ASYMPTOTIC COMPLEXITY

CS2111
CS2110 – Fall 2014

Readings, Homework

Issues

1. How to look at a program and calculate, formally or informally, its execution time.
2. Determine whether some function f(n) is O(g(n))

Worst case for selection / insertion sorts

```
Selection sort b[0..n-1]
//inv b[0..i-1] sorted,  b[0..i-1] <= b[i..n-1]
for (int i= 0; i < n; i= i+1) {
   int j= pos of min of b[i..n-1];
   Swap b[i] and b[j]
}
Insertion sort b[0..n-1]
//inv: b[0..i-1] sorted
for (int i= 0; i < n; i= i+1) {
   Push b[i] down to its sorted;
   position in b[0..i]
}
```

Count swaps?

Iteration i requires 1 swap.
Total of n

Number of swaps is not the thing to count!

Iteration i requires i swaps.
Total of 0 + 1 + … n-1 =
(n-1) n / 2

Worst case for selection / insertion sorts

Count array element comparisons

```
Selection sort b[0..n-1]
//inv b[0..i-1] sorted,  b[0..i-1] <= b[i..n-1]
for (int i= 0; i < n; i= i+1) {
   int j= pos of min of b[i..n-1];
   Swap b[i] and b[j]
}
Insertion sort b[0..n-1]
//inv: b[0..i-1] sorted
for (int i= 0; i < n; i= i+1) {
   Push b[i] down to its sorted;
   position in b[0..i]
}
```

ALWAYS n-i comparisons
Total of (n-1) n / 2

Iteration i requires i comparisons in worst case, 1 in best case.
Total of 0 + 1 + … n-1 =
(n-1) n / 2 (worst case)

Find first occurrence of r in s (indexOf)

```
/** = position of first occurrence of r in s (-1 if not in) */
public static int find(String r, String s) {
   int nr= r.length(); int ns= s.length();
   // inv: r is not in s[0..i-1+nr-1]
   for (int i= 0; i < ns – nr; i= i+1) {
      if (s.substring(i, i+nr).equals(r))
         return i;
   }
   return -1;
}
```

How much time does this take   O(nr)

Executed how many times --worst case?   ns – nr + 1

Therefore worst-case time is O(nr *(ns –nr + 1))

nr = 1: O(ns).   nr = ns: O(ns).   nr = ns/2: O(ns*ns)

Dealing with nested loops

```
int c = 0;
for (int i= 0; i < n; i++) {                 n iterations
   for (int j= 0; j < n; j++) {              n iterations
      if ((j % 2) == 0) {                    True n*n/2 times
         for (int k= i; k < n; k++)  c= c+1;
      }
      else {
         for (int h= 0; h < j; h++) c= c+1;
      }
   }
}
```

Loop is executed n/2 times, with i = 0, 1, 2, …, n-1
It has n-i iterations. That's 1 + 2 + … n = n*(n+1)/2 its.
That's O(n*n*n)

Dealing with nested loops

7

```
int i= 0;  int c= 0;
while (i < n) {
    int k= i;
    while (k < n  &&  b[k] == 0) {
        c= c+1;  k= k + 1;
    }
    i= k+1;
}
```

What is the execution time?

It is O(n). It looks at Each element of b[0..n-1] ONCE.

Using Big-O to Hide Constants

8

We say f(n) is *order of* g(n) if f(n) is bounded by a constant times g(n)

Notation: f(n) is O(g(n))

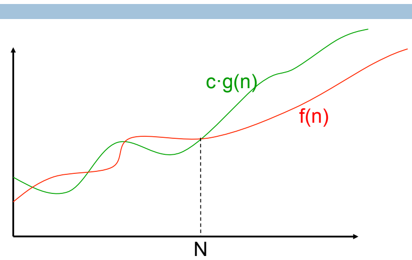
Roughly, f(n) is O(g(n)) means that f(n) grows like g(n) or slower, to within a constant factor

"Constant" means fixed and independent of n

Formal definition: f(n) is O(g(n)) if there exist constants c and N such that for all n ≥ N, $f(n) \le c \cdot g(n)$

A Graphical View

9



To prove that f(n) is O(g(n)):

- Find N and c such that $f(n) \le c\, g(n)$ for all n > N
- Pair (c, N) is a *witness pair* for proving that f(n) is O(g(n))

Big-O Examples

10

Let $f(n) = 3n^2 + 6n - 7$

Prove that f(n) is $O(n^2)$

$3n^2 + 6n - 7 \quad <= \quad c\, n^2$?

What c? what N?

$3n^2 + 6n - 7$
$< \quad 3n^2 + 6n$
$<= \quad 3n^2 + 6n^2$ for n >= 1
$= \quad 9n^2$

For n >= 1, $n <= n^2$

Choose N = 1 and c = 9

f(n) is O(g(n)) if there exist constants c and N such that for all n ≥ N, $f(n) \le c \cdot g(n)$

Big-O Examples

11

Let $f(n) = 3n^2 + 6n + 7$

Prove that f(n) is $O(n^2)$

$3n^2 + 6n + 7 \quad <= \quad c\, n^2$?

What c? what N?

$3n^2 + 6n + 7$
$<= \quad 3n^2 + 6n^2 + 7$ for n >= 1
$= \quad 9n^2 + 7$
$<= \quad 9n^2 + n^2$ for n >= 7
$= \quad 10n^2$

For n >= 1, $n <= n^2$

Choose N = 7 and c = 10

f(n) is O(g(n)) if there exist constants c and N such that for all n ≥ N, $f(n) \le c \cdot g(n)$

Big-O Examples

12

Let $f(n) = 3n^2 + 6n - 7$

Prove that f(n) is $O(n^3)$

So, f(n) is $O(n^2)$ $O(n^3)$, $O(n^4)$, …

$3n^2 + 6n - 7$
$< \quad 3n^2 + 6n$
$<= \quad 3n^2 + 6n^2$ for n >= 1
$= \quad 9n^2$
$<= \quad 9n^3$ for n >= 1

Choose N = 1 and c = 9

f(n) is O(g(n)) if there exist constants c and N such that for all n ≥ N, $f(n) \le c \cdot g(n)$

Big-O Examples

13

T(0) = 1
T(n) = 2 * T(n-1)
Give a closed formula (no recursion) for T(n)

T(0) = 1
T(1) = 2
T(2) = 4
T(3) = 8

One idea:
Look at all small cases and find a pattern

T(n) = 2^n

Big-O Examples

14

For quicksort in best case, i.e. two partitions are same size.
T(0) = 1
T(1) = 1
T(n) = K*n + 2 * T(n/2) // The Kn is to partition array

T(0) = K //Simplify computation: assume K > 1
T(1) = K // And use K instead of 1
T(2^1) = T(2) = 2K + 2K = 4K
T(2^2) = T(4) = 4K + 2(4K) = 12K = 3*(2^2)K
T(2^3) = T(8) = 8K + 2(12K) = 32K = 4*(2^3)K
T(2^4) = T(16) = 16K + 2(32K) = 80K = 5*(2^4)K

T(2^n) = (n+1)*(2^n)*K T(m) = log(2m)*m*K