3 Analysis of Non-Recursive Functions

Tuesday, September 3, 2024 11:57 AM

E.G.

```
FUNCTION MaxElement(A[0...n-1])
    maxval ← A[0]                          } 2
    for i ← 1 to n-1 do                    } n-1 Σ_{i=1} 4
        if A[i] > maxval                   } 2
            maxval ← A[i]                  } 2   } 4
    return maxval.
```

$2 + \sum_{i=1}^{n-1} 4$

$R_{MaxE}(n) = 2 + (n-1)4 =$
$= 2 + 4n - 4$
$= 4n - 2$
$4n - 2 \in \Theta(n)$

$2 + (4 + 4 + 4 + \dots + 4)$
i: 1 2 3 ⋯ n-1
n-1

Computing whole runtime function: too much work.

Instead lets consentrate on a **basic operation**
an operation that characterizes the algorithm.

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

Dont trust that a single loop is always n
a double loop is always n^2

E.G.

```
FUNCTION foo(A[0...n-1])
    for i = 1 to n*n
        do something ← basic operation.
```

GENERAL PLAN FOR ANALYSING THE TIME EFFICIENCY OF NON_RECURSIVE ALGORITHMS

1.- Decide on parameter(s) that indicate the input size.

2.- identify the basic operation
   ↪ find it in the innermost loop
3.- check whether the number of times the basic operation is executed depends only on input size or on other factors.
   if so: worst-case analysis is enough
   otherwise: average-case or amortized analysis may be applicable.
4.- Set up a sum counting the number of executions of the basic operation.
5.- Simplify sum to closed form
   by - rules of sum manipulation
      - standard sum formulas.
   or at least, try to ascertain the order-of-growth of the function.

SUMS : REVIEW

- $\sum_{i=L}^{U} a \cdot i = a \cdot \sum_{i=L}^{U} i$
- $\sum_{i=L}^{U} (a \pm b) = \sum_{i=L}^{U} a + \sum_{i=L}^{U} b$
- $\sum_{i=L}^{U} 1 = U - L + 1$
- $\sum_{i=0}^{U} i = \sum_{i=1}^{U} = 1+2+3+\ldots = \frac{U(U+1)}{2} \approx \frac{1}{2}U^2 \in \Theta(U^2)$

E.G.

```
FUNCTION allUnique( A[0..n-1])
    for i ← 0 to n-2 do
        for j ← i+1 to n-1 do
            if A[i] = A[j] return false
    return true.
```

basic operation

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

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

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

$$= \sum_{i=0}^{n-2} \left[n - 1 - i \right]$$

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

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

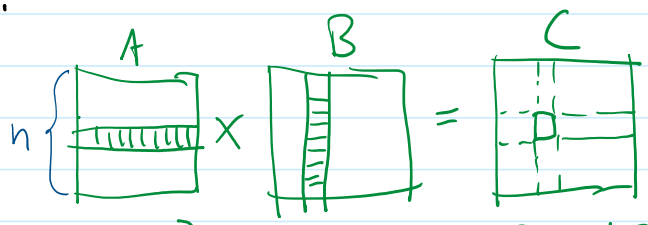
$$= (n-1) \cdot (n-2-0+1) - \frac{(n-2)(n-1)}{2}$$

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

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

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

E.G.



$$C[i,j] = A[i,0] \cdot B[0,j] + A[i,1] \cdot B[1,j] + A[i,2] \cdot B[2,j] + \dots + A[i,n-1] \cdot B[n-1,j]$$

```
FUNCTION SqMatrixMult(A, B, VAR C)
    for i ← 0 to n-1
        for j ← 0 to n-1
            C[i,j] ← 0.0
            for k ← 0 to n-1
                C[i,j] ← C[i,j] + A[i,k] * B[k,j]
```

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

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n$$

$$= \sum_{i=0}^{n-1} n \cdot \sum_{j=0}^{n-1} 1$$

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

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

- What about while?
  - worst case of while? infinite loop.
  - estimate the worst-case terminating scenario. from conditions in while loop.

E.G. A while loop and a non-polinomial iterative algorithm.

```
FUNCTION BinLen(n)
    // the number of binary digits needed to represent n
    count ← 1
    while n > 1 do
        count ← count + 1
        n ← ⌊n/2⌋
    return count.
```

n		
16	10000	5
255	1111111	8

basic operation.

How many times the loop repeats?

↳ How many times can we divide n by 2

$$D(n) = \log_2 n \in \Theta(\log n)$$

save question as: $2^k = n$ k?

Why we don't care about the base?

for bases a b

$$\log_a X = \frac{\log_b X}{\log_b a} \quad : \quad \log_b X = \log_b a \cdot \log_a X$$

$$= C \cdot \log_a X$$

$\log_b X$ is $\Theta(\log_a X)$

—o— EOF