2: Asymptotic Analysis

CSE326 Spring 2002

April 1, 2002

--- Linked List Search ---

![Linked List Diagram]

```c
bool ListFind(int k, Node *node)
{
    while (node)
    {
        if (node->key == k)
            return true;
        n = n->next;
    }
    return false;
}
```

--- Analyzing Algorithms ---

- Best case:

```c
bool ListFind(int k, Node *node)
{
    while (node)
    {
        if (node->key == k)
            return true;
        n = n->next;
    }
    return false;
}
```

- Worst case:

- Most of the time:
Array Search

7 6 4 9 2 1 3 6 7 5
array; n=10

bool ArrayFind(int k, int *array, int n)
{

}

Analyzing Algorithms

- Best case:

  bool ArrayFind(int k, int *key_array, int n)
  {
    for (int i = 0; i < n; i++)
      if (key_array[i] == k)
        return true;
    return false;
  }

- Worst case:

- Most of the time:

  bool ArrayFind(int k, int *array, int n)
  {
    for (int i = 0; i < n; i++)
      if (array[i] == k)
        return true;
    return false;
  }

Binary Search

7 6 4 9 2 1 3 6 7 5

bool BinarySearch(int k, int *array, int high, int low=0)
{
  assert(low >= 0);
  if (low >= high)
    return false;
  int mid = (high-low)/2;
  if (k == array[mid])
    return true;
  else if (k < array[mid])
    return BinarySearch(k, array, mid, low);
  // k > array[mid]
  return BinarySearch(k, array, high, mid+1);
}
bool BinarySearch(int k,  
int = array,  
int high,  
int low=0)  
{  
if (low >= high)  
    return false;  
int mid = (high−low)/2;  
if (k == array[mid])  
    return true;  
else if (k < array[mid])  
    return BinarySearch(k, array, mid, low);  
// k > array[mid]  
return BinarySearch(k, array, high, mid+1);  
}  

● Best case:  

● Worst case:  

Solving the Recurrence Relation  

Analysis Summary  

<table>
<thead>
<tr>
<th>List (linear search)</th>
<th>Array</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worst Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Most of the Time</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Criteria for choosing an algorithm:

- Speed of execution
- Ease of coding
- Preprocessing required

Speed is usually most important—easiest to quantify.
Best was to compare speeds is to measure and then graph.
Fast Computer vs. Slow Computer

With same algorithm, the faster computer always wins.

Fast Computer vs. Smart Programmer I

Linear search beats binary search?

Fast Computer vs. Smart Programmer II

Binary search always wins—eventually.
Asymptotic Analysis

- Asymptotic analysis looks at the order of the running-time of an algorithm.
  - What happens when the input gets large?
  - Ignore effects of different machines.
- Linear search is $O(n)$ (whether on a list or an array!).
- Binary search is $O(\log n)$.

Order Notation: Intuition

Suppose we have 3 algorithms running at the following speeds:

- $100n \log n + 5000n$
- $10n^2 + 1000n$
- $15n^2 + 2n$

Which algorithm is fastest?

$\iff$ Which function grows slowest?
Order Notation: Intuition

$15n^2 + 2n < 10n^2 + 1000n < 100n \log n + 5000x$

Order Notation: Intuition

$O(n \log n)$ vs. $O(n^2)$ is what matters

Order Notation

Precise Definition

- $O(f(n))$ is a set of functions
- $g(n) \in O(f(n))$ when
  
  There exist $c$ and $n_0$ such that
  
  $g(n) \leq c \cdot f(n)$, for all $n \geq n_0$. 

Order Notation: Example

- $10n^2 + 1000n \leq 1 \cdot (15n^2 + 2n)$ for $n \geq 250$,
  $\Rightarrow 10n^2 + 1000n \in \mathcal{O}(15n^2 + 2n)$

- $10n^2 + 1000n$ vs. $n^2$?

Order Notation: Definition

$10n^2 + 1000n \leq 11 \cdot n^2$ for $n \geq 1200$,
$\Rightarrow 10n^2 + 1000n \in \mathcal{O}(n^2)$

Order Notation

$O(n^2)$

$O(15n^2 + 2n) = O(n^2)$
Order Notation

\[ O(n^2) \]

\[ O(n \log n) \]

\[ O(n \log n) \subset O(n^2) \]

Hierarchy of Orders

\[
\begin{align*}
O(2^n) & \quad \text{for constant } k > 0 \\
O(n^k) & \quad \text{for constant } k \\
O(n^2) & \\
O(n^{1+\epsilon}) & \quad \text{for constant } \epsilon > 0 \\
O(n \log n) & \\
O(n^\epsilon) & \quad \text{for constant } \epsilon > 0 \\
O(\log^k n) & \quad \text{for constant } k \\
O(\log n) & \\
O(\log \log n) & \\
O(1) & 
\end{align*}
\]

\[ \Omega(f(n)) \] is all functions asymptotically less than or equal to \( f(n) \).

"Big Omega"
- \( 4n^2 \in \Omega(n^2) \)
- \( \log n \in \Omega(n^2) \)
- \( n^3 \notin \Omega(n^2) \)

\[ \Theta(f(n)) \] is all functions asymptotically equal to \( f(n) \).

"Big Theta"
- \( 4n^2 \in \Theta(n^2) \)
- \( \log n \notin \Theta(n^2) \)
- \( n^3 \notin \Theta(n^2) \)
### Menagerie of Symbols

<table>
<thead>
<tr>
<th>Asymptotic Notation</th>
<th>Mathematics</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O}$</td>
<td></td>
<td>$\leq$</td>
</tr>
<tr>
<td>$\Omega$</td>
<td></td>
<td>$\geq$</td>
</tr>
<tr>
<td>$\Theta$</td>
<td></td>
<td>$\approx$</td>
</tr>
<tr>
<td>$\omega$</td>
<td></td>
<td>$\ll$</td>
</tr>
</tbody>
</table>

### Rules of Thumb

- **Polynomials**
  
  Take term with highest power, drop coefficient.

  \[
  45n^4 + 20n^2 + 45n + 60 \in \mathcal{O}(n^4) \\
  34n^2 + 16n^{0.1} + 784 \log n + 2 \in \mathcal{O}(n^2)
  \]

- **Logarithms**
  
  Take log-term with highest power, drop coefficients and powers in argument.

  \[
  50 \log n^{300} \in \mathcal{O}(\log n) \\
  32 \log^2 n + \log n^4 + \log \log n^2 \in \mathcal{O}(\log^2 n)
  \]