CSE3500

HW1 due on 10.4.16 @ 3:30 PM

(1)

Result = 0

for i = 1 to n do

  for j = 1 to i do

    Result ++; ← 1

  i

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$

(2c)

DEFN.

We say $f(n) = o(g(n))$

if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.

Example: $f(n) = n$; $g(n) = n \log n$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{1}{\log n} = 0.$$

$\Rightarrow n = o(n \log n)$

$$2^{\sqrt{\log n}}$$

$$\sqrt{n}$$

$$=$$

$$n^{1/2}$$

$$= \left(2^{\log n}\right)^{1/2}$$

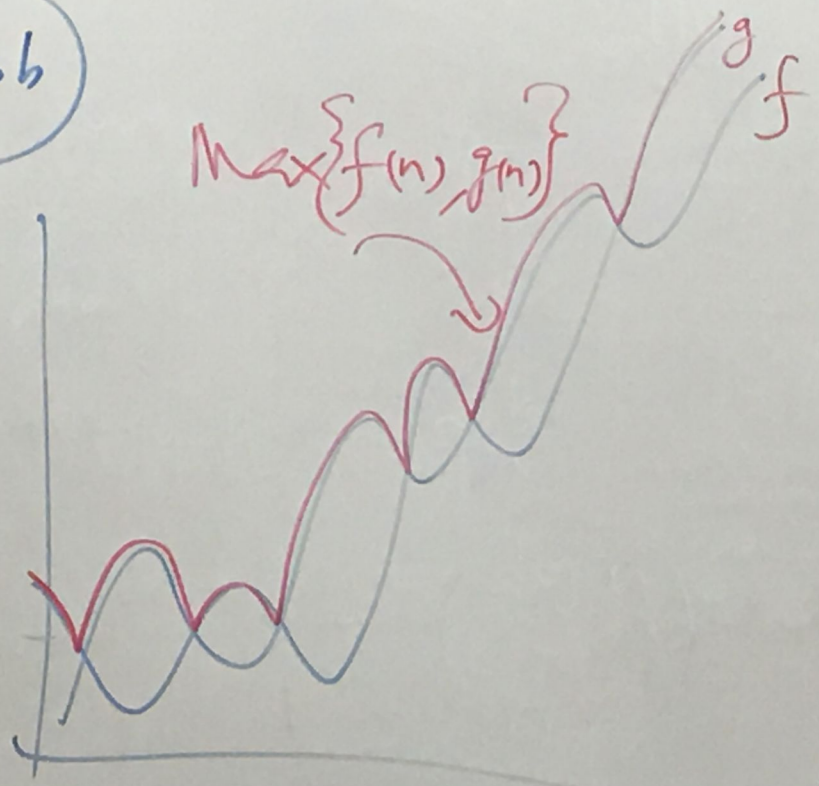
$$= 2^{\frac{1}{2}\log n}$$

$\theta(1), (\log n)^{\epsilon}, \log n, (\log n)^{1+\alpha}, (\log n)^{\frac{1}{\epsilon}}, 2^{n^{\epsilon}}, n, n^{1+\alpha}, 2^{n}, 2^{n^{1+\alpha}}$

ϵ - CONSTANT > 0 and < 1

α - CONSTANT > 0

3b



FACT: $\text{Max}\{f(n), g(n)\} = O(f(n) + g(n))$.

$\text{Max}\{f(n), g(n)\} \overset{?}{\leq} c(f(n) + g(n))$

TRUE FOR $c = 1$.

$\Rightarrow \text{Max}\{f(n), g(n)\} = O(f(n) + g(n))$.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2 2^{\sqrt{\log n}}$$

$a = 4,\ b = 2;\ f(n) = n^2 2^{\sqrt{\log n}}$

$n^{\log_b a} = n^{\log_2 4} = n^2$

Does there exist an ϵ s.t.

$f(n) = \Omega\left(n^{\log_b a} \cdot n^{\epsilon}\right)$?

$n^2 2^{\sqrt{\log n}} = \Omega\left(n^2 n^{\epsilon}\right)$?

$2^{\sqrt{\log n}} = \Omega(n^{\epsilon})$?

CSE3500

HW1 due on 10.4.16 @ 3:30 PM

EXAM 1 10.18.16 5PM

SOLVING RECURRENCE RELATIONS.

(3) PROOF BY INDUCTION.

1. START with a guess to the solution.
2. Attempt to prove the guess by induction
3. ITERATE

Example:

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

(1) Guess: $T(n) \leq c\, n \log n$

Base case is easy.

INDUCTION:

ASSUME the hypothesis

for all inputs of size $\leq (n-1)$.

We'll prove it for n.

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad \text{——} (1)$$

Applying the Hypothesis on (1) we get:

$$T(n) \leq 2.\left[c\,\frac{n}{2}.\log\left(\frac{n}{2}\right)\right] + n$$

$$= cn(\log n - 1) + n = cn\log n - (c-1)n \quad \text{——} (2)$$

RHS of (2) will be $\leq cn\log n$ when $c \geq 1$.

We pick $c = 1$.

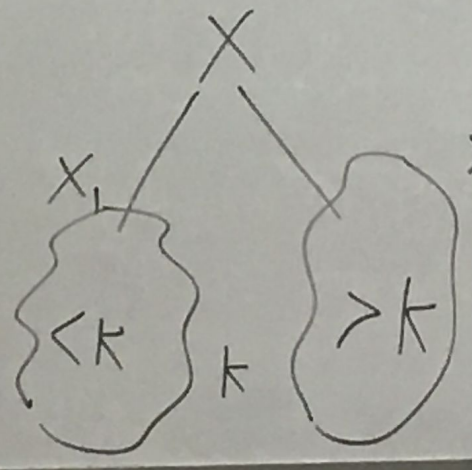
$\Rightarrow T(n) \leq n\log n$. □

Quick Sort.

C.A.R. HOARE
1967

$X = k_1, k_2, \ldots, k_n$.

1. Pick a PIVOT $k \in X$.
2. PARTITION X into $X_1 = \{q \in X : q < k\}$ and $X_2 = \{q \in X : q > k\}$



3. Recursively sort X_1;
   Recursively sort X_2;
4. Output: Sorted X_1, k, Sorted X_2

Let $T(n)$ be the RUN TIME of Quicksort on any input of size n.

$$T(n) = n + T(|X_1|) + T(|X_2|)$$

A WORST CASE: X - SORTED ALREADY.
k - FIRST element of X

$$\begin{aligned} T(n) &= n + T(0) + T(n-1) \\ &= T(n-1) + n \\ &= T(n-2) + (n-1) + n = T(n-3) + (n-2) + (n-1) + n \\ &\vdots \\ &= \sum_{i=1}^{n} i = \Theta(n^2). \end{aligned}$$

A
Best case: $|X_1| \simeq |X_2| \approx \frac{n}{2}$.

$\Rightarrow T(n) = n + 2T\left(\frac{n}{2}\right) = \Theta(n \log n)$.

LEMMA: Average RUN TIME of Quicksort is $O(n \log n)$.

Proof:

Let $\Pi_1, \Pi_2, \ldots, \Pi_i, \ldots, \Pi_j, \ldots, \Pi_n$ be the sorted order of X.

Let $X_{ij} = \begin{cases} 1 & \text{if } \Pi_i \text{ and } \Pi_j \text{ will be COMPARED} \\ 0 & \text{Otherwise.} \end{cases}$

Total # of comparisons $= \sum_{j=i+1}^{n} \sum_{i=1}^{n} X_{ij}$

We want to compute $E\left[\sum_{j=i+1}^{n} \sum_{i=1}^{n} X_{ij}\right]$

$= \sum_{j=i+1}^{n} \sum_{i=1}^{n} E[X_{ij}]$ ——— (1)

Let P_{ij} be the probability that Π_i and Π_j will be Compared.

$E[X_{ij}] = P_{ij} \cdot 1 + (1 - P_{ij}) \cdot 0$

$= P_{ij}$.

$\Rightarrow$ AVERAGE RUN TIME $= \sum_{j=i+1}^{n} \sum_{i=1}^{n} P_{ij}$ ——— (2)

CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 9: September 27, 2016

Hints on Homework 1

- Problem 1: When analyzing nested loops start from the innermost loop and progress outward. As an example, consider:

```
Result = 0;
for i = 1 to n do
    for j = 1 to i do
        Result++;
```

  The instruction in the second **for** loop takes one unit of time per execution. As a result, the second **for** loop takes i time. This in turn means that the run time of the first **for** loop is $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$.

- Problem 2c: Let $f(n)$ and $g(n)$ be two non-negative functions of n. We say $f(n) = o(g(n))$ if $\lim_{n\to\infty} \frac{f(n))}{g(n)} = 0$. As an example, let $f(n) = n^2$ and $g(n) = n^2 \log\log n$. Then $\lim_{n\to\infty} \frac{f(n))}{g(n)} = \lim_{n\to\infty} \frac{1}{\log\log n} = 0$. Thus, in this case $f(n) = o(g(n))$.

- A general hint: When comparing two functions, it might help to express both of them as 2 to the power of some functions and compare the exponents. For example consider the two functions $f_1(n) = \sqrt{n}$ and $f_2(n) = 2^{\sqrt{\log n}}$. We note that $\sqrt{n} = 2^{(1/2)\log n}$. When we compare the exponents of these two functions we realize that $f_1(n)$ is larger than $f_2(n)$.

- Here is a (small) list of functions in increasing order (In this list ϵ is any constant such that $0 < \epsilon < 1$ and α is a constant > 0):

$$\Theta(1), (\log n)^{\epsilon}, \log n, (\log n)^{1+\alpha}, 2^{(\log n)^{\epsilon}}, n^{\epsilon}, n, n^{1+\alpha}, 2^{n^{\epsilon}}, 2^{n}, 2^{n^{1+\alpha}}$$

- Problem 3b: $\max\{f(n), g(n)\}$ is nothing but the pointwise maximum between $f(n)$ and $g(n)$.

- We can easily see that $\max\{f(n), g(n)\} = O(f(n) + g(n))$ since $\max\{f(n), g(n)\} \leq (f(n) + g(n))$.

Master Theorem

- We studied the Master theorem in the last lecture.
- There are recurrence relations that may not be solvable using the Master theorem.
- As an example, consider the recurrence relation: $T(n) = 27T(n/3) + n^3 2^{\sqrt{\log n}}$. Here $a = 27, b = 3$, and $f(n) = n^3 2^{\sqrt{\log n}}$. $n^{\log_b a} = n^3$. It seems like case 3 may be applicable. For case 3 to apply, it should be the case that there exists a constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a} n^\epsilon)$. I.e., $n^3 2^{\sqrt{\log n}} = \Omega(n^3 n^\epsilon)$. In other words, $2^{\sqrt{\log n}} = \Omega(n^\epsilon)$. However, there is no such constant since $2^{\sqrt{\log n}} = o(n^\epsilon)$ for any constant $\epsilon > 0$.

Solving a recurrence relation by induction

- Another technique for solving a recurrence relation uses a guess and a proof by induction.
- Steps involved in this technique are:

  1. Guess a solution to the recurrence relation;
  2. Attempt to prove the guess by induction;
  3. Iterate if there is a need.

- An Example: Consider the recurrence relation: $T(n) = 2T(n/2) + n$. We make the following guess: $T(n) \leq cn \log n$ for some constant c.
- Now a proof by induction is attempted. The base case is easy.
- Induction step: Assume the hypothesis for all inputs of size up to $n - 1$. We'll prove it for n.
- $T(n) = 2T(n/2) + n$. Applying the induction hypothesis to $T(n/2)$ we realize that $T(n) \leq 2c\frac{n}{2} \log\left(\frac{n}{2}\right) + n = cn \log n - (c-1)n$. The RHS will be $\leq cn \log n$ when $c \geq 1$. Thus we infer that $T(n) \leq n \log n$.

Quick sort

- We have revisited the quick sort algorithm and analyzed its worst case and best case run times.
- **Lemma:** The average run time of quick sort is $O(n \log n)$ on any sequence of n elements.

- **Proof:** Let $X = k_1, k_2, \ldots, k_n$ be the input sequence and let $\pi_1, \pi_2, \ldots, \pi_n$ be the sorted order of X.

  Let

  $$X_{ij} = \begin{cases} 1 & \text{if } \pi_i \text{ and } \pi_j \text{ will be compared} \\ 0 & \text{otherwise.} \end{cases}$$

  Total number of comparisons made in the algorithm is $\sum_{j=i+1}^{n} \sum_{i=1}^{n} X_{ij}$. We are interested in computing the expected value of this.

  The average number of comparisons made in the algorithm $A(n)$ is given by

  $$A(n) = E[\sum_{j=i+1}^{n} \sum_{i=1}^{n} X_{ij}] = \sum_{j=i+1}^{n} \sum_{i=1}^{n} E[X_{ij}] \tag{1}$$

  using the fact that $E[X_1 + X_2] = E[X_1] + E[X_2]$.

  Let p_{ij} be the probability that π_i and π_j will be compared in the quick sort algorithm. Then, $E[X_{ij}] = 1 \times p_{ij} + 0 \times (1 - p_{ij}) = p_{ij}$.

  Substituting this in equation 1, we get:

  $$A(n) = \sum_{j=i+1}^{n} \sum_{i=1}^{n} p_{ij}. \tag{2}$$

  The proof will be completed in the next lecture.