Data Structures and Algorithms

Spring 2008-2009

Outline

1. Algorithm Analysis (contd.)
    - A Bean-Counting Approach
    - A Saner Approach

Outline

1. Algorithm Analysis (contd.)
    - A Bean-Counting Approach
    - A Saner Approach

Our First Algorithm

Compute $S = \sum_{i=1}^{n} i^2$

- We would expect S to be $\simeq \frac{1}{3}n^3$ as given in Lect01
- But it is *running time* we care about here

```
int sum(int n)
{
   int partial_sum = 0;          // l. 3

   for (int i = 1; i <= n; i++) // l. 5
      partial_sum += i*i;        // l. 6

   return partial_sum;           // l. 8
}
```

Our First Algorithm (contd.)

Counting Operations

- Lines 3 and 8 get executed once so they contribute 2 units
- Line 5 will contribute $1, n+1, 2n$ units respectively for the three parts; total $= 3n+2$
- Line 6 contributes 1 add, 1 mult and 1 assignment each time it is executed, for a total of $3n$
- Full count of operations is $T(n) = 6n+4$
- Therefore the *asymptotic* running time of the algorithm is $\Theta(n)$, *i.e.*, both $O(n)$[1] and $\Omega(n)$[2]

[1] $T(n) = 6n+4 < 7n, n > 4$

[2] $T(n) = 6n+4 > 5n, n \geq 0$

Outline

1. Algorithm Analysis (contd.)
   - A Bean-Counting Approach
   - A Saner Approach

General Principles

Abstraction: Less Pain, More Gain

- The previous analysis is too painful so we concentrate only on loops and recursive function calls
- Also, when analysing nested loops, we analyse them from inside to out; but watch out for loops that contribute only a constant amount of work each time through

Nested loops: $O(n^2)$

```
for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j++)
      a[i][j]+= i+j;
```

- Ignoring the overheads of the two loops, running time is the number of times the increment operator is applied: $n \cdot n$

General Principles (contd.)

Nested loops: $O(n^2)$

```
for (int i = 0; i < n; i++)
   for (int j = i; j < n; j++)
      a[i][j]+= i+j;
```

- Inside loop gets executed $n - i$ times; total number of increments is $n + (n-1) + \ldots + 1 = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$

General Principles (contd.)

Nested loops: $O(n)$

```
for (int i = 0; i < 5; i++)
    for (int j = i; j < n; j++)
        a[i][j]+= i+j;
```

- Since the outer loop only executes a *fixed* number of times, the running time is simply a constant times the inner loop.

General Principles (contd.)

Loops The running time of a loop is at most the running time of the statements inside the loop (including tests) times the number of iterations

Nested Loops The total running time of a statement inside a group of nested loops is the running time of the statement times the product of the sizes of the enclosing loops

Consecutive Statements The combined running time is the sum of the running times $\Longrightarrow$ the maximum is the one that counts

General Principles (contd.)

If-then-else If there is a statement:

```
if (condition)
    s1;
else
    s2;
```

then the running time is no more than the larger of the running times of `s1` and `s2` plus the time taken to make the test, `condition`