 1$, we have that $f(n) \leq c \cdot g(n)$ since $n^{\log_3 2} \leq n^{\log_2 3}$. Therefore, $f(n) = O(g(n))$.

3. By Stirling's approximation, we have $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$. Therefore, $f(n) = (n!)^{1/\log n} = 2^{\Theta(n \log n / \log n)} = 2^{\Theta(n)}$. Therefore, we have $f(n) = \Omega(g(n))$.

Topics: Recurrences

- Recurrences
  - Representing running time by a recurrence
  - Solving common recurrences
  - Master Theorem

Practice Question: Recurrences

```
F(n):
  For i = 1,…,n²: Print "Hi"
  For i = 1,…,2: F(2n/3)
```

- Write a recurrence for the running time of this algorithm. Write the asymptotic running time given by the recurrence.

$$T(n) = 2T(2n/3) + n^2$$

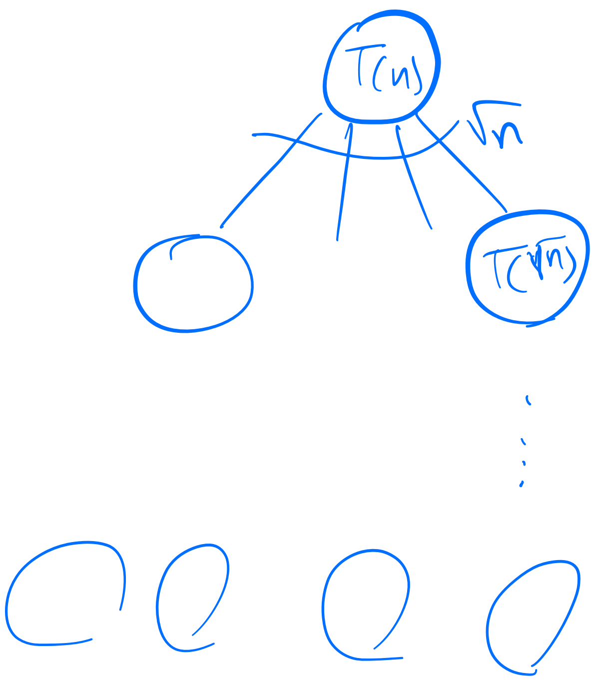
Practice Question: Recurrences

```
F(n):
  For i = 1,…,n²: Print "Hi"
  For i = 1,…,2: F(2n/3)
```

Problem 2 (25 Points). Solve the recurrence $T(n) = 2T(\frac{2}{3}n) + n^2$ in two ways. First way: by adding up the work done in each layer of the recursion tree. Second way: using the Master Theorem.

Topics: Recurrences

- Consider the recurrence $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$ with $T(1) = 1$. Solve using a recursion tree.



n

$\sqrt{n} + \cdots + \sqrt{n} = n$

$n \to n^{1/2} \to (n^{1/2})^{1/2} \to \cdots$

$n^{1/2^i}$ for $i = \lg\lg n$

we have

$n^{\frac{1}{\lg n}} = 2^{\lg n / \lg n} = 2$

$T(n) = L \cdot n$

$O(\lg\lg n)$

Topics: Divide-and-Conquer

- Divide-and-Conquer
  - Writing pseudocode
  - Proving correctness by induction
  - Analyzing running time via recurrences

- Examples we've studied:
  - MergeSort, Binary Search, Karatsuba's

Practice Question

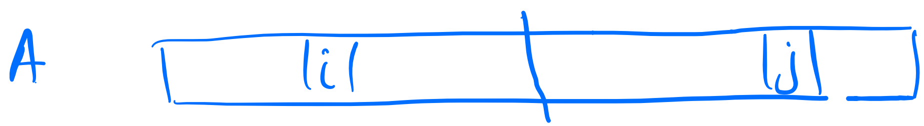
- Consider the following sorting algorithm

```
A[1:n] is a global array
SillySort(1,n):
   if (n <= 2): put A in order
   else:
      SillySort(1,2n/3)
      SillySort(n/3,n)
      SillySort(1,2n/3)
```

- Prove that it is correct
- Analyze its running time

Topics: Divide-and-Conquer

Problem 3 (25 Points). Give a $\Theta(n \log n)$ time *divide and conquer* algorithm that given an array A of n integers, finds two indices $i < j$ such that $A[j] - A[i]$ is maximized. Analyze and show that your algorithm runs in the required $\Theta(n \log n)$ time.



Three cases: (1) both i, j in the first half

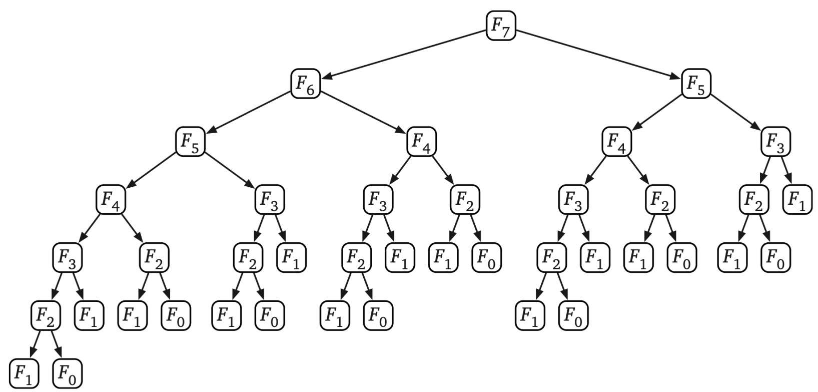
(2) " " " second "

Call the same function recursively

(3) i is in the first half and j is in the second half

Topics: Dynamic Programming

- *Dynamic programming* refers to a strategy of solving (and remembering) solutions to subproblems that can be used to compute larger solutions




```
RecFibo(n):
  if n = 0
      return 0
  else if n = 1
      return 1
  else
      return RecFibo(n − 1) + RecFibo(n − 2)
```

```
MemFibo(n):
  if n = 0
      return 0
  else if n = 1
      return 1
  else
      if F[n] is undefined
          F[n] ← MemFibo(n − 1) + MemFibo(n − 2)
      return F[n]
```

```
IterFibo(n):
  F[0] ← 0
  F[1] ← 1
  for i ← 2 to n
      F[i] ← F[i − 1] + F[i − 2]
  return F[n]
```

Topics: Dynamic Programming

- Examples:
  - Fibonacci
  - Weighted Interval scheduling
  - Knapsack
  - Longest Common Subsequence
  - Longest Increasing Subsequence
  - Edit Distance

Topics: Dynamic Programming

- Solution structure
  - Describe the subproblems (English)
  - Describe how the subproblems relate (English)
  - Write pseudocode (top down or bottom up)
  - Analyze time complexity

Topics: Dynamic Programming

Problem 4 (25 Points). The price of a stock on each day is given to you in an array. Assume you have enough money to buy a stock on all of the days. However, you cannot buy if you already have a stock in hand (every buy must be followed by a sell before another buy). To be more precise, on each day you have one of three options. You can either buy exactly one share of the stock, sell exactly one share of the stock, or do nothing. However, you can't buy any additional shares if you already possess one share, and you cannot sell any shares if you don't possess one share. In other words, you can only have exactly 0 or 1 share at any given moment. A transaction is 1 buy followed by 1 sell. You can perform at most K transactions.

As the manager of some traders in your company you want to come up with an algorithm to find out the maximum profit your traders can possibly get by buying/selling the stock on these days.

B S B S

For example, if the given array is [100, 200, 250, 330, 40, 30, 700, 400], and $K = 2$ the maximum profit can be earned by buying on day 1, selling on day 4, again buy on day 6 and selling on day 7. As another example, if the given prices in the array keep decreasing, then no profit can be earned. You only need to output the profit (a number).

$\underset{1 \text{ to } n}{\text{profit}[i]}\underset{1 \text{ to } K}{[j]}$ = max Profit within the first i days using at most j transactions

$$\text{Profit}[i][j] = \max\Big(\overbrace{\text{Profit}[i-1][j]}^{\text{no transaction on day } i},\ \text{Profit}[i-2][j-1] - A[i-1] + A[i],$$
$$\text{Profit}[i-3][j-1] - A[i-2] + A[i]$$
$$\text{Profit}[i-4][j-1] - A[i-3] + A[i]$$
$$\vdots$$
$$\text{Profit}[0][j-1] - A[1] + A[i] \Big)$$

Base Case: $\text{Profit}[0][j] = 0$

$\text{Profit}[i][0] = 0$

n time to update, kn cells $\Rightarrow O(kn^2)$ time.

for $i = 1$ to n:

$\quad$ $Profit[i][0] = 0$

for $j = 1$ to K:

$\quad$ $Profit[0][j] = 0$

Base Cases

for $i = 1$ to n:

$\quad$ for $j = 1$ to K:

$O(i)$ time $\rightarrow$ $Profit[i][j] = \max\left(Profit[i-1][j],\ \max_{1 \le i' < i} \left(profit[i'-1][j-1] - A[i'] + A[i] \right) \right)$

return $profit[n][K]$.

overall takes $O(n \times K \times n) = O(Kn^2)$.