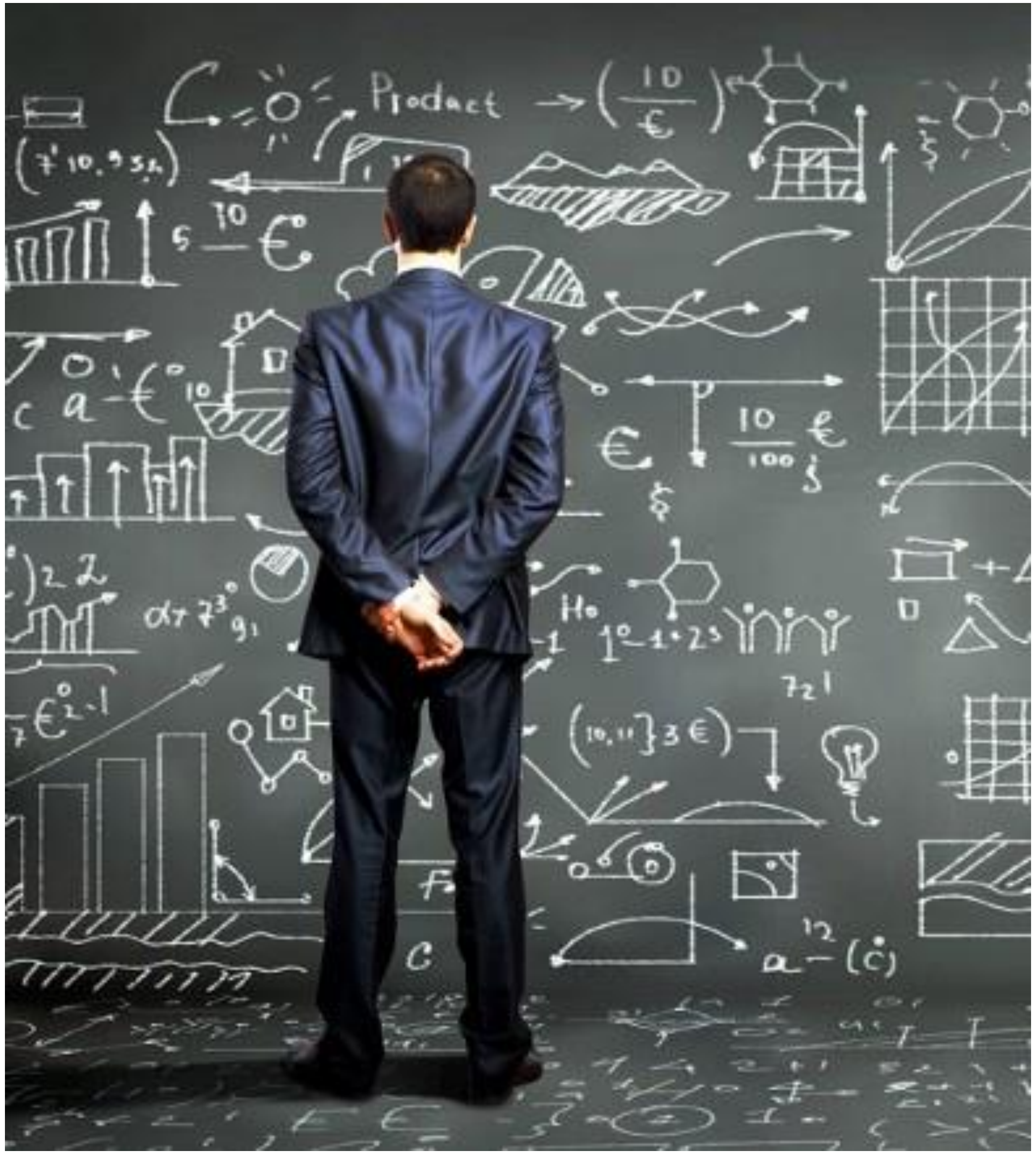
DSE 2256 DESIGN & ANALYSIS OF ALGORITHMS

Lecture 6 & 7 :

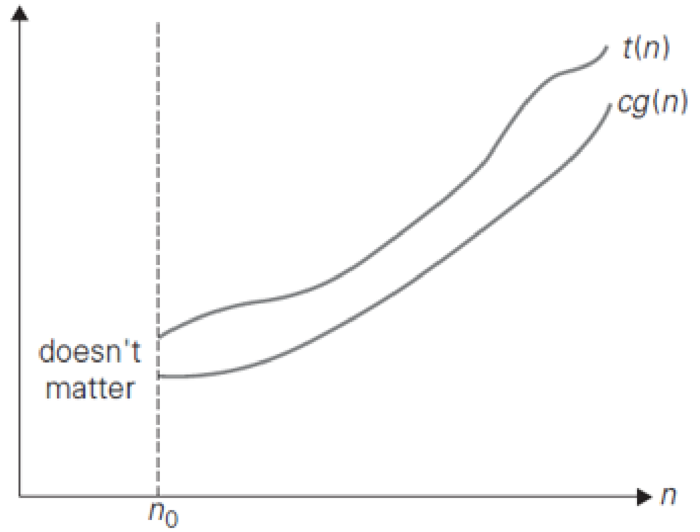
Mathematical Analysis of Non-Recursive Algorithms



Recap of L4 & L5

- Worst-case, best-case & average-case efficiencies
- Asymptotic notations
  - Big-oh (O)
  - Big-omega (Ω)
  - Theta (Θ)
- Properties of asymptotic notations
  - Analyzing algorithms with two consecutively executed parts
- Using limits to compare orders of growth
- Basic asymptotic efficiency classes

$t(n) >= c\, g(n)$ for every $n \geq n0$ n_0



$\lim_{n\to\infty}$ t

of $g(n)$

of $g(n)$

of $g(n)$

Recap of L4 & L5 : Exercise

1. **Use the definition of O, Ω and Θ to determine whether the following assertions are true or false.**

a) $\frac{n(n+1)}{2} \in O(n^3)$

b) $\frac{n(n+1)}{2} \in O(n^2)$

c) $\frac{n(n+1)}{2} \in \Theta(n^3)$

d) $\frac{n(n+1)}{2} \in \Omega(n)$

More Exercises

2. Let $f(n) = n$ and $g(n) = n^{(1+\sin n)}$, where n is a positive integer. Which of the following statements is/are correct?

I. $f(n) = O(g(n))$

II. $f(n) = \Omega(g(n))$

(A) only I

(B) Only II

(C) Both I and II

(D) Neither I nor II

3. Compare the orders of growth of $\frac{1}{2}n(n-1)$ and n^2.

More Exercises

4. Consider the following function from positive integers to real numbers:

$$10,\ n\sqrt{n},\ \log_2 n,\ 100/n$$

The CORRECT arrangement of the above functions in increasing order of asymptotic complexity is:

(A) $\log_2 n$, $100/n$, 10, $n\sqrt{n}$

(B) $100/n$, 10, $\log_2 n$, $n\sqrt{n}$

(C) 10, $100/n$, $n\sqrt{n}$, $\log_2 n$

(D) $100/n$, $\log_2 n$, 10, $n\sqrt{n}$

Mathematical analysis of Non-recursive Algorithms I

- **Example 1**

ALGORITHM *MaxElement*$(A[0..n-1])$

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if $A[i] > maxval$
        $maxval \leftarrow A[i]$
return $maxval$

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \boldsymbol{\Theta(n)}$$

Mathematical analysis of Non-recursive Algorithms II

General Plan for Analyzing the Time Efficiency:

1. **Decide on parameter *n* indicating input size.**
2. **Identify algorithm's basic operation.**
3. **Determine worst, average, and best cases for input of size *n*.**
4. **Set up a sum for the number of times the basic operation is executed.**
5. **Simplify the sum using standard formulas and rules.**

Useful summation Formulas and Rules

Important summation formulas

1. $\sum_{i=l}^{u} 1 = \underbrace{1 + 1 + \cdots + 1}_{u-l+1 \text{ times}} = u - l + 1$ (l, u are integer limits, $l \le u$); $\sum_{i=1}^{n} 1 = n$
2. $\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$
3. $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$
4. $\sum_{i=1}^{n} i^k = 1^k + 2^k + \cdots + n^k \approx \frac{1}{k+1} n^{k+1}$
5. $\sum_{i=0}^{n} a^i = 1 + a + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1}$ $(a \ne 1)$; $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$
6. $\sum_{i=1}^{n} i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2$
7. $\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \ln n + \gamma$, where $\gamma \approx 0.5772\ldots$ (Euler's constant)
8. $\sum_{i=1}^{n} \lg i \approx n \lg n$

Sum manipulation rules

1. $\sum_{i=l}^{u} ca_i = c \sum_{i=l}^{u} a_i$
2. $\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$
3. $\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i$, where $l \le m < u$
4. $\sum_{i=l}^{u} (a_i - a_{i-1}) = a_u - a_{l-1}$

Mathematical analysis of Non-recursive Algorithms III

- **Example 2**

ALGORITHM *UniqueElements*$(A[0..n-1])$

//Determines whether all the elements in a given array are distinct
//Input: An array $A[0..n-1]$
//Output: Returns "true" if all the elements in A are distinct
//   and "false" otherwise
for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[i] = A[j]$ **return false**
return true

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1]$$

$$= \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$= (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2}$$

$$= \frac{n\,(n-1)}{2} \approx \theta(n^2)$$

Mathematical analysis of Non-recursive Algorithms IV

- **Example 3**

ALGORITHM *MatrixMultiplication*$(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$

```
//Multiplies two square matrices of order n by the definition-based algorithm
//Input: Two n × n matrices A and B
//Output: Matrix C = AB
for i ← 0 to n − 1 do
    for j ← 0 to n − 1 do
        C[i, j] ← 0.0
        for k ← 0 to n − 1 do
            C[i, j] ← C[i, j] + A[i, k] ∗ B[k, j]
return C
```

$$\sum_{k=0}^{n-1} 1$$

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1$$

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = \sum_{i=0}^{n-1} n^2 = n^3$$

Running time of the algorithm on a particular machine,

$$T(n) \approx c_m M(n) = c_m n^3$$

Accurate estimate is obtained if additions are also considered,

$$T(n) \approx c_m M(n) + c_a A(n) = c_m n^3 + c_a n^3$$

$$= (c_m + c_a) n^3$$

Cm is the time of one multiplication on the machine , Ca time of one addition

EXAMPLE 4 The following algorithm finds the number of binary digits in the binary representation of a positive decimal integer.

ALGORITHM *Binary(n)*

//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
$count \leftarrow 1$
while $n > 1$ **do**
    $count \leftarrow count + 1$
    $n \leftarrow \lfloor n/2 \rfloor$
return $count$

- **The most frequently executed operation here is not inside the while loop but rather the comparison $n > 1$ that determines whether the loop's body will be executed.**
- **The value of n is halved on each repetition of the loop, log2 n.**

The exact formula for the number of times the comparison n>1 will be executed is $\lfloor \log_2 n \rfloor + 1$

The number of bits in the binary representation of *n* according to formula

Thank you!

Any queries?