Running Time Analysis

Introduction to O-notation

How can we quantify and compare performance of different algorithms given:

- different machines, processors, architectures?
- different size data sets, orderings?
- different computer languages?
- different compilers?

Unfortunately, raw performance times don't tell us much (rigorously).

Possible Approaches

- Benchmarks -- test data or test programs that are designed to help us quantitatively evaluate performance.
- O-notation (Big-O)

Quantify and compare performance of different algorithms that is independent of:

- machine, processor, architecture
- size of data sets, ordering of data
- computer language
- compiler used

```
void guess_game(int n)
{
   int guess;
   char answer;

   assert(n >= 1);

   cout << "Think of a number between 1 and " << n << ".\n";
   answer = 'N';
   for(guess = n; guess > 0 and answer != 'Y' and answer != 'y';--guess)
   {
         cout << "Is your number " << guess << "?" << endl;
         cout << "Please answer Y or N, and press return:";
         cin >> answer;
   }
   if(answer == 'Y' or answer == 'y') cout << "Got it :) \n";
   else cout << "I think you are cheating :( \n";
}
```

Algorithm Performance

- Worst case performance?
- Best case performance?
- Average case performance?

Algorithm Performance

- Worst case performance: loops n times!!
- Best case performance?
- Average case performance?

Algorithm Performance

- Worst case performance: loops n times!!
- Best case performance: loops once.
- Average case performance?

Algorithm Performance

- Worst case performance: loops n times!!
- Best case performance: loops once.
- Average case performance:
  - assume: all answers between 1 and n are equally likely.
  - average case

```
   void guess_game(int n)
   {
       int guess;
       char answer;

1      assert(n >= 1);

1      cout << “Think of a number between 1 and ” << n << “.\n”;
1      answer = ‘N’;
2n+2  for(guess = n; guess > 0 and answer != ‘Y’ and answer != ‘y’;--guess)
       {
n          cout << “Is your number ” << guess << “?” << endl;
n          cout << “Please answer Y or N, and press return:”;
n          cin >> answer;
        }
1      if(answer == ‘Y’ or answer == ‘y’)
1          cout << “Got it :) \n”;
1       else cout << “I think you are cheating :( \n”;
    }
Total: f(n) = 5n +  7
```

Computation required as function of n

- What is the total number of operations needed in guess_game?
- The number of operations required is a linear function of n: f(n) = c + kn.
- As n increases, computation required increases linearly. We say it is O(n).

Why Simplify?

- As n gets bigger, highest order term dominates.
- Take for instance

- then when n = 2000, the square term accounts for more than 99% of running time!!

Examples

Examples

Intuition

scale of strength ↓

Adjective	O-notation
constant	O(1)
logarithmic	O(logn)
linear	O(n)
nlogn	O(nlogn)
quadratic	$O(n^2)$
cubic	$O(n^3)$
exponential	$O(2^n)$, $O(10^n)$, etc.

Intuition

scale of strength ↓

Example	O-notation
constant	O(1)
binary search	O(logn)
scale vector	O(n)
vector, matrix multiply	$O(n^2)$
matrix, matrix multiply	$O(n^3)$

Running time for algorithm

f(n)	n=256	n=1024	n=1,048,576
1	1μsec	1μsec	1μsec
$\log_2 n$	8μsec	10μsec	20μsec
n	256μsec	1.02ms	1.05sec
$n \log_2 n$	2.05ms	10.2ms	21sec
n^2	65.5ms	1.05sec	1.8wks
n^3	16.8sec	17.9min	36,559yrs
2^n	3.7×10^{63}yrs	5.7×10^{294}yrs	2.1×10^{315639}yrs

Largest problem that can be solved if Time <= T at 1μsec per step

f(n)	T=1min	T=1hr	T=1wk	T=1yr
n	6×10^7	3.6×10^9	6×10^{11}	3.2×10^{13}
nlogn	2.8×10^6	1.3×10^8	1.8×10^{10}	8×10^{11}
n^2	7.8×10^3	6×10^4	7.8×10^5	5.6×10^6
n^3	3.9×10^2	1.5×10^3	8.5×10^3	3.2×10^4
2^n	25	31	39	44

Warning:

Some algorithms do not always take the same amount of time for problems of a given size n.

Worst case performance vs.
Average case performance

In general, best case performance is not a good measure.

Formal Definition

We say f(n) is O(g(n)) if there exist two positive constants k and n_0 such that

$|f(n)| <= k|g(n)|$ for all $n >= n_0$

The total number of steps does not exceed g(n)*constant provided we deal with sufficiently large problems (large n).