Announcements

- Homework requires you get the textbook (Either 2nd or 3rd Edition)
- Section Thursday
- Homework #1 out today (Wednesday)
  - Due at the beginning of class next Wednesday (Apr 6).
- Program Assignment #1 is available
  - Get environment set up and compile the program by Thursday

Measuring performance

bool LinearArrayContains(int array[], int n, int key)
{
  for( int i = 0; i < n; i++ ) {
    if( array[i] == key )
      // Found it!
      return true;
    }
  return false;
}

Linear Search Analysis

<table>
<thead>
<tr>
<th>Best Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>37</td>
<td>50</td>
</tr>
<tr>
<td>73</td>
<td>75</td>
</tr>
</tbody>
</table>

Binary Search Analysis

bool BinArrayContains(int array[], int low, int high, int key)
{
  // The subarray is empty
  if( low > high ) return false;
  // Search this subarray recursively
  int mid = (high + low) / 2;
  if( key == array[mid] ) {
    return true;
  } else if( key < array[mid] ) {
    return BinArrayFind( array, low, mid - 1, key );
  } else {
    return BinArrayFind( array, mid + 1, high, key );
  }
}

Solving Recurrence Relations

1. Determine the recurrence relation and base case(s).
2. “Expand” the original relation to find an equivalent expression in terms of the number of expansions (k).
3. Find a closed-form expression by setting k to a value which reduces the problem to a base case.
Linear Search vs Binary Search

<table>
<thead>
<tr>
<th></th>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Case</td>
<td>4</td>
<td>5 at [middle]</td>
</tr>
<tr>
<td>Worst Case</td>
<td>3n+3</td>
<td>7 ( \lfloor \log n \rfloor + 9 )</td>
</tr>
</tbody>
</table>

Empirical comparison

Gives additional information

Asymptotic Analysis

- Consider only the order of the running time
  - A valuable tool when the input gets “large”
  - Ignores the effects of different machines or different implementations of same algorithm

Asymptotic Analysis

- To find the asymptotic runtime, throw away the constants and low-order terms
  - Linear search is \( T_{LS}^{worst}(n) = 3n + 3 \in O(n) \)
  - Binary search is \( T_{BS}^{worst}(n) = 7 \lfloor \log_2 n \rfloor + 9 \in O(\log n) \)

Remember: the “fastest” algorithm has the slowest growing function for its runtime

Asymptotic Analysis

- Eliminate low order terms
  - 4n + 5 \( \Rightarrow \) 4n
  - 0.5 n log n + 2n + 7 \( \Rightarrow \) 0.5 n log n
  - \( n^3 + 3 \cdot 2^n + 8n \) \( \Rightarrow \) \( n^3 \cdot 2^n \)

- Eliminate coefficients
  - 4n \( \Rightarrow \) 4n
  - 0.5 n log n \( \Rightarrow \) 0.5 n log n
  - 3 \( 2^n \) \( \Rightarrow \) 3 \( 2^n \)

Properties of Logs

- Basic:
  - \( A \cdot 2^B = B \)
  - \( \log_b A = \)

- Independent of base:
  - \( \log(AB) = \)
  - \( \log(A/B) = \)
  - \( \log(A^B) = \)
  - \( \log((A^B)^C) = \)
Properties of Logs

Changing base → multiply by constant
  - For example: \( \log_{2}x = 3.22 \log_{10}x \)
  - More generally
    \[ \log_{A} n = \left( \frac{1}{\log_{A} B} \right) \log_{B} n \]
  - Means we can ignore the base for asymptotic analysis (since we’re ignoring constant multipliers)

Comparing functions

- \( f(n) \) is an upper bound for \( h(n) \)
  if \( h(n) \leq f(n) \) for all \( n \)

This is too strict – we mostly care about large \( n \)

Still too strict if we want to ignore scale factors

Definition of Order Notation

- \( h(n) \in O(f(n)) \) if there exist positive constants \( c \) and \( n_0 \)
  such that \( h(n) \leq c f(n) \) for all \( n \geq n_0 \)

\( O(f(n)) \) defines a class (set) of functions

Order Notation: Intuition

\[ a(n) = n^3 + 2n^2 \]
\[ b(n) = 100n^2 + 1000 \]

Although not yet apparent, as \( n \) gets “sufficiently large”, \( a(n) \) will be “greater than or equal to” \( b(n) \)

Order Notation: Example

\[ 100n^2 + 1000 \leq (n^3 + 2n^2) \text{ for all } n \geq 100 \]
So \( 100n^2 + 1000 \in O(n^3 + 2n^2) \)
Example

\[ h(n) \in O(f(n)) \] if and only if there exist positive constants \( c \) and \( n_0 \) such that:

\[ h(n) \leq c f(n) \] for all \( n \geq n_0 \)

Example:

\[ 100n^2 + 1000 \leq 1(n^3 + 2n^2) \] for all \( n \geq 100 \)

So \( 100n^2 + 1000 \in O(n^3 + 2n^2) \)

Constants are not unique

\[ h(n) \in O(f(n)) \] if and only if there exist positive constants \( c \) and \( n_0 \) such that:

\[ h(n) \leq c f(n) \] for all \( n \geq n_0 \)

Example:

\[ 100n^2 + 1000 \leq 1(n^3 + 2n^2) \] for all \( n \geq 100 \)

\[ 100n^2 + 1000 \leq \frac{1}{2}(n^3 + 2n^2) \] for all \( n \geq 198 \)

Another Example: Binary Search

\[ h(n) \in O(f(n)) \] if and only if there exist positive constants \( c \) and \( n_0 \) such that:

\[ h(n) \leq c f(n) \] for all \( n \geq n_0 \)

Is \( 7\log_2 n + 9 \in O(\log_2 n) \)?

Order Notation:

Worst Case Binary Search

Some Notes on Notation

Sometimes you'll see (e.g., in Weiss)

\[ h(n) = O(f(n)) \]

or

\[ h(n) \text{ is } O(f(n)) \]

These are equivalent to

\[ h(n) \in O(f(n)) \]

Big-O: Common Names

- Constant: \( O(1) \)
- Logarithmic: \( O(\log n) \) \( (\log_2 n, \log n^2 \subseteq O(\log n)) \)
- Linear: \( O(n) \)
- Log-linear: \( O(n \log n) \)
- Quadratic: \( O(n^2) \)
- Cubic: \( O(n^3) \)
- Polynomial: \( O(n^k) \) \( (k \text{ is a constant}) \)
- Exponential: \( O(c^n) \) \( (c \text{ is a constant } > 1) \)
Asymptotic Lower Bounds

• \( \Omega(g(n)) \) is the set of all functions asymptotically greater than or equal to \( g(n) \)

• \( h(n) \in \Omega(g(n)) \) iff
  - There exist \( c > 0 \) and \( n_0 > 0 \) such that \( h(n) \geq c \cdot g(n) \) for all \( n \geq n_0 \)

Asymptotic Tight Bound

• \( O(f(n)) \) is the set of all functions asymptotically equal to \( f(n) \)

• \( h(n) \in O(f(n)) \) iff
  - \( h(n) \leq c \cdot f(n) \) for all \( n \geq n_0 \)
  - This is equivalent to:
    \[
    \lim_{n \to \infty} \frac{h(n)}{f(n)} = 0
    \]

Full Set of Asymptotic Bounds

• \( O(f(n)) \) is the set of all functions asymptotically less than or equal to \( f(n) \)
  - \( o(f(n)) \) is the set of all functions asymptotically strictly less than \( f(n) \)

• \( \Omega(g(n)) \) is the set of all functions asymptotically greater than or equal to \( g(n) \)
  - \( \omega(g(n)) \) is the set of all functions asymptotically strictly greater than \( g(n) \)

• \( \Theta(f(n)) \) is the set of all functions asymptotically equal to \( f(n) \)

Asymptotic Notation | Mathematics Relation
--- | ---
O | \( \leq \)
\( \Omega \) | \( \geq \)
\( \Theta \) | \( = \)
O | \( < \)
\( \omega \) | >

Formal Definitions

• \( h(n) \in O(f(n)) \) iff
  - There exist \( c > 0 \) and \( n_0 > 0 \) such that \( h(n) \leq c \cdot f(n) \) for all \( n \geq n_0 \)
  - This is equivalent to:
    \[
    \lim_{n \to \infty} \frac{h(n)}{f(n)} = 0
    \]

• \( h(n) \in \Omega(f(n)) \) iff
  - \( h(n) \) is the set of all functions asymptotically greater than or equal to \( f(n) \)
  - This is equivalent to:
    \[
    \lim_{n \to \infty} \frac{h(n)}{f(n)} = \infty
    \]

Complexity cases (revisited)

Problem size \( N \)

- **Worst-case complexity:** \( \max \) # steps algorithm takes on “most challenging” input of size \( N \)
- **Best-case complexity:** \( \min \) # steps algorithm takes on “easiest” input of size \( N \)
- **Average-case complexity:** \( \text{avg} \) # steps algorithm takes on random inputs of size \( N \)
- **Amortized complexity:** \( \text{max} \) total # steps algorithm takes on \( M \) ”most challenging” consecutive inputs of size \( N \), divided by \( M \) (i.e., divide the max total by \( M \)).
Bounds vs. Cases

Two orthogonal axes:

- **Bound Flavor**
  - Upper bound \( O, o \)
  - Lower bound \( \Omega, \omega \)
  - Asymptotically tight \( \Theta \)

- **Analysis Case**
  - Worst Case (Adversary), \( T_{\text{worst}}(n) \)
  - Average Case, \( T_{\text{avg}}(n) \)
  - Best Case, \( T_{\text{best}}(n) \)
  - Amortized, \( T_{\text{amort}}(n) \)

One can estimate the bounds for any given case.