Running Time Expressed as Sum

Consider

$$
\begin{array}{l}
\textbf{for } j \leftarrow 1 \textbf{ to } n^2 \\
\quad \textbf{for } k \leftarrow 1 \textbf{ to } j^3 \\
\quad\quad q \leftarrow q + j + k
\end{array}
$$

Inner Loop

- iterates j^3 times
- i'th iteration takes constant time, $\Theta(1)$

Thus the inner loop runs in time (proportional to)

$$\sum_{i=1}^{j^3} 1 = j^3$$

Outer Loop

- iterates n^2 times
- in i'th iteration, $j = i$ so it runs in time i^3

Thus the outer loop runs in time (proportional to)

$$\sum_{i=1}^{n^2} i^3 \in \Theta(n^{???})$$

Estimating a Sum

Let us estimate

$$T_0(n) = \sum_{i=1}^{n^2} i^3$$

- easy upper bound: n^8
- easy lower bound: n^2

but actually $T_0(n) \in \Omega(n^8)$ (and thus $T_0(n) \in \Theta(n^8)$)

- as seen from an exact calculation using non-trivial math
- but which can also be demonstrated using a simple trick which generalizes to a wide variety of sums.

Estimating General Sums

Recall that the specific sum

$$T_0(n) = \sum_{i=1}^{n^2} i^3$$

has solution

$$T_0(n) \in \Theta(n^8) = \Theta(n^2\,(n^2)^3)$$

This is special case, with $g(n) = n^2$ and $f(i) = i^3$, of

$$T(n) = \sum_{i=1}^{g(n)} f(i)$$

which we may thus expect to have solution

$$T(n) \in \Theta(g(n)\,f(g(n)))$$

That is, we can approximate a sum by multiplying

- the number of terms
- the last term

but only under certain (quite common) conditions.

Requirement: Non-Decreasing

For f non-decreasing we clearly have

$$\sum_{i=1}^{g(n)} f(i) \in O(g(n)\, f(g(n)))$$

but it does not always hold that

$$\sum_{i=1}^{g(n)} f(i) \in \Omega(g(n)\, f(g(n)))$$

- Problem: the rectangle cover for a sharply increasing function covers too much
- Fix: demand f to grow smoothly

A non-decreasing function f is smooth if there exists c and n_0 such that for all $n \geq n_0$: $f(2n) \leq cf(n)$

- all polynomials are smooth (if positive coefficients)
- exponentials are not smooth

General Theorem

For f smooth (and thus non-decreasing), we have (assuming g eventually becomes greater than any constant)

$$\sum_{i=1}^{g(n)} f(i) \in \Omega(g(n)\, f(g(n)))$$

and thus even

$$\sum_{i=1}^{g(n)} f(i) \in \Theta(g(n)\, f(g(n)))$$

as is the content of *Howell*'s Theorem 3.28.

Analyzing Multiple Nesting

for $i \leftarrow 0$ **to** n
    for $j \leftarrow i$ **to** n
        $\mathrm{sum} \leftarrow 0$
        for $k \leftarrow i$ **to** $j - 1$
            $\mathrm{sum} \leftarrow \mathrm{sum} + A[k]$
        $m \leftarrow \mathsf{Max}(m, \mathrm{sum})$

Middle Loop:

- iterates $n - i + 1$ times
- for q'th iteration, $j = q + i - 1$ and thus running time is proportional to $q - 1$

and thus total running time is

$$\sum_{q=1}^{n-i+1} (q-1) \in \Theta((n-i+1)(n-i)) = \Theta((n-i)^2)$$

Outer Loop can be estimated by sum

$$\sum_{i=0}^{n} (n-i)^2 = \sum_{i=1}^{n} i^2 \in \Theta(n \cdot n^2) = \Theta(n^3)$$

Analyzing While Loops

$$
\begin{array}{l}
x, k \leftarrow 0, 1 \\
\textbf{while } k \le n \\
\quad k \leftarrow k + k \\
\quad x \leftarrow x + 1
\end{array}
$$

- each iteration takes constant time, $\Theta(1)$
- in i'th iteration, k increases from 2^{i-1} to 2^i.
- the loop terminates when $k > n$, that is $2^i > n$, that is $i > \lg(n)$

Thus the total running time is proportional to

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

Reasoning about Worst Case

Consider the algorithm

Input: A strictly increasing
Output: x removed from A

for $i \leftarrow 1$ **to** n
  if $A[i] = x$
    for $j \leftarrow i$ **to** $n - 1$
      $A[j] \leftarrow A[j+1]$

- the inner loop has worst case running time $\Theta(n)$ which suggests total running time in $\Theta(n^2)$
- but at most one iteration can exhibit worst-case behavior

Correct analysis:

- total time spent shifting: $O(n)$
- total time spent apart from shifting: $\Theta(n)$

and thus total running time is in $\Theta(n)$.

Insertion Sort, Revisited

Recall

for $i \leftarrow 2$ **to** n
  $j \leftarrow i$
  while $j > 1$ and $A[j] < A[j-1]$
    $A[j] \leftrightarrow A[j-1]$
    $j \leftarrow j - 1$

- in the worst case, inner loop runs in in $\Theta(i)$
- that worst case may happen every iteration (when the array is sorted in reverse order)
- hence the worst case total running time is proportional to

$$\sum_{i=1}^{n} i \in \Theta(n \cdot n) = \Theta(n^2)$$

Elementary instructions

Recall

$i, j \leftarrow 1, 1$
for $k \leftarrow 1$ **to** $n - 1$
    $i, j \leftarrow j, i + j$
return j

- We may naively think this runs in linear time
- but in the k'th iteration we add $\Theta(k)$-bit numbers so the running time is rather $1 + 2 + \ldots + n \in \Theta(n^2)$

Still, for most programs we can safely assume that assignments execute in constant time.