Search of an Ordered List

Problem: To search an ordered list for a given item.

Formulation: Given a list of $n$ items:

$$a[1], a[2], \ldots, a[n]$$

arranged in ascending order

$$a[1] < a[2] < \cdots < a[n]$$

($a[i]$’s may any type of item, so long as they can be consistently ordered, e.g. numbers, words ordered alphabetically etc.)
For an item $x$ determine whether $x$ belongs to the list and if so find the index or key $i$ such that $a[i] = x$.

**Example**

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td>Ann</td>
<td>Dawn</td>
<td>Erik</td>
<td>Gail</td>
<td>Juan</td>
</tr>
<tr>
<td>$i$</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>$a[i]$</td>
<td>Matt</td>
<td>Max</td>
<td>Rita</td>
<td>Tsuji</td>
<td>Yuen</td>
</tr>
</tbody>
</table>

If $x = \text{Max}$, then a search should return the index $i = 7$.

**Sequential Search**

**Algorithm**: Begin at the start of the list and continue item by item until a match is attained.

More precisely:

1. Begin at index $i = 1$ and compare $a[1]$ to the required item $x$. If it matches we are done.
2. Move to index $i = 2$ and compare $a[2]$ to $x$.
3. Continue until an index $i$ is found such that $a[i] = x$ or we find that $a[i] > x$ without a match, in which case we can be sure that the item $x$ is not in the list.
Example

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td>Ann</td>
<td>Dawn</td>
<td>Erik</td>
<td>Gail</td>
<td>Juan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td>Matt</td>
<td>Max</td>
<td>Rita</td>
<td>Tsuji</td>
<td>Yuen</td>
</tr>
</tbody>
</table>

A search for $x = \text{Max}$ returns index $i = 7$ and requires 7 comparisons.

A search for $x = \text{Gary}$ fails and requires 5 comparisons – we need to get to $a[5] = \text{Juan}$ to be sure Gary is not in the list.

Worst Case Analysis

Let $S(n)$ be the number of comparisons needed to complete the sequential search of a list of $n$ items in the worst case.

The worst case occurs when the required item $x$ is the last item in the list or, when the item is not in the list, it is greater than the second last item.

In either case $n$ comparisons are required, thus

$$S(n) = n$$
Digression – Floor and Ceiling Functions

1. The **floor** of a real number $x$, denoted by $\lfloor x \rfloor$ is the greatest integer not exceeding $x$, e.g. $\lfloor 3.2 \rfloor = 3$ and $\lfloor -4.5 \rfloor = -5$.

2. The **ceiling** of a real number $x$, denoted by $\lceil x \rceil$ is the smallest integer not less than $x$, e.g. $\lceil 3.2 \rceil = 4$ and $\lceil -4.5 \rceil = -4$.

Binary Search

To search a list of $n$ items for item $x$:

1. Look at the middle item of the list $a[n/2]$. If it matches we are done.

2. Otherwise we know which half of the list, either $\{a[i] \mid i < n/2\}$ or $\{a[i] \mid i > n/2\}$ the item $x$ lies in.

3. Look at the middle item of the appropriate sublist. Again, if it matches we are done. Otherwise we know in which half of the list to look next.

4. Repeat this process until the item $x$ is found.
Example

If at some stage of binary search we have a sublist $a[j], \ldots, a[k]$ we take the middle index to be $[(j+k)/2]$. This removes an ambiguity when a list or sublist has an even number of items.

Let us apply binary search to finding Max in the list:

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td>Ann</td>
<td>Dawn</td>
<td>Erik</td>
<td>Gail</td>
<td>Juan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td>Matt</td>
<td>Max</td>
<td>Rita</td>
<td>Tsuji</td>
<td>Yuen</td>
</tr>
</tbody>
</table>

1. We choose the middle index $i = \lfloor (1+10)/2 \rfloor = 5$. $a[5] = Juan$.

   Ann Dawn Erik Gail Juan Matt Max Rita Tsuji Yuen

   Juan is less than Max, so we move to the sublist $a[6], \ldots, a[10]$.

   Ann Dawn Erik Gail Juan Matt Max Rita Tsuji Yuen

2. The middle index is now $i = \lfloor (6+10)/2 \rfloor = 8$. $a[8] = Rita$.

   Ann Dawn Erik Gail Juan Matt Max Rita Tsuji Yuen
Rita is greater than Max so we move to the sublist \( a[6], \ldots, a[7] \).

Ann Dawn Erik Gail Juan \underline{Matt} Max Rita Tsuji Yuen

3. The middle index is now \( i = \lfloor (6 + 7)/2 \rfloor = 6 \). \( a[6] = \) Matt.

Ann Dawn Erik Gail Juan \underline{Matt} Max Rita Tsuji Yuen

Matt is less than Max so we move to the sublist \( a[7], \ldots, a[7] \).

Ann Dawn Erik Gail Juan Matt \underline{Max} Rita Tsuji Yuen

4. We now have a one element list and \( a[7] = \) Max so we are done.

We needed 4 comparisons to complete the search.
Now let us search for Gary in the same list:

1. We choose the middle index \( i = \lfloor (1 + 10)/2 \rfloor = 5 \). \( a[5] = \text{Juan} \).

   Ann Dawn Erik Gail \underline{Juan} Matt Max Rita Tsuji Yuen

   Juan is greater than Gary, so we move to the sublist \( a[1], \ldots, a[4] \).

2. The middle index is now \( i = \lfloor (1 + 4)/2 \rfloor = 2 \). \( a[2] = \text{Dawn} \).

   Ann \underline{Dawn} Erik Gail Juan Matt Max Rita Tsuji Yuen

   Dawn is less than Gary so we move to the sublist \( a[3], \ldots, a[4] \).

   Ann Dawn \underline{Erik Gail} Juan Matt Max Rita Tsuji Yuen

3. The middle index is now \( i = \lfloor (3 + 4)/2 \rfloor = 3 \). \( a[3] = \text{Erik} \).

   Ann Dawn \underline{Erik} \underline{Gail} Juan Matt Max Rita Tsuji Yuen

   Erik is less than Gary so we move to the sublist \( a[4], \ldots, a[4] \).

   Ann Dawn Erik \underline{Gail} Juan Matt Max Rita Tsuji Yuen
4. We now have a one element list and \( a[4] = \text{Gail} \) so we can assert that Gary is not in the list.

We needed 4 comparisons to complete the search.

---

**Analysis of the Algorithm**

This analysis has some similarities to the analysis of bubble sort in Tutorial 1.

Let \( B(n) \) be the number of comparisons needed to complete a binary search of a list of \( n \) items in the **worst case**.

At all but the final step of binary search we perform a comparison and then move to a sublist.

1. \( B(1) = 1 \). Even for a one element list we need to check that the item matches \( x \).
2. When \( n > 1 \) we perform a comparison and then search either the left or right sublist.

3. When \( n \) is odd, both the right and left sublists have \((n - 1)/2\) items. Thus
   \[
   B(n) = 1 + B((n - 1)/2)
   \]

4. When \( n \) is even, the left sublist has \( n/2 - 1 \) items and the right sublist has \( n/2 \) items. The right sublist is longer, so
   \[
   B(n) = 1 + B(n/2)
   \]

---

\[
\begin{align*}
B(1) &= 1 \\
B(2) &= 1 + B(2/2) = 1 + B(1) = 2 \\
B(3) &= 1 + B((3 - 1)/2) = 1 + B(1) = 2 \\
B(4) &= 1 + B(4/2) = 1 + B(2) = 3 \\
B(5) &= 1 + B((5 - 1)/2) = 1 + B(2) = 3 \\
B(6) &= 1 + B(6/2) = 1 + B(3) = 3 \\
B(7) &= 1 + B((7 - 1)/2) = 1 + B(3) = 3 \\
B(8) &= 1 + B(8/2) = 1 + B(4) = 4 \\
B(9) &= 1 + B((9 - 1)/2) = 1 + B(4) = 4
\end{align*}
\]
Slide 19

\[
B(10) = 1 + B(10/2) = 1 + B(5) = 4 \\
= 4 \\
B(15) = 1 + B((15 - 1)/2) = 1 + B(7) = 4 \\
B(16) = 1 + B(16/2) = 1 + B(8) = 5 \\
B(100) = 1 + B(50) = 2 + B(25) = 3 + B(12) \\
= 7 \\
B(1000) = 1 + B(500) = 2 + B(250) = 3 + B(125) \\
= 4 + B(62) = 5 + B(31) = 6 + B(15) \\
= 10
\]

Slide 20

Conclusions:
1. \( B(n) = B(n - 1) \) except when \( n \) is a power of 2.
2. \( B(n) = 1 + B(n - 1) \) when \( n \) is a power of 2.
3. \( B(2^m) = 1 + m \)
**Digression – Logarithms**

Logarithms are closely related to powers

\[ a^x = y \] is equivalent to \[ x = \log_a y \]

in other words, \( \log_a y \) – the logarithm to the base \( a \) of \( y \) is the power we need to raise \( a \) to to get \( y \).

Basic properties:

1. \( \log_a 1 = 0 \)
2. \( \log_a a = 1 \)
3. \( \log_a b^x = x \log_a b \)
4. \( \log_a(xy) = \log_a x + \log_a y \)

Most often the base \( a \) is 10 or \( e = 2.718 \ldots \) (natural logarithms. In computer science the base 2, \( \log_2 \), is most used.

Logarithms are slowly growing functions:

<table>
<thead>
<tr>
<th>( \log_2 n )</th>
<th>( n )</th>
<th>( n^2 )</th>
<th>( 2^n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3.32</td>
<td>10</td>
<td>100</td>
<td>1024</td>
</tr>
<tr>
<td>6.64</td>
<td>100</td>
<td>1000</td>
<td>( 1.3 \times 10^{30} )</td>
</tr>
<tr>
<td>9.97</td>
<td>1000</td>
<td>( 10^6 )</td>
<td>( 1.1 \times 10^{301} )</td>
</tr>
</tbody>
</table>

Note:

\[ |\log_2 x| < x \quad \text{for } x \geq 1 \]