

# Searching and Sorting

CSE 130: Introduction to Programming in C  
Spring 2005

1

# Searching and Sorting

- Searching and sorting are among the most common operations performed
  - Ex. databases
- It's much easier to search data that has been sorted
  - Ex. searching a telephone directory
  - Compare to a linear search

2

# Binary Search

- Method: choose an element at random (usually the middle) and decide whether to search the left or right half
- Ex. Searching a dictionary for a word
- At each decision point, the space to be searched is cut in half
- Requires sorted data to work properly
- Very efficient: requires  $\log(n)$  comparisons

3

# Pseudocode

If (range contains only one element):

Look for desired value

Else:

1. Get midpoint of range
2. Determine which half of range contains the value
3. Search that half of the range using binary search

4

# Binary Search Example

- Ex. Find 29
- Start: [10, 13, 14, 29, 37]
- Examine 14: [10, 13] [14] [29, 37]
- Find 29: [10, 13, 14] [29] [37]

5

```
if (first > last)
    return -1; // value not in array
else
{
    int mid = (first + last)/2;
    if (value == A[mid])
        return mid;
    else if (value < A[mid])
        BinarySearch(A, first, mid-1, value);
    else
        BinarySearch(A, mid+1, last, value);
}
```

6

## Sorting Techniques

- Many sorting techniques exist: quicksort, radix sort, mergesort, bubblesort, insertion sort, shell sort, selection sort, etc.
- These techniques differ in efficiency
  - Different sorting techniques take different amounts of time to sort the same data
  - The rate of time increase also differs

7

## Bubble Sort

- Method: compare pairs of adjacent items, and swap them if they are “out of order”
- At the end of each pass, the largest remaining element is in its proper place
- Elements “bubble” to their proper places
- This is a very easy algorithm to implement
- It’s also very inefficient

8

## Pseudocode

- I. Repeat N-1 times:
  - A. Repeat N-1 times:
    - I. if  $A[x] > A[x+1]$ , swap them

9

## Bubble Sort Example

Start: [29, 10, 14, 37, 13]

After 1st pass: [10, 14, 29, 13, 37]

After 2nd pass: [10, 14, 13, 29, 37]

After 3rd pass: [10, 13, 14, 29, 37]

End: [10, 13, 14, 29, 37]

10

## Bubble Sort Code

```
for (end = N - 1; end > 0; end--)
{
    for (loc = 0; loc < end - 1; loc++)
    {
        if (A[loc] > A[loc+1])
            // Swap A[loc] and A[loc+1]
    }
}
```

11

## Selection Sort

- Method: Search for the largest remaining item, and put it in its proper sorted position
- Ex. Looking at an entire hand of cards and selecting one at a time
- On each pass, swap the largest item with the last (unsorted) element

12

## Pseudocode

1. Repeat N-1 times:
  1. Find the largest (unsorted) element
  2. Swap A[last] with A[largest]

13

## Selection Sort Example

Start: [29, 10, 14, 37, 13]  
After 1st pass: [29, 10, 14, 13, 37]  
After 2nd pass: [13, 10, 14, 29, 37]  
After 3rd pass: [13, 10, 14, 29, 37]  
After 4th pass: [10, 13, 14, 29, 37]  
End: [10, 13, 14, 29, 37]

14

## Selection Sort Code

```
int indexOfLargest(int A[], int size)
{
    int currIndex, indexSoFar = 0;
    for (currIndex = 1; currIndex < size; currIndex++)
    {
        if (A[currIndex] > A[indexSoFar])
            indexSoFar = currIndex;
    }
    return indexSoFar;
}
```

15

## Selection Sort Code (2)

```
for (last = N-1; last >= 1; last--)
{
    int L = indexOfLargest(A, last+1);
    // Swap A[L] and A[last]
}
```

16

## Insertion Sort

- Method: select one element at a time and insert it into its proper position
- Ex. arranging a hand of cards
- Begin by dividing the array into two regions: sorted and unsorted
  - Initially, the sorted region is empty
  - At each step, move first unsorted item into its proper position in the sorted region

17

## Pseudocode

1. A[0] is sorted; A[1]-A[N-1] are unsorted
2. Repeat N times:
  1. nextItem = first unsorted element
  2. Shift sorted elements > nextItem over one position (A[x] = A[x-1])
  3. Insert nextItem into correct position

18

## Insertion Sort Example

- Start: [29][ 10, 14, 37, 13]
- Move 10: [10, 29][ 14, 37, 13]
- Move 14: [10, 14, 29][ 37, 13]
- Move 37: [10, 14, 29, 37][ 13]
- Move 13: [10, 13, 14, 29, 37][ ]
- End: [10, 13, 14, 29, 37]

19

## Insertion Sort Code

```
int unsorted;
for (unsorted = 1; unsorted < N; unsorted++)
{
    int nextItem = a[unsorted];
    int loc;
    for (loc = unsorted; (loc > 0) && (a[loc-1] > nextItem); loc--)
        a[loc] = a[loc-1]; /* shift a[loc-1] to the right */
    a[loc] = nextItem; /*insert nextItem into sorted region */
}
```

20