Desain & Analisis Algoritma

Mathematical Analysis of Non-Recursive Algorithms

General Plan

1. Decide on a parameter indicating an input's size.
2. Identify the algorithm's basic operation.
3. Check wether 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, determine the worst, best, and average case.
4. Set up a sum expressing the number of times the algorithm's basic operation is executed.
5. Using standard formulas and rules of sum manipulation, at the very last, establish its order of growth

Analyzing nonrecursive algorithm

- **Example 1**. Consider the problem of finding the value of the largest element in a list of n numbers. For simplicity, 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
```

- The algorithm's basic operation is comparison A[i]>*maxval*
- There's no need to distinguish the worst, best, and average case. Why ?
- C(n) is number of times the comparison is executed.

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

EXAMPLE 2 Consider the ***element uniqueness problem***: check whether all the elements in a given array are distinct. This problem can be solved by the following straightforward algorithm.

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

```
//Checks 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
```

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

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

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

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-by-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
```

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

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 ← 1
while n > 1 do
    count ← count + 1
    n ← ⌊n/2⌋
return count
```

Exercise

a. Write an algorithm to compute :

$$1^2+2^2+3^2+4^2+\ldots n^2$$

b. What its basic operation ?

c. How many times is basic operation executed?

Exercises (cont.): calculating O, Ω, *and* Θ

```
for (i=1; i<n; i++)
     sum++;
```

```
        for (i=1; i<n; i=i+2)
             sum++;
```

```
       for (i=0; i<n; i++)
            for (j = 0; j<i; j++)
                 sum++;
```

Calculating (cont.)

```
for (i=1; i<n; i=i*2)
     sum++;
```

```
for (i=1; i<n; i++)
     for (j=1; j < n/2; j++)
          sum++;
```

Next..

Mathematical Analysis of Recursive Algorithm