Searching

Chapter 18

Supplementary Readings:
   http://comjnl.oxfordjournals.org/content/26/2/154.abstract
Objectives

• Search an array by using
  ▪ a sequential search
  ▪ a binary search

• Search a chain of linked nodes
  ▪ sequentially

• Describe time efficiency of search algorithms
## Contents

- **Search**: The Problem & Motivation
- Computation complexity of Linear and Binary Search
- Choosing a Search Method

<table>
<thead>
<tr>
<th></th>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Array</strong></td>
<td>Linear Search</td>
<td>Binary Search</td>
</tr>
<tr>
<td></td>
<td>- Iterative</td>
<td>Java: <code>binarySearch</code></td>
</tr>
<tr>
<td></td>
<td>- Recursive</td>
<td></td>
</tr>
<tr>
<td><strong>Chain of linked Nodes</strong></td>
<td>Linear Search</td>
<td>Linear Search</td>
</tr>
<tr>
<td></td>
<td>- Iterative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Recursive</td>
<td></td>
</tr>
</tbody>
</table>
Search: The Problem & Motivation

Figure 18-1 Searching is an everyday occurrence
Figure 18-2 An iterative sequential search of an array that (a) finds its target.

Ex. List the comparison when searching for 6.
Figure 18-2 An iterative sequential search of an array that (b) does not find its target

(b) A search for 6

Look at 9:

9 5 8 4 7

6 ≠ 9 so continue searching.

Look at 5:

9 5 8 4 7

6 ≠ 5 so continue searching.

Look at 8:

9 5 8 4 7

6 ≠ 8 so continue searching.

Look at 4:

9 5 8 4 7

6 ≠ 4 so continue searching.

Look at 7:

9 5 8 4 7

6 ≠ 7 so continue searching.

No entries are left to consider, so the search ends. 6 is not in the array.
Unsorted Array: **Linear Search (Recursive)**

```java
public boolean contains(T anEntry) {
    return search(0, numberOfEntries - 1, anEntry);
} // end contains

private boolean search(int first, int last, T desiredItem)
{
    boolean found;
    if (first > last)
        found = false; // no elements to search
    else if (desiredItem.equals(list[first]))
        found = true;
    else
        found = search(first + 1, last, desiredItem);
    return found;
} // end search
```

Figure 18-3 A recursive sequential search of an array that (a) finds its target
Figure 18-3 A recursive sequential search of an array that (b) does not find its target

(b) A search for 6

Look at the first entry, 9:

```
9 5 8 4 7
```

6 ≠ 9, so search the next subarray.

Look at the first entry, 5:

```
5 8 4 7
```

6 ≠ 5, so search the next subarray.

Look at the first entry, 4:

```
4 7
```

6 ≠ 4, so search the next subarray.

Look at the first entry, 7:

```
7
```

6 ≠ 7, so search an empty array.

No entries are left to consider, so the search ends. 6 is not in the array.
Contents

• Search: The Problem & Motivation
• Computation complexity of Linear and Binary Search
• Choosing a Search Method

<table>
<thead>
<tr>
<th></th>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>Linear Search</td>
<td>Binary Search</td>
</tr>
<tr>
<td></td>
<td>- Iterative</td>
<td>Java: binarySearch</td>
</tr>
<tr>
<td></td>
<td>- Recursive</td>
<td></td>
</tr>
<tr>
<td>Chain of linked Nodes</td>
<td>Linear Search</td>
<td>Linear Search</td>
</tr>
<tr>
<td></td>
<td>- Iterative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Recursive</td>
<td></td>
</tr>
</tbody>
</table>
Sequential Search of a Sorted Array

• Sequential search can be more efficient if the data is sorted
  ▪ Search may be terminated earlier when looking for year 2000!

• Exercise: Modify iterative contains() method to take advantage of sorted array

```java
public boolean contains(T anEntry)
{
    boolean found = false;
    for (int index = 0; !found && (index < numberOfEntries); index++)
    {
        if (anEntry.equals(list[index]))
            found = true;
    } // end for
    return found;
} // end contains
```

• Leading Question? Can we do better?
Java Class Library: The Method `binarySearch`

- Class `Arrays` contains versions of static method
  - Note specification

```java
/** Searches an entire array for a given item.
 * @param array an array sorted in ascending order
 * @param desiredItem the item to be found in the array
 * @return index of the array entry that equals desiredItem;
 *         otherwise returns -belongsAt - 1, where belongsAt is
 *         the index of the array element that should contain
 *         desiredItem */

public static int binarySearch(type[] array, type desiredItem);
```
Figure 18-5 Eliminating one half of the data when the data is sorted
Binary Search of a Sorted Array

- Algorithm for binary search

```java
Algorithm binarySearch(a, first, last, desiredItem)
mid = (first + last) / 2 // approximate midpoint
if (first > last)
    return false
else if (desiredItem equals a[mid])
    return true
else if (desiredItem < a[mid])
    return binarySearch(a, first, mid - 1, desiredItem)
else // desiredItem > a[mid]
    return binarySearch(a, mid + 1, last, desiredItem)
```

Ex. 6. Which elements in the array (4, 8, 12, 14, 20, 24) are compared to target, when the target is
(a.) 2     (b.) 8     (c.) 15

Ex. 8. What changes are needed to code if array is sorted in descending order?
Figure 18-6 A recursive binary search of a sorted array that (a) finds its target;

Ex. 5. How many comparisons are used by binary search when searching for
(a.) 8
(b.) 16
Figure 18-6 A recursive binary search of a sorted array that (b) does not find its target

(b) A search for 16

Look at the middle entry, 10:

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>15</th>
<th>18</th>
<th>21</th>
<th>24</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

16 > 10, so search the right half of the array.

Look at the middle entry, 18:

<table>
<thead>
<tr>
<th>12</th>
<th>15</th>
<th>18</th>
<th>21</th>
<th>24</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

16 < 18, so search the left half of the array.

Look at the middle entry, 12:

<table>
<thead>
<tr>
<th>12</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

16 > 12, so search the right half of the array.

Look at the middle entry, 15:

<table>
<thead>
<tr>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

16 > 15, so search the right half of the array.

The next subarray is empty, so the search ends. 16 is not in the array.
Binary Search of a Sorted Array

- Implementation of a sorted list

```java
public class SortedArrayList<T extends Comparable<? super T>> implements SortedListInterface<T>
```

- Method `contains` that will call `binarySearch`

```java
public boolean contains(T anEntry) {
    return binarySearch(0, numberOfEntries - 1, anEntry);
} // end contains
```

```java
private boolean binarySearch(int first, int last, T desiredItem) {
    boolean found;
    int mid = first + (last - first) / 2;
    if (first > last)
        found = false;
    else if (desiredItem.equals(list[mid]))
        found = true;
    else if (desiredItem.compareTo(list[mid]) < 0)
        found = binarySearch(first, mid - 1, desiredItem);
    else
        found = binarySearch(mid + 1, last, desiredItem);
    return found;
} // end binarySearch
```
Computational Complexity

• Given \( n \) elements to be searched
• Linear search may take up to \( n \) operations
• Binary search: Number of recursive calls is of order \( \log_2 n \)
  ▪ Proof sketch: page 460
• **Ex. 9.** Think of a number between 1 and 1 million. When I guess at your number, tell me whether my guess is correct, too high or too low. At most, how many attempts will I need before I guess correctly?

\[
\begin{align*}
\text{The time efficiency of a sequential search of an array} \\
\text{Best case: } & \quad O(1) \\
\text{Worst case: } & \quad O(n) \\
\text{Average case: } & \quad O(n)
\end{align*}
\]

\[
\begin{align*}
\text{The time efficiency of a binary search} \\
\text{Best case: } & \quad O(1) \\
\text{Worst case: } & \quad O(\log n) \\
\text{Average case: } & \quad O(\log n)
\end{align*}
\]
### Contents

- **Search**: The Problem & Motivation
- Computation complexity of Linear and Binary Search
- Choosing a Search Method

<table>
<thead>
<tr>
<th></th>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Array</strong></td>
<td>Linear Search</td>
<td>Binary Search</td>
</tr>
<tr>
<td></td>
<td>- Iterative</td>
<td>Java: binarySearch</td>
</tr>
<tr>
<td></td>
<td>- Recursive</td>
<td></td>
</tr>
<tr>
<td><strong>Chain of linked Nodes</strong></td>
<td>Linear Search</td>
<td>Linear Search</td>
</tr>
<tr>
<td></td>
<td>- Iterative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Recursive</td>
<td></td>
</tr>
</tbody>
</table>

Copyright ©2012 by Pearson Education, Inc. All rights reserved
Iterative Sequential Search of an **Unsorted** Chain

Figure 18-7 A chain of linked nodes that contain the entries in a list

```java
public boolean contains(T anEntry)
{
    boolean found = false;
    Node currentNode = firstNode;
    while (!found && (currentNode != null))
    {
        if (anEntry.equals(currentNode.getData()))
            found = true;
        else
            currentNode = currentNode.getNextNode();
    } // end while

    return found;
} // end contains
```
Sequential Search of a Sorted Chain

**Exercise:** Which of the following is better for searching a sorted chain of node?

```java
public boolean contains(T anEntry)
{
    Node currentNode = firstNode;
    while ( (currentNode != null) &&
        (anEntry.compareTo(currentNode.getData()) > 0) )
    {
        currentNode = currentNode.getNextNode();
    } // end while
    return (currentNode != null) &&
        anEntry.equals(currentNode.getData());
} // end contains
```

```java
public boolean contains(T anEntry)
{
    boolean found = false;
    Node currentNode = firstNode;
    while (!found &&
        (currentNode != null))
    {
        if (anEntry.equals(currentNode.getData()))
            found = true;
        else
            currentNode = currentNode.getNextNode();
    } // end while
    return found;
} // end contains
```
Why not Binary Search a Sorted Chain?

• Recall finding middle element in an array is easy:
  ▪ mid = first + (last - first) / 2
• Finding middle element of sorted chain is costly!
  ▪ Must traverse the chain to middle
  ▪ Less efficient than sequential search
• What may be done for linked collection of nodes?
  ▪ Organize differently: Binary Search Trees (Chapter 25)

Efficiency of a Sequential Search

The time efficiency of a sequential search of a chain

Best case: \( O(1) \)
Worst case: \( O(n) \)
Average case: \( O(n) \)
Summary: Choosing a Search Method

- Sequential search
  - Suitable for smaller array
  - Objects must have appropriate `equals` method

- Binary search
  - Suitable for larger collections
  - Objects must have appropriate `compareTo` method

- In both cases ask, “Search how often (relative to insert/delete)?”

- May one search faster than binary search?
  - Hash Tables (Chapter 21)

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Best Case</th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential search (unsorted data)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Sequential search (sorted data)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Binary search (sorted array)</td>
<td>O(1)</td>
<td>O(log n)</td>
<td>O(log n)</td>
</tr>
</tbody>
</table>

---

Figure 18-8 The time efficiency of searching, expressed in Big Oh notation.