Searching and Sorting

Topics

- Sequential Search on an Unordered File
- Sequential Search on an Ordered File
- Binary Search
- Bubble Sort
- Insertion Sort

Reading

- Sections 6.6 - 6.8

Common Problems

- There are some very common problems that we use computers to solve:
  - **Searching** through a lot of records for a specific record or set of records
  - Placing records in order, which we call **sorting**
- There are numerous algorithms to perform searches and sorts. We will briefly explore a few common ones.

Searching

- A question you should always ask when selecting a search algorithm is "How fast does the search have to be?" The reason is that, in general, the faster the algorithm is, the more complex it is.
- Bottom line: you don't always need to use or should use the fastest algorithm.
- Let's explore the following search algorithms, keeping speed in mind.
  - Sequential (linear) search
  - Binary search

Sequential Search on an Unordered File

- Basic algorithm:

```
Get the search criterion (key)
Get the first record from the file
While ( (record != key) and (still more records) )
   Get the next record
End_while
```

- When do we know that there wasn't a record in the file that matched the key?

Sequential Search on an Ordered File

- Basic algorithm:

```
Get the search criterion (key)
Get the first record from the file
While ( (record < key)  and  (still more records) )
   Get the next record
End_while
If ( record = key )
   Then success
   Else there is no match in the file
End_else
```

- When do we know that there wasn't a record in the file that matched the key?

Sequential Search of Ordered vs. Unordered List

- Let's do a comparison.
- If the order was ascending alphabetical on customer's last names, how would the search for John Adams on the ordered list compare with the search on the unordered list?
  - Unordered list
    - if John Adams was in the list?
    - if John Adams was not in the list?
  - Ordered list
    - if John Adams was in the list?
    - if John Adams was not in the list?

Ordered vs Unordered (cont.)

- How about George Washington?
  - Unordered
    - if George Washington was in the list?
    - If George Washington was not in the list?
  - Ordered
    - if George Washington was in the list?
    - If George Washington was not in the list?
- How about James Madison?

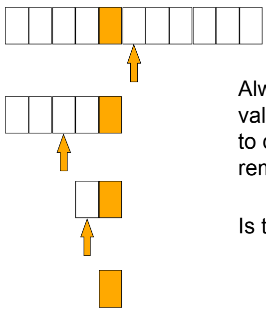
Ordered vs. Unordered (cont.)

- Observation: the search is faster on an ordered list only when the item being searched for is not in the list.
- Also, keep in mind that the list has to first be placed in order for the ordered search.
- Conclusion: the **efficiency** of these algorithms is roughly the same.
- So, if we need a faster search, we need a completely different algorithm.
- How else could we search an ordered file?

Binary Search

- If we have an ordered list and we know how many things are in the list (i.e., number of records in a file), we can use a different strategy.
- The **binary search** gets its name because the algorithm continually divides the list into two parts.

How a Binary Search Works



Always look at the center value. Each time you get to discard half of the remaining list.

Is this fast ?

How Fast is a Binary Search?

- Worst case: 11 items in the list took 4 tries
- How about the worst case for a list with 32 items ?
  - 1st try - list has 16 items
  - 2nd try - list has 8 items
  - 3rd try - list has 4 items
  - 4th try - list has 2 items
  - 5th try - list has 1 item

How Fast is a Binary Search?

List has 250 items

1st try - 125 items
2nd try - 63 items
3rd try - 32 items
4th try - 16 items
5th try - 8 items
6th try - 4 items
7th try - 2 items
8th try - 1 item

List has 512 items

1st try - 256 items
2nd try - 128 items
3rd try - 64 items
4th try - 32 items
5th try - 16 items
6th try - 8 items
7th try - 4 items
8th try - 2 items
9th try - 1 item

What's the Pattern?

- List of 11 took 4 tries
- List of 32 took 5 tries
- List of 250 took 8 tries
- List of 512 took 9 tries

- $32 = 2^5$ and $512 = 2^9$
- $8 < 11 < 16$ $2^3 < 11 < 2^4$
- $128 < 250 < 256$ $2^7 < 250 < 2^8$

A Very Fast Algorithm!

- How long (worst case) will it take to find an item in a list 30,000 items long?

$2^{10} = 1024$	$2^{13} = 8192$
$2^{11} = 2048$	$2^{14} = 16384$
$2^{12} = 4096$	$2^{15} = 32768$

- So, it will take only 15 tries!

Lg n Efficiency

- We say that the binary search algorithm runs in **$\log_2$ n time**. (Also written as **lg n**)
- Lg n means the log to the base 2 of some value of n.
- $8 = 2^3$ lg 8 = 3 $16 = 2^4$ lg 16 = 4
- <u>There are no algorithms that run faster than lg n time</u>.

Sorting

- So, the binary search is a very fast search algorithm.
- But, the list has to be sorted before we can search it with binary search.
- To be really efficient, we also need a fast sort algorithm.

Common Sort Algorithms

Bubble Sort	Heap Sort
Selection Sort	Merge Sort
Insertion Sort	Quick Sort

- There are many known sorting algorithms. Bubble sort is the slowest, running in **n^2 time**. Quick sort is the fastest, running in **n lg n time**.
- As with searching, the faster the sorting algorithm, the more complex it tends to be.
- We will examine two sorting algorithms:
  - Bubble sort
  - Insertion sort

Bubble Sort - Let's Do One!

C
P
G
A
T
O
B

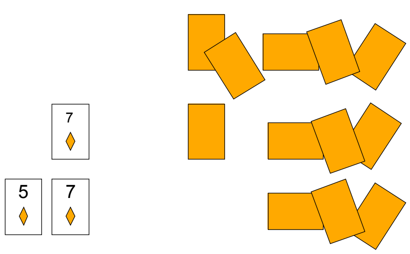
Bubble Sort Code

```
void bubbleSort (int a[ ] , int size)
{
   int i, j, temp;
   for ( i = 0; i < size; i++ )    /* controls passes through the list */
   {
           for ( j = 0; j < size - 1; j++ )   /* performs adjacent comparisons */
           {
                   if ( a[ j ] > a[ j+1 ] )   /* determines if a swap should occur */
                   {
                           temp = a[ j ];       /* swap is performed */
                           a[ j ] = a[ j + 1 ];
                           a[ j+1 ] = temp;
                   }
           }
    }
}
```

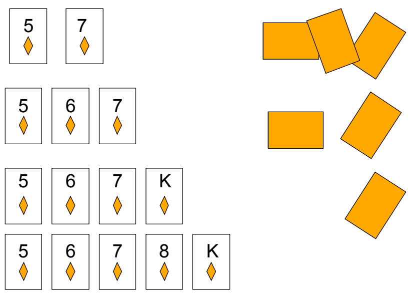
Insertion Sort

- Insertion sort is slower than quick sort, but not as slow as bubble sort, and it is easy to understand.
- Insertion sort works the same way as arranging your hand when playing cards.
  - Out of the pile of unsorted cards that were dealt to you, you pick up a card and place it in your hand in the correct position relative to the cards you're already holding.

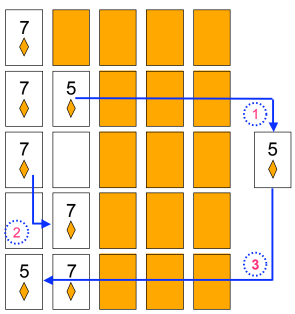
Arranging Your Hand



Arranging Your Hand



Insertion Sort



Unsorted - shaded

Look at 2nd item - 5.

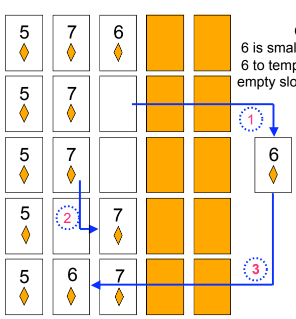
Compare 5 to 7.

5 is smaller, so move 5 to temp, leaving an empty slot in position 2.

Move 7 into the empty slot, leaving position 1 open.

Move 5 into the open position.

Insertion Sort (cont.)



Look at next item - 6.

Compare to 1st - 5.

6 is larger, so leave 5.

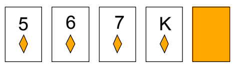
Compare to next - 7.

6 is smaller, so move 6 to temp, leaving an empty slot.

Move 7 into the empty slot, leaving position 2 open.

Move 6 to the open 2nd position.

Insertion Sort (cont.)



Look at next item - King.

Compare to 1st - 5.

King is larger, so
leave 5 where it is.

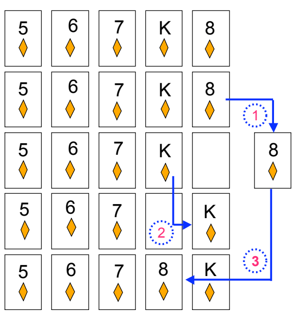
Compare to next - 6.

King is larger, so
leave 6 where it is.

Compare to next - 7.

King is larger, so
leave 7 where it is.

Insertion Sort (cont.)



Courses at UMBC

- Data Structures - CMSC 341
  - Some mathematical analysis of various algorithms, including sorting and searching
- Design and Analysis of Algorithms - CMSC 441
  - Detailed mathematical analysis of various algorithms
- Cryptology - CMSC 443
  - The study of making and breaking codes