Searching

Problem: Given a collection of items and a key, find an item with the given key.

Questions: What if no item contains the key? How do you signal this? What if more than one item contains the key? Do you want all the items? the first item? the last item?

There are several solutions: linear search, binary search, and hashing. We will examine linear search and binary search and defer hashing to the next course.
Linear Search

Linear search strategy: "It’s always in the last place I look.” Search the collection one item at a time. Stop when the key is found.

Setup: Each item contains a key. The keys are in an array list of size numElems. The given search key will be called val. We return the first location of the key in the array as position. Since arrays are indexed starting at 0, we return -1 to flag the absence of val in the array.
Example: TBD

Algorithm

\begin{verbatim}
lsearch(list, numElems, val)
1.    found ← false;
2.    index ← 0;
3.    position ← -1;
4.    while (found == false and index < numElems ) do
5.        if list[index] == val
6.            then
7.                found ← true;
8.                position ← index;
9.                index ++;
10.        return position
\end{verbatim}

C++ Code is in the textbook, pages 456-457.
Efficiency

• Linear search runs in “linear” time and space.
  – The solution to a problem of size $n$ requires $c_1 + c_2 n$ time, where $c_1$ and $c_2$ are constants.
  – The solution to a problem of size $n$ requires $a_1 + a_2 n$ space to solve, where $a_1$ and $a_2$ are constants.

• Linear search is optimal.
  – Consider any algorithm $A$ that searches by comparing items and makes no assumptions about the ordering of items. Then there is some input of size $n$ for which $A$ must perform $n$ comparisons.
Binary Search

Binary search strategy: Telephone book or dictionary lookup. The collection is ordered from smallest to largest. Test the middle item against the key. If they are equal, return the middle. If the middle item is larger than the key, then the key is in the "smaller" half of the array. If the middle item is smaller than the key, then the key is in the "larger" half of the array. Stop when the key is found or when you run out of items.

Maintain two array indices, first and last, that bracket the portion of the array left to be searched.

Setup: The items are ordered by keys from smallest to largest. Each item contains a key. The keys are in an array list of size numElems. The given search key will be called val. We return the first location of the key in the array as position. Since arrays are indexed starting at 0, we return -1 to flag the absence of val in the array.

Example: TBD
Algorithm

\textbf{bsearch}(list, \text{n}umElems, \text{val})
\begin{algorithmic}
1. \text{found} \leftarrow \text{false};
2. \text{first} \leftarrow 0;
3. \text{last} \leftarrow 0;
4. \text{position} \leftarrow -1;
5. \textbf{while} (\text{found} == \text{false} \textbf{and} \text{first} \leq \text{last}) \textbf{do}
6. \hspace{1em} \text{middle} \leftarrow (\text{first} + \text{last})/2;
7. \hspace{1em} \textbf{if} (\text{list}[\text{middle}] == \text{val})
8. \hspace{2em} \textbf{then}
9. \hspace{3em} \text{found} \leftarrow \text{true};
10. \hspace{3em} \text{position} \leftarrow \text{middle};
11. \hspace{1em} \textbf{else if} (\text{list}[\text{middle}] > \text{val})
12. \hspace{2em} \textbf{then}
13. \hspace{3em} \text{last} \leftarrow \text{middle} -1
14. \hspace{1em} \textbf{else}
15. \hspace{2em} \text{first} \leftarrow \text{middle} + 1
16. \hspace{1em} \textbf{return} \text{position}
\end{algorithmic}

C++ Code is in the textbook, pages 456-457.
Efficiency

- Binary search runs in "linear" space.
  - The solution to a problem of size $n$ requires $a_1 + a_2 n$ space to solve, where $a_1$ and $a_2$ are constants.

- Binary search runs in "log" time
  - The solution to a problem of size $n$ requires $c_1 + c_2 \log_2 n$ time, where $c_1$ and $c_2$ are constants and $\log$ is log base 2.