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 \ge 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;</pre>
       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?

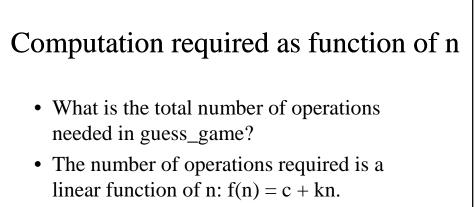
Algorithm Performance

- 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

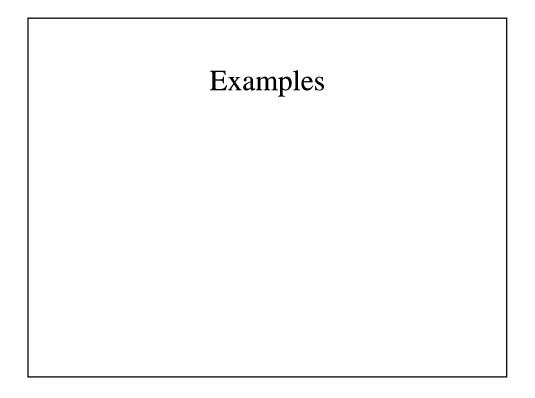
```
void guess_game(int n)
  {
    int guess;
    char answer;
     assert(n \ge 1);
1
     cout << "Think of a number between 1 and " << n << ".\n";
1
     answer = 'N';
1
2n+2 for(guess = n; guess > 0 and answer != 'Y' and answer != 'y';--guess)
     {
       cout << "Is your number " << guess << "?"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

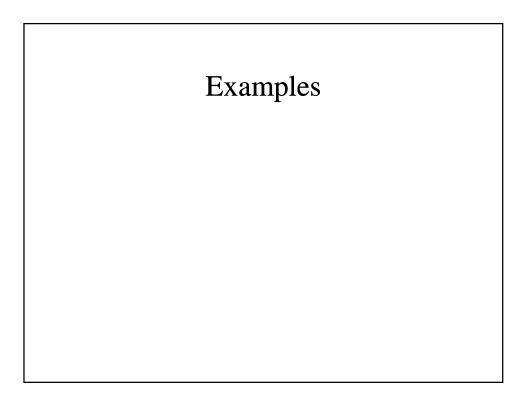


• As n increases, computation required increases linearly. We say it is O(n).

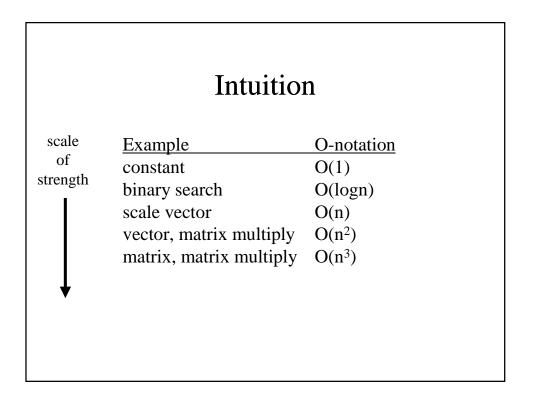
Why Simplify?

- 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!!





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



Running time for algorithm							
f(n)	n=256	n=1024	<u>n=1,048,576</u>				
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	3.7x10 ⁶³ yrs	5.7x10 ²⁹⁴ yrs	2.1x10 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	6x10 ⁷	3.6x10 ⁹	6x10 ¹¹	3.2×10^{13}			
nlogn	2.8×10^{6}	1.3x10 ⁸	1.8×10^{10}	8×10^{11}			
n^2	7.8×10^3	6x10 ⁴	7.8x10 ⁵	5.6×10^{6}			
n ³	3.9×10^2	1.5×10 ³	8.5x10 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

 $|f(n)| \le k|g(n)|$ for all $n \ge n_0$

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