SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

16IT302 – DESIGN AND ANALYSIS OF ALGORITHMS

III YEAR V SEM

UNIT-I-Introduction

TOPIC: Mathematical Analysis for Non Recursive Algorithm

Prepared by
T.Shanmugapriya,AP/IT

MATHEMATICAL ANALYSIS FOR NON RECURSIVE ALGORITHM

Subject :Design and Analysis of Algorithm
Unit :I

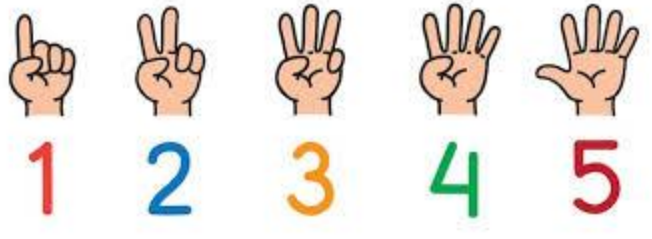
Fundamentals of the Analysis of Algorithm Efficiency

- Analysis Framework
- Asymptotic Notations and its properties
- Mathematical analysis for Recursive algorithms.
- Mathematical analysis for Non recursive algorithms.

Mathematical analysis for Non recursive algorithms.

Counting

- We just count the **number of basic operations**.
- Loops will become **series sums**
- So we'll need some **series formulas**



Example: Maximum Element

Algorithm MaxElement($A[0...n\text{-}1]$)

$maxval \leftarrow A[0]$

for $i \leftarrow 1$ **to** n-1 **do**

if $A[i] > maxval$ **then** $maxval \leftarrow A[\mathrm{i}]$

return *maxval*

What is the problem size? *n*

Most frequent operation? Comparison in the for loop

Depends on worst case or best case? No, has to go through the entire array

$C(n)$ = number of comparisons

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

Mathematical Analysis For Non Recursive Algorithms

General Plan for Analyzing the Time Efficiency of Non recursive Algorithms

- Decide on a parameter (or parameters) indicating **an input's size.**
- Identify the algorithm's **basic operation**. (As a rule, it is located in th e inner- most loop.)
- Check whether the **number of times the basic operation is executed** depends only on the size of an input. If it also depends on some additional property, the **worst-case, average-case, and, if necessary, best-case efficiencies** have to be investigated separately.
- **Set up a sum** expressing the number of times the algorithm's basic operation is executed.4
- Using **standard formulas** and rules of sum manipulation, either find a closed- form formula for the count or, at the very least, establish its order of growth.

Series Rules and Formulas

- Multiplication of a Series: $\sum_{i=l}^{u} ca_i = c\sum_{i=l}^{u} a_i$
- Sum of two sequences: $\sum_{i=l}^{u} (a_i + b_i) = \sum_{i=l}^{u} a_i + \sum_{i=l}^{u} b_i$
- Sum of constant sequences: $\sum_{i=l}^{u} 1 = u - l + 1$
- Sum of linear sequences: $\sum_{i=0}^{n} i = n(n+1)/2$ = length of sequence times the average of the first and last el ements

Example: Uniqueness

Consider the ***element uniqueness problem***: check whether all the elements in a given array of *n* elements are distinct. This problem can be solved by the following straightforward algorithm.

Algorithm *UniqueElements*($A[0...n\text{-}1]$)

for $i \leftarrow 0$ **to** $n\text{-}2$ **do**

for $j \leftarrow i+1$ **to** $n\text{-}1$ **do**

if $A[i] = A[j]$ **return false**

return true

List
10
20
30
30
50
60
70

List has duplicates

Uniqueness

1. Problem size? ***n***

2. Basic operation? **if-test**

3. Worst and best case are different. Best case is when the first two elements are equal the n **Θ(*n*)**

Worst case is if array elements are unique then all sequences of the for loops are executed

Uniqueness

4. The sum:

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

5. Solove

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

$$= \sum_{i=0}^{n-2}[n-1- i] = \sum_{k=n-1}^{1}k \qquad \text{where } k = n - i -1$$

$$\boldsymbol{C_{worst}(n) = \sum_{k=1}^{n-1}k = (n-1)(n-1+1)/2 = n(n-1)/2\ \varepsilon\ \Theta(n^2)}$$

Note for a unique array there is minimal of $n(n-1)/2$ comparisons. Is this neces sary, is there a better algorithm?

Yes we could pre-sort.

Example: Binary Length

The following algorithm finds the number of binary digits in the binary representa tion of a positive decimal integer.

Algorithm *Binary*(*n*)

count ← 1

while *n* > 1 do

count++

n ← floor(*n*/2)

return *count*

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Binary Length

1. Problem size? **integer, *n***
2. Basic operation? **comparison in the while loop**
3. Worst and best case are the same.
4. The sum:

How many times is the while loop executed?

approximately lg(n), exactly lg(n) + 1 because it must fail once

$$\boldsymbol{C(n) = \sum_{i=1}^{\lg(n)+1} 1}$$

5. Solve

$$\boldsymbol{C(n) = \lg(n) + 1 - 1 + 1 \in \Theta(\lg(n))}$$

Example -Matrix multiplication

Given two *n n* matrices *A* and *B*, find the time efficiency of the definition-based algorithm for computing their product *C AB*. By definition, *C* is an *n n* matrix whose elements are computed as the scalar (dot) products of the rows of matrix *A* and the col umns of matrix *B*:

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

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 $n \times n$ matrices A and B

//Output: Matrix $C = AB$

for $i \leftarrow 0$ **to** $n-1$ **do**

for $j \leftarrow 0$ **to** $n-1$ **do**

$C[i, j] \leftarrow 0.0$

for $k \leftarrow 0$ **to** $n-1$ **do**

$C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$

return C

Example -Matrix multiplication

and the total number of multiplications *M(n)* is expressed by the following triple sum:

. . .

$M(n)=(x+a)^n = \sum(n) \sum(n) \sum(n)$

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

Assessment

Write the missing steps to analyze non recursive algorithms

1. input’s size
2. ---------------------
3. number of times the basic operation is executed
4. ---------------------
5. ---------------------

Thank you!