Text Book:
Introduction to the Design and Analysis of Algorithms
Author: Anany Levitin
2nd Edition

Mathematical Analysis of Non-Recursive algorithms

General Plan for Analysing the Time Efficiency of Non-recursive Algorithms

1. Decide on a *parameter* (or parameters) indicating an input's size.
2. Identify the algorithm's *basic operation* (The operation that consumes maximum amount of execution time).
3. Check whether the *number of times the basic operation is executed* depends only on the size of an input. If the number of times the basic operation gets executed varies with specific instances (inputs), we need to carry out Best, Worst and Average case analysis
4. Set up a *sum* expressing the number of times the algorithm's basic operation is executed.
5. Simplify the sum using standard formulas and rules , establish its *order of growth*

EXAMPLE 1:

Find the largest element in a list of n numbers.

Assumption: list is implemented as an array.

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 ←A[0]
for i ←1 **to** n − 1 **do**
    if A[i]>maxval
        maxval←A[i]
return maxval

Algorithm analysis

- The measure of an input's size: the number of elements in the array, i.e., n.
- Basic Operation
  There are two operations in the for loop's body:
  - A[i]> maxval
  - Maxval ←A[i].

  The comparison operation is considered as the algorithm's basic operation, because the comparison is executed on each repetition of the loop and not the assignment.
- Best /Worst/Average Case
  The number of comparisons will be the same for all arrays of size n; therefore, there is no need to distinguish among the worst, average, and best cases for this problem.
- The algorithm makes one comparison on each execution of the loop, which is repeated for each value of the loop's variable i within the bounds 1 and n-1 (inclusively). Hence,

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

$$\Rightarrow T(n) \in \Theta(n)$$

EXAMPLE 2:

Element uniqueness problem:

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 ←0 **to** n − 2 **do**
    for j ←i + 1 **to** n − 1 **do**
        if A[i]= A[j]
            return false
return true

Algorithm analysis

- Input size: n (the number of elements in the array).
- Basic Operation: Comparison if A[i]= A[j]
- The number of element comparisons depends not only on n but also on whether there are equal elements in the array and, if there are, which array positions they occupy. So we need to do best, worst and average case analysis we discuss best and worst case analysis here
  - Best-case situation:
    First two elements of the array are the same
    Number of comparison. Best case = 1 comparison.
  - Worst-case situation:
    The worst- case happens for two-kinds of inputs:
    – Arrays with no equal elements
    – Arrays in which only the last two elements are the pair of equal elements

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

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

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

$\Rightarrow \quad T(n)_{worst} \in O(n)$

EXAMPLE 3:

Matrix multiplication.

C=A*B
C[i, j]= A[i, 0]B[0, j]+ . . . + A[i, k]B[k, j]+ . . . + A[i, n − 1]B[n − 1, j]
for every pair of indices 0 ≤ i, j ≤ n − 1.

ALGORITHM MatrixMultiplication(A[0..n − 1, 0..n − 1], B[0..n − 1, 0..n − 1])
//Multiplies two square matrices of order n
//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

Algorithm analysis

- Input Size: matrix order n.
- There are two arithmetical operations in the innermost loop, multiplication and addition. But we consider multiplication as the basic operation as multiplication is more expensive as compared to addition
- Total number of elementary multiplications executed by the algorithm depends only on the size of the input matrices, we do not have to investigate the worst-case, average-case, and best-case efficiencies separately.

  $M(n) \in \Theta(n^3)$

  $\Rightarrow$ **$T(n) \in \Theta(n^3)$**

EXAMPLE 4

Determine 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 ←1
while n > 1 **do**
    count ←count + 1
    n←n/2
return count

Algorithm analysis

- An input's size is n.
- Basic operation Either Division or Addition
- Let us consider addition as basic operation. Number of times addition is executed depends only on the value of n so we don't need to do best, worst and average case analysis separately
- The loop variable takes on only a few values between its lower and upper limits. Since the value of n is about halved on each repetition of the loop, so number of times count ←count + 1 is executed is $\log_2 n+ 1$.

  $\Rightarrow T(n) \in (\log_2 n)$