Ch 8. Searching and Sorting Arrays
Part 2

CS 2308
Fall 2011

Jill Seaman

Lecture 2

Sorting Algorithms

- Sort: rearrange the items in an array into some order (smallest to biggest, alphabetical order, etc.).
- Bubble Sort
- Selection Sort
- Quicksort

The Bubble Sort

1. Compare first two elements. If the first is bigger, they exchange places (swap).
2. Compare second and third elements. If second is bigger, exchange them.
3. Repeat until last two elems of array are compared.
4. Repeat the process until you make a complete pass with no exchanges.

Example

• 7 2 3 8 9 1	7 > 2, swap
• 2 7 3 8 9 1	7 > 3, swap
• 2 3 7 8 9 1	!(7 > 8), no swap
• 2 3 7 8 9 1	!(8 > 9), no swap
• 2 3 7 8 9 1	9 > 1, swap
• 2 3 7 8 1 9	finished pass 1, did 3 swaps

Example cont.

- 2 3 7 8 1 9   2<3<7<8, no swap, !(8<1), swap
- 2 3 7 1 8 9   (8<9) no swap
- finished pass 2, did one swap
- 2 3 7 1 8 9   2<3<7, no swap, !(7<1), swap
- 2 3 1 7 8 9   7<8<9, no swap
- finished pass 3, did one swap

Example continued

- 2 3 1 7 8 9   2<3, !(3<1) swap, 3<7<8<9
- 2 1 3 7 8 9   finished pass 4, did one swap
- 2 1 3 7 8 9   !(2<1) swap, 2<3<7<8<9
- 1 2 3 7 8 9   finished pass 5, did on swap
- 1 2 3 7 8 9   1<2<3<7<8<9, no swaps
- finished pass 6, no swaps, list is sorted!

How does it work?

- After the nth pass, the last n+1 items of the list are sorted.
- During the next pass (n+1), the item before the last n+1 items will be swapped into the right position, so that the last n+2 items are sorted.
- If the list has z elements, it will be sorted within z passes.

Pseudocode

```
procedure bubbleSort( A : list of sortable items )
  repeat
    swapped = false
    for i = 0 to length(A)-2 inclusive do:
      if A[i] > A[i+1] then
        swap( A[i], A[i+1] )
        swapped = true
      end if
    end for
  until not swapped
end procedure
```

(From wikipedia)

Bubble Sort in C++

```
void sortArray (int array[], int size) {

   bool swap;
   int temp;

   do {

      swap = false;
      for (int i = 0; i < (size-1); i++) {

         if (array [i] > array[i+1]) {

            temp = array[i];
            array[i] = array[i+1];
            array[i+1] = temp;
            swap = true;
         }
      }
   } while (swap);
}
```

Program using bubble sort

```
#include <iostream>
using namespace std;

void sortArray(int [], int);
void showArray(int [], int);

int main() {
   int values[6] = {7, 2, 3, 8, 9, 1};

   cout << “The unsorted values are: \n”;
   showArray (values, 6);

   sortArray (values, 6);

   cout << “The sorted values are: \n”;
   showArray(values, 6);
}

//see book for definition of showArray
```

Efficiency of Bubble Sort

- $O(n^2)$
- We pass over the list N times.
- Each pass has N-1 comparisons
- Worst case: N*(N-1)

Selection Sort

- The smallest (minimum) element in the array is exchanged with element at location 0.
- The smallest element in the array from location 1 to the end is exchanged with the element at location 1.
- Repeat until the end of the array.

Example

7 2 3 8 9 1	minimum: 1, swap with 7
1 2 3 8 9 7	minimum after 1: 2, no swap
1 2 3 8 9 7	minimum after 2: 3, no swap
1 2 3 8 9 7	min after 3: 7, swap with 8
1 2 3 7 9 8	min after 7: 8, swap with 9
1 2 3 7 8 9	done

Pseudocode

```
find index of minimum value of an array A starting at position x:
   minI = x
   for i = x+1 to length(A) – 1  inclusive do
      if (A[i] < A[mini])
         minI = i
      end if
   end for
   return minI

sort an array B:
   for j = 0 to length(B) – 2 inclusive do
      minIndex = find index of minimum value of array B
                 starting at position j+1
      swap (B[j],B[minIndex]);
   end for
```

Selection Sort in C++

```
int findIndexOfMin (int array[], int size, int start) {
   int minIndex = start;
   for (int i = start+1; i < size; i++) {
      if (array[i] < array[minIndex]) {
         minIndex = i;
      }
   }
   return minIndex;
}

void selectionSort (int array[], int size) {
   int index;
   for (index = 0; index < (size -1); index++) {
      minIndex = findIndexOfMin(array, size, index+1);
      temp = array[minIndex];
      array[minIndex] = array[index];
      array[index] = temp;
   }
}
```

Program using Selection Sort

- See Program using Bubble Sort
  - replace “sortArray” with “selectionSort”
  - test and make sure the result is the same.

Efficiency of Selection Sort

- $O(n^2)$
- First pass: N-1 comparisons.
- Second pass: N-2 comparisons
- etc.
- Worst case: (N-1) + (N-2) + ... + 2 + 1
- Still approximates N^2, more so than it does N log N

Quicksort

- A divide and conquer algorithm
- If the list is size zero or one, it is already sorted. Otherwise:
- Pick an element, called a pivot, from the list.
- Reorder (partition) the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater thn (or equal to) the pivot come after it.
- Sort the sub-list of lesser elements
- Sort the sub-list of greater elements.

Example

- 7 2 3 8 9 1 Pick 7 as the pivot, partition:
- 2 3 1 7 8 9 Sort 2 3 1, then sort 8 9
- 2 3 1 Pick 2 as the pivot, partition:
- 1 2 3 Sort 1, then sort 3 (done)
- 8 9 Pick 8 as the pivot, partition:
- 8 9 Sort 9 (done)
- 1 2 3 7 8 9 All done

Efficiency of Quicksort

- O(n log n)
- Oversimplified explanation:
  - Partition requires N comparisons
  - Each step divides the size of the list to be sorted in half (hopefully).
  - We can only divide the list in half $\log_2 N$ times.
- O(n log n) is better than $O(n^2)$