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:
  - assume: all answers between 1 and n are equally likely.
  - average case

```cpp
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";
}
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

<table>
<thead>
<tr>
<th>Adjective</th>
<th>O-notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>O(1)</td>
</tr>
<tr>
<td>logarithmic</td>
<td>O(logn)</td>
</tr>
<tr>
<td>linear</td>
<td>O(n)</td>
</tr>
<tr>
<td>nlogn</td>
<td>O(nlogn)</td>
</tr>
<tr>
<td>quadratic</td>
<td>O(n²)</td>
</tr>
<tr>
<td>cubic</td>
<td>O(n³)</td>
</tr>
<tr>
<td>exponential</td>
<td>O(2ⁿ), O(10ⁿ), etc.</td>
</tr>
</tbody>
</table>

## Intuition

<table>
<thead>
<tr>
<th>Example</th>
<th>O-notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>O(1)</td>
</tr>
<tr>
<td>binary search</td>
<td>O(logn)</td>
</tr>
<tr>
<td>scale vector</td>
<td>O(n)</td>
</tr>
<tr>
<td>vector, matrix multiply</td>
<td>O(n²)</td>
</tr>
<tr>
<td>matrix, matrix multiply</td>
<td>O(n³)</td>
</tr>
</tbody>
</table>
## Running time for algorithm

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$n=256$</th>
<th>$n=1024$</th>
<th>$n=1,048,576$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1$</td>
<td>1μsec</td>
<td>1μsec</td>
<td>1μsec</td>
</tr>
<tr>
<td>$\log_2 n$</td>
<td>8μsec</td>
<td>10μsec</td>
<td>20μsec</td>
</tr>
<tr>
<td>$n$</td>
<td>256μsec</td>
<td>1.02ms</td>
<td>1.05sec</td>
</tr>
<tr>
<td>$n \log_2 n$</td>
<td>2.05ms</td>
<td>10.2ms</td>
<td>21sec</td>
</tr>
<tr>
<td>$n^2$</td>
<td>65.5ms</td>
<td>1.05sec</td>
<td>1.8wks</td>
</tr>
<tr>
<td>$n^3$</td>
<td>16.8sec</td>
<td>17.9min</td>
<td>36,559yrs</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$3.7 \times 10^{63}$ yrs</td>
<td>$5.7 \times 10^{204}$ yrs</td>
<td>$2.1 \times 10^{315639}$ yrs</td>
</tr>
</tbody>
</table>

## Largest problem that can be solved if Time $\leq T$ at 1μsec per step

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>T=1min</th>
<th>T=1hr</th>
<th>T=1wk</th>
<th>T=1yr</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$6 \times 10^7$</td>
<td>$3.6 \times 10^9$</td>
<td>$6 \times 10^{11}$</td>
<td>$3.2 \times 10^{13}$</td>
</tr>
<tr>
<td>$n \log n$</td>
<td>$2.8 \times 10^6$</td>
<td>$1.3 \times 10^8$</td>
<td>$1.8 \times 10^{10}$</td>
<td>$8 \times 10^{11}$</td>
</tr>
<tr>
<td>$n^2$</td>
<td>$7.8 \times 10^3$</td>
<td>$6 \times 10^4$</td>
<td>$7.8 \times 10^5$</td>
<td>$5.6 \times 10^6$</td>
</tr>
<tr>
<td>$n^3$</td>
<td>$3.9 \times 10^2$</td>
<td>$1.5 \times 10^3$</td>
<td>$8.5 \times 10^3$</td>
<td>$3.2 \times 10^4$</td>
</tr>
<tr>
<td>$2^n$</td>
<td>25</td>
<td>31</td>
<td>39</td>
<td>44</td>
</tr>
</tbody>
</table>
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

$|f(n)| \leq k|g(n)|$ for all $n \geq n_0$

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