Runtime Analysis

Big O, Θ, some simple sums.

(see section 1.2 for motivation)

Notes, examples and code adapted from Data Structures and Other Objects Using C++ by Main & Savitch
Sums - review

• Some quick identities (just like integrals):

\[ \sum_{i} cf(i) = c \sum_{i} f(i) \text{, where } c \text{ is constant} \]

\[ \sum_{i} (f(i) + g(i)) = \sum_{i} f(i) + \sum_{i} g(i) \]

• Not like the integral:

\[ \sum_{i=b}^{a} f(i) = \sum_{i=a}^{b} f(i) \]
Closed form for simple sums

• A couple easy ones you really should stick in your head:

\[ \sum_{i=a}^{b} c = c + c + \ldots + c = (|b - a| + 1)c, \text{ where } c \text{ is constant} \]

Remember, +1 for the mule you're sitting on

\[ \sum_{i=1}^{m} i = 1 + 2 + 3 + \ldots + m = \frac{m(m+1)}{2} \]

(Thank Prof. Gauss for that one.)
Example (motivation), sect 1.2

- **Problem:** to count the # of steps in the Eiffel Tower
- **Setup:** Jack and Jill are at the top of the tower w/paper and a pencil
- Let $n \equiv \# \text{ of stairs to climb the Eiffel Tower}$
Eiffel Tower, alg. 1

• 1st attempt:
  – Jack takes the paper and pencil, walks stairs
  – Makes mark for each step
  – Returns, shows Jill

• Runtime:
  – # steps Jack took: 2n
  – # marks: n
  – So,
    \[ T_1(n) = 2n + n = 3n \]
Eiffel Tower, alg. 2

2\textsuperscript{nd} plan:
- Jill doesn’t trust Jack, keeps paper and pencil
- Jack descends first step
- Marks step w/hat
- Returns to Jill, tells her to make a mark
- Finds hat, moves it down a step
- etc.
Eiffel Tower, alg. 2 (cont.)

• Analysis:
  – # marks = n
  – # steps = 2 ( 1+2+3+…+n )
    = 2 * n(n+1)/2
    = n^2 + n
  – So,
    \[ T_2(n) = n^2 + 2n \]
Eiffel Tower, alg. 3

- 3rd try:
  - Jack & Jill see their friend Steve on the ground
  - Steve points to a sign
  - Jill copies the number down

- Runtime:
  - # steps Jack took: 0
  - # marks: $\log_{10}(n)$
  - So,
    $$T_3(n) = \log_{10}(n)$$
Comparison of algorithms

- Cost of algorithm \( T(n) \) vs. number of inputs, \( n \), (the number of stairs).
Wrap-up

• $T_1(n)$ is a line
  – So, if we double the # of stairs, the runtime doubles

• $T_2(n)$ is a parabola
  – So, if we double the # of stairs, the runtime quadruples (roughly)

• $T_3(n)$ is a logarithm
  – We’d have to multiply the number of stairs by a factor of 10 to increase T by 1 (roughly)
  – Very nice function
Some “what ifs”

- Suppose 3 stairs (or 3 marks) can be made per 1 second
- (There are really 2689 steps)

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>2n</th>
<th>3n</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1(n)$</td>
<td>45 min</td>
<td>90 min</td>
<td>135 min</td>
</tr>
<tr>
<td>$T_2(n)$</td>
<td>28 days</td>
<td>112 days</td>
<td>252 days</td>
</tr>
<tr>
<td>$T_3(n)$</td>
<td>2 sec</td>
<td>2 sec</td>
<td>2 sec</td>
</tr>
</tbody>
</table>
More observations

• While the relative times for a given n are a little sobering, what is of larger importance (when evaluating algorithms) is how each function grows
  – $T_3$ apparently doesn’t grow, or grows very slowly
  – $T_1$ grows linearly
  – $T_2$ grows quadratically
Asymptotic behavior

- When evaluating algorithms (from a design point of view) we don’t want to concern ourselves with lower-level details:
  - processor speed
  - the presence of a floating-point coprocessor
  - the phase of the moon
- We are concerned simply with how a function grows as n grows \textit{arbitrarily large}
- I.e., we are interested in its \textit{asymptotic behavior}
Asymptotic behavior (cont.)

• As \( n \) gets large, the function is dominated more and more by its highest-order term (so we don’t really need to consider lower-order terms)

• The coefficient of the leading term is not really of interest either. A line w/a steep slope will eventually be overtaken by even the laziest of parabolas (concave up). That coefficient is just a matter of scale. Irrelevant as \( n \to \infty \)
Big-O, $\Theta$

- A formal way to present the ideas of the previous slide:
  \[ T(n) = O(f(n)) \text{ iff there exist constants } k, n_0 \text{ such that:} \]
  \[ k \cdot f(n) > T(n) \]
  for all $n > n_0$

- In other words, $T(n)$ is bound above by $f(n)$. I.e., $f(n)$ gets on top of $T(n)$ at some point, and stays there as $n \to \infty$

- So, $T(n)$ grows no faster than $f(n)$
Further, if $f(n) = \Theta(T(n))$, then $T(n)$ grows no slower than $f(n)$.

We can then say that $T(n) = \Theta(n)$.

I.e., $T(n)$ can be bound both above and below with $f(n)$.
Setup

• First we have to decide what it is we’re counting, what might vary over the algorithm, and what actions are constant

• Consider:
  ```c
  for( i=0; i<5; ++i )
      ++cnt;
  ```
  – i=0 happens exactly once
  – ++i happens 5 times
  – i<5 happens 6 times
  – ++cnt happens 5 times
• If \( i \) and \( \text{cnt} \) are ints, then assignment, addition, and comparison is constant (exactly 32 bits are examined)

• So, \( i=0 \), \( i<5 \), and \( ++i \) each take some constant time (though probably different)

• We may, for purposes of asymptotic analysis, ignore the overhead:
  – the single \( i=0 \)
  – the extra \( i<5 \)

and consider the cost of executing the loop a single time
Setup (cont.)

- We decide that `++cnt` is a constant-time operation (integer addition, then integer assignment)
- So, a single execution of the loop is done in constant time
- Let this cost be \( c \):
Setup (cont.)

- So, the total cost of executing that loop can be given by:

\[ T(n) = \sum_{i=0}^{4} c = 5c \]

- Constant time

- Makes sense. Loop runs a set number of times, and each loop has a constant cost
Setup (cont.)

• $T(n) = 5c$, where $c$ is constant
• We say $T(n) = O(1)$
• From the definition of Big-O, let $k = 6c$:
  \[ 6c(1) > 5c \]
• This is true everywhere, for all $n$, so, for all $n > 0$, certainly. Easy
Eg 2

- Consider this loop:

```c
for( i=0; i<n; ++i )
    ++cnt;
```

- Almost just like last one:

\[
T(n) = \sum_{i=0}^{n-1} c = cn
\]
Eg 2 (cont.)

• Now, $T(n)$ is linear in $n$
• $T(n) = O(n)$
• This means we can multiply $n$ by something to get bigger than $T(n)$:
  \[ n > cn, \text{ let } k = 2c \]
  \[ 2cn > cn \]
• This isn’t true everywhere. We want to know that it becomes true somewhere and stays true as $n \to \infty$
• Solve the inequality:
  
  \[ 2cn > cn \]
  
  \[ cn > 0 \]
  
  \[ n > 0 \] (c is strictly positive, so, not 0)

• \( cn \) gets above \( T(n) \) at \( n=0 \) and stays there as \( n \) gets large

• So, \( T(n) \) grows no faster than a line
Eg. 3

• Let’s make the previous example a little more interesting:
  • Say $T(n) = 2cn + 13c$
  • $T(n) = O(n)$
  • So, find some $k$ such that $kn > 2cn + 13c$ (past some $n_0$)
  • Let $k = 3c$. $\Rightarrow$
    $3cn > 2cn + 13c$
• Find values of $n$ for which the inequality is true:
  \[ 3cn > 2cn + 13c \]
  \[ cn > 13c \]
  \[ n > 13 \]

• $3cn$ gets above $T(n)$ at $n=13$, and stays there.

• $T(n)$ grows no faster than a line
Eg 4

- Nested loops:
  
  ```c
  for( i=0; i<n; ++i )
  for( j=1; j<=n; ++j )
  ++cnt;
  ```

- Runtime given by:

  \[
  T(n) = \sum_{i=0}^{n-1} \left( \sum_{j=1}^{n} c \right) = \sum_{i=0}^{n-1} c n = cn^2
  \]
Eg. 4 (cont.)

• Claim: $T(n) = O(n^2)$
  $\Rightarrow$ there exists a constant $k$ such that
  $kn^2 > cn^2$, let $k = 2c$:
  $2cn^2 > cn^2$

• Where is this true?
  $cn^2 > 0$
  $n^2 > 0$
  $n > 0$
Eg. 4 (cont.)

- $2cn^2$ gets above $cn^2$ at $n=0$ and stays there
- $T(n)$ is bound above by a parabola
- $T(n)$ grows no faster than a parabola
Eg. 5

• Let’s say
  \[ T(n) = 2cn^2 + 2cn + 3c \]
• Claim: \( T(n) = O(n^2) \)
• We just need to choose a \( k \) larger than the coefficient of the leading term.

• Let \( k = 3c \)

\[ 3cn^2 > 2cn^2 + 2cn + 3c \]

• Where? (Gather like terms, move to one side)

\[ cn^2 - 2cn - 3c > 0 \]
Eg. 5 (cont.)

- This one could be solved directly, but it is usually easier to find the roots of the parabola on the left (which would be where our original 2 parabolas intersected).
- This happens at \( n=-1 \) and \( n=3 \).
- So, plug something like \( n=4 \) into our original inequality. Is it true?
- Then it’s true for all \( n>3 \) (since we know they don’t intersect again).
Eg. 5 (cont.)

- $3cn^2$ gets above $T(n)$ at $n=3$ and stays there
- $T(n)$ grows no faster than a parabola
- $T(n)$ can be bound above by a parabola