ECE 242
Data Structures and Algorithms

http://www.ecs.umass.edu/~polizzi/Teaching/ECE242/

Arrays

Lecture 2

Prof. Eric Polizzi
Array: most commonly used data storage structure

Collection of primitives variables (int, double, etc.) or objects with fixed size.

- Example: with generic type
  \[\text{type[]} \text{ array } = \text{ new type}[\text{MaxItem}];\]

- Number of active items \(N \leq \text{MaxItem}\)
  - Example: \(\text{MaxItem}=10; \ N=5\)

- Basic operations on arrays: insert (initialization), permute, shift, linear search, delete, etc

- Higher level interfaces for Arrays via Object Oriented Programming

\[\text{HighArray} \text{ array } = \text{ new HighArray}(\text{MaxItem});\]

- Class handles index numbers and number of current items privately

- Ordered arrays allow the use of very fast binary search algorithm
**Binary Search**

- **Analogy:** The **guess-a-number** game!
  - Choose a number between 1-100

- Ok Let me guess....
  - (1-100) 50? ........................... nope too high
  - (1-49) 25? ........................... nope too low
  - (26-49) 37? ........................... nope too high
  - (26-36) 31? ........................... nope too low
  - (32-36) 34? ........................... nope too high
  - (32-33) 32? ........................... nope too low
  - (33-33) 33? ............................Correct!
public int find(long searchKey) {
    int lowerBound = 0;
    int upperBound = nElems - 1;
    int curIn;
    while (true) {
        curIn = (lowerBound + upperBound) / 2;
        if (a[curIn] == searchKey)
            return curIn; // found it
        else if (lowerBound > upperBound)
            return nElems; // can't find it
        else // divide range
            { // end else divide range
                if (a[curIn] < searchKey)
                    lowerBound = curIn + 1; // in upper half
                else
                    upperBound = curIn - 1; // in lower half
            } // end else divide range
    } // end while
} // end find()
Binary Search: Number of steps?

- In the worst case scenario, the algorithm progresses until one item is left in the search range.
- In practice, at each step the range of values is approximately divided by 2 (+ or – 1 item).
- In theory, we can consider that the number of items \( N \) keeps being exactly divided by 2 until approximately one item is left.
  - Example \( N=100 \)

<table>
<thead>
<tr>
<th>Step s</th>
<th>(N) ?</th>
<th>Number of items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100/2 = 50.0000 ((N/2^1))</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>50/2 = 25.0000 ((N/2^2))</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>25/2 = 12.5000 ((N/2^3))</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>12.5/2 = 6.25000 ((N/2^4))</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>6.25/2 = 3.12500 ((N/2^5))</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3.125/2 = 1.56250 ((N/2^6))</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1.5625/2 = 0.78125 ((N/2^7))</td>
<td>1</td>
</tr>
</tbody>
</table>
Mathematically, the number of steps $s$ needed must satisfy:

$$\frac{N}{2^s} = 1$$

**Problem:** $N$ is given, find $s$ such that

- $\log(N) = s \log(2)$
- $\log=\ln=\log_e$ is the **natural log** $\log(e)=1$ [log base e]

$$s = \frac{\log(N)}{\log(2)} = \log_2(N)$$

[log base 2] $\log_2(2)=1$
## Linear vs Binary Search

**Linear search**

\[ s = \frac{N}{2} \]

**Binary search**

\[ s = \log_2(N) \]

<table>
<thead>
<tr>
<th>( N )</th>
<th>Linear search ( s = \frac{N}{2} )</th>
<th>( \log_2(N) )</th>
<th>Binary search ( s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>3.32</td>
<td>4</td>
</tr>
<tr>
<td>100</td>
<td>50</td>
<td>6.64</td>
<td>7</td>
</tr>
<tr>
<td>1,000</td>
<td>500</td>
<td>9.97</td>
<td>10</td>
</tr>
<tr>
<td>10,000</td>
<td>5,000</td>
<td>13.3</td>
<td>14</td>
</tr>
<tr>
<td>100,000</td>
<td>50,000</td>
<td>16.6</td>
<td>17</td>
</tr>
<tr>
<td>1,000,000</td>
<td>500,000</td>
<td>19.9</td>
<td>20</td>
</tr>
<tr>
<td>10,000,000</td>
<td>5,000,000</td>
<td>23.3</td>
<td>24</td>
</tr>
<tr>
<td>100,000,000</td>
<td>50,000,000</td>
<td>26.6</td>
<td>27</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>500,000,000</td>
<td>29.9</td>
<td>30</td>
</tr>
</tbody>
</table>

Logarithm growth rate \( \ll \) Linear growth rate
Comments on Running Time

**Question:** How efficient an Algorithm is?

**Some ”not-so meaningful” statements**
- This algorithm converges in 10 seconds
  ...don't computers have different speed?
- Algorithm A is twice faster than B
  ...would it still be the case if the number of items N changes?

**Solutions:**

**Empirical approach**
Implement all algorithms and perform all the testings/comparisons
...inefficient strategy

**Analytical approach**
Analysis of the algorithm complexity to understand its capabilities
...knowing in advance if an algorithm is worth implementing
Comments on Running Time

- Time to insert in an unordered array
  - $T = K$ (seconds)
  - $K$ is a constant related to speed of processor, compiler efficiency, etc.
  - to perform the single operation of insertion

- Linear search
  - $T = K \times \frac{N}{2}$ (average number of comparisons)
  - $K/2$ is a constant (the factor 2 can be lumped into the $K$)
  - It comes $T = K \times N$

- Binary Search
  - $T = K \times \log_2(N)$
  - $K$ is a constant (the factor $\log(2)$ can be lumped into the $K$)
  - It comes $T = K \times \log(N)$
Big O Notation

- All that really matter is how the running time $T$ is affected the number of items $N$ (the constant $K$ is not meaningful)
- The Big O notation introduces the letter $O$ which means “order of”
  - Insertion in unordered array $O(1)$ (it means constant)
  - Linear search $O(N)$
  - Binary search $O(\log N)$
- Cost: $O(1) < O(\log N) < O(N)$ (excellent > good > fair)

**TABLE 2.5** Running Times in Big O Notation

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Running Time in Big O Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear search</td>
<td>• $O(N)$</td>
</tr>
<tr>
<td>Binary search</td>
<td>• $O(\log N)$</td>
</tr>
<tr>
<td>Insertion in unordered array</td>
<td>• $O(1)$</td>
</tr>
<tr>
<td>Insertion in ordered array</td>
<td>• $O(N)$</td>
</tr>
<tr>
<td>Deletion in unordered array</td>
<td>• $O(N)$</td>
</tr>
<tr>
<td>Deletion in ordered array</td>
<td>• $O(N)$</td>
</tr>
</tbody>
</table>
Big O Notation

The graph illustrates different complexities using Big O Notation, showing how the number of steps grows with the number of items (N). It includes:

- $O(1)$: Constant time complexity
- $O(N)$: Linear time complexity
- $O(\log N)$: Logarithmic time complexity
- $O(N^2)$: Quadratic time complexity