UNIVERSITY OF MASSACHUSETTS
DARTMOUTH

ECE160: Foundations of Computer Engineering I

Lecture #24 – Array Sort

Instructor: Dr. Liudong Xing
SENG-213C, lxing@umassd.edu
ECE Dept.

Administrative Issues (4/7)

- Last day to withdraw from a class is **Friday, April 7 (Today)**

- Today's topics
  - Arrays & Functions (Finish L#23)
  - Array Sorting (Then L#24)

Review of Lectures #23

- We can pass an individual array element to a function like any other variables as long as the array element type matches the function parameter type!
  - Pass by values
  - Pass by references

- To pass the whole array to a function, we pass the address of the array (via array name), i.e., pass by references!

Add100 to myarray[2]

```c
#include "stdio.h"

void add(int *number);

void main(void)
{
    int myarray[5] = {1,2,9,3,6};
    add(&myarray[2]);
    printf("The value of myarray[2]
            is: %d\n", myarray[2]);
}

void add(int *number)
{
    *number = *number + 100;
}
```

Pass an array element's address to a function

```c
#include "stdio.h"

void add(int arr[]);

void main(void)
{
    int myarray[5]= {1,2,9,3,6};
     add(myarray);
    printf("The value of myarray[2]
    is: %d\n", myarray[2]);
}

void add(int arr[])
{
    arr[2] = arr[2] + 100;
}
```

Pass the whole array to a function

Agenda

- **Array sorting**
  - Problem statement
  - Bubble sort
  - Selection sort

The Sorting Problem

- Sort a sequence of numbers into non-decreasing (from minimum value to maximum value) or non-increasing (from maximum value to minimum value) order

3,4,6,2,4,5 → A Sorting Program → {2,3,4,4,5,6}

An input instance

The output

The Sorting Problem – Formal Definition

- **Input:** A sequence of n numbers $a_1, a_2, \ldots, a_n$.
- **Output:** A permutation (reordering) $a_1', a_2', \ldots, a_n'$ of the input sequence such that

$$a_1' \leq a_2' \leq \cdots \leq a_n'$$ non-decreasing

or

$$a_1' \geq a_2' \geq \cdots \geq a_n'$$ non-increasing

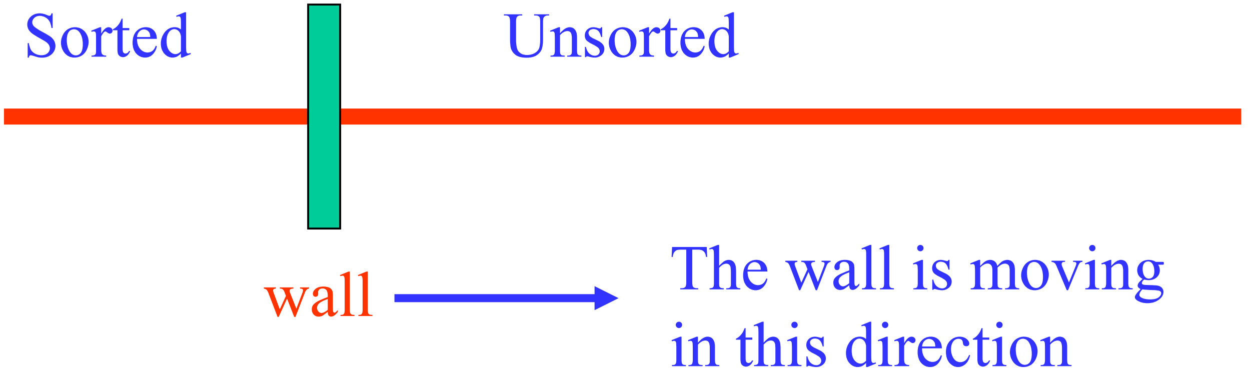
Sorting Algorithms/Methods

- Bubble sort: works by repeatedly swapping adjacent elements that are out of order

- Selection sort: works by repeatedly selecting the smallest/largest remaining element

Bubble Sort

- In the bubble sort, the list of elements to be sorted is divided into two sublists: sorted and unsorted.
- The smallest element is bubbled from the unsorted sublist and moved to the sorted sublist.
- Then the wall moves one element ahead, increasing # of sorted elements and decreasing # of unsorted ones.



Bubble Sort (Cont'd)

- Works by repeatedly comparing adjacent elements and swapping adjacent elements that are out of order
- Example: a[6] = {23, 78, 45, 8, 32, 56}
  - start from the right 56 and compare it to 32
  - 56 does not move, because 32 is smaller.
  - 32 does not move because 8 is smaller.
  - Swap 45 and 8 because 8 is smaller than 45.
  - Swap 78 and 8 because 8 is smaller
  - Swap 23 and 8 because 8 is smaller
  - 8 bubbles up to the top!

    8 | 23 78 45 32 56.
  - The next time (2nd pass) 23 is going to bubble up to the left (sorted list)
  - …..

An Example

Initial array	23	78	45	8	32	56
1st pass	8	23	78	45	32	56
2nd pass	8	23	32	78	45	56
3rd pass	8	23	32	45	78	56
4th pass	8	23	32	45	56	78
5th pass	8	23	32	45	56	78

A Function to Implement Bubble Sort

```
void bubbleSort(int list[], int last)  /*last = (array size -1) */
{
    int current, walker, temp;
    for(current=0; current < last; current++)
        for(walker=last; walker > current; walker--)
            if(list[walker] < list[walker-1])
            {
               temp = list[walker];             /*Swapping two
               list[walker] = list[walker-1];   array elements
               list[walker-1] = temp;           (Lecture #22)*/
            }
}
```

Note!

- Why does the outer **for** loop need to run for only the first *n*-1 elements, rather than for all *n* elements, if *n* is the array_size?

- Answer: After the first n – 1 elements, the subarray $A[1\ .\ .\ n - 1]$ contains the smallest n - 1 elements, sorted, and therefore element $A[n]$ must be the largest element.

Bubble Sort (Example Revisit)

Initial array	23	78	45	8	32	56
1st pass current=0	8	23	78	45	32	56
2nd pass current=1	8	23	32	78	45	56
3rd pass current=2	8	23	32	45	78	56
4th pass current=3	8	23	32	45	56	78
5th pass current=4	8	23	32	45	56	78

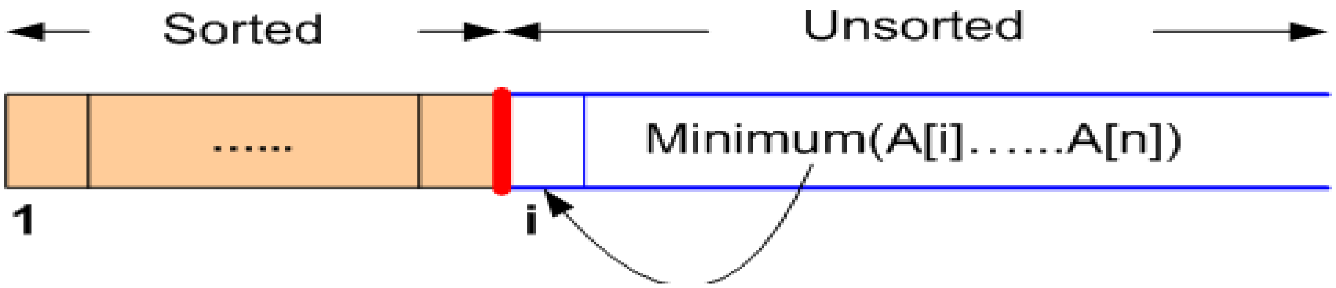
Exercise (1): Fill in missing parts to complete the program

```c
#include "stdio.h"
#define ARRAY_SIZE 6

//add function prototype here
                    ???

void main(void)
{
int myarray[ARRAY_SIZE];
int i = 0;

printf("Please input the array
elements:\n");
for (i = 0; i < ARRAY_SIZE; i++)
{
  scanf_s("%d", &myarray[i]);
}

//call the bubbleSort function here
                    ???

printf("The array elements after sorting
are:\n");
                    ???

}
```

```c
void bubbleSort(int list[], int last)
{
int current, walker, temp;
for (current = 0; current < last; current++)
  for (walker = last; walker > current; walker--)
     if (list[walker] < list[walker - 1])
     {
       temp = list[walker];
       list[walker] = list[walker - 1];
       list[walker - 1] = temp;
     }
}
```

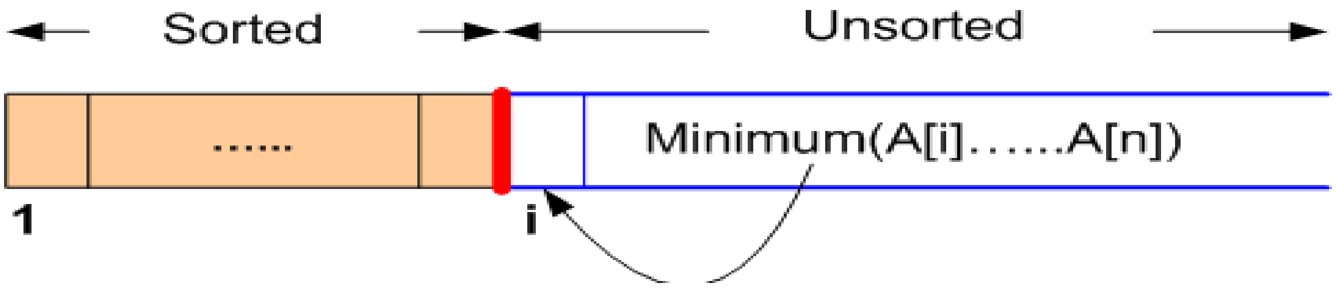
Selection Sort

- Works by repeatedly selecting the smallest remaining element
- The list of elements to be sorted is divided into two sublists: sorted and unsorted.
- Find the smallest element from the unsorted list and exchange it with the element at the first position of the unsorted list

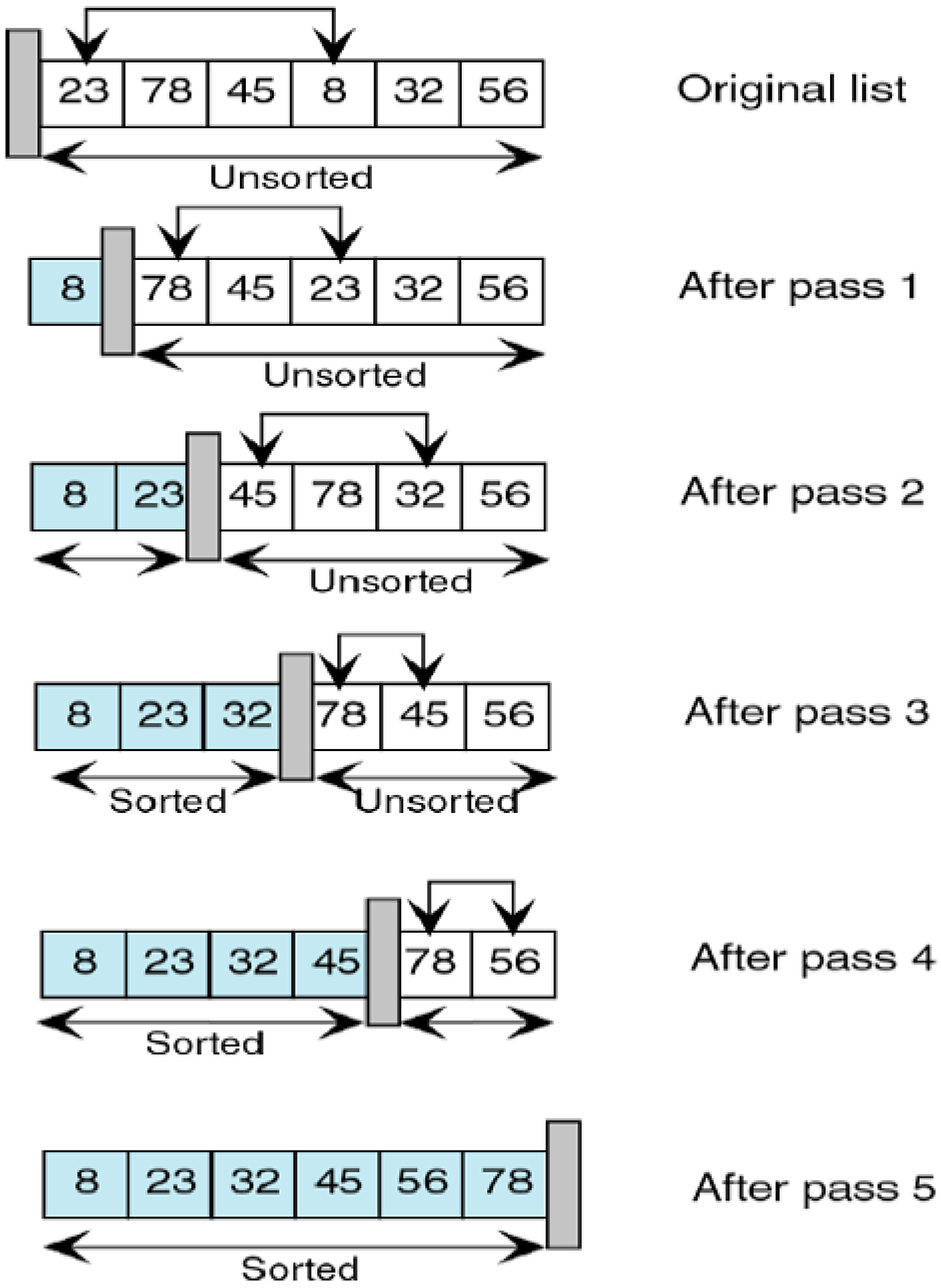


Selection Sort (Cont'd)

- Then move the wall one element ahead, increasing # of sorted elements and decreasing # of unsorted ones
- Until the entire array is sorted



An Example



23 78 45 8 32 56
Unsorted
Original list
8 78 45 23 32 56
Unsorted
After pass 1
8 23 45 78 32 56
Unsorted
After pass 2
8 23 32 78 45 56
Sorted
Unsorted
After pass 3
8 23 32 45 78 56
Sorted
After pass 4
8 23 32 45 56 78
Sorted
After pass 5

A Function to Implement Selection Sort

```
void selectionSort(int list[], int last)
{
    int current, walker, temp, min;
    for(current=0; current < last; current++)
    {
        min=current;

        for(walker=current+1; walker <=last; walker++)
            if(list[walker] < list[min])
                min=walker;

        /*smallest selected: exchange with current element*/
          temp = list[current];
          list[current] = list[min];
          list[min] = temp;
    }
}
```

Exercise (2)

- Write a program that sorts the elements of an array in the non-decreasing order using **selection sort**, and then prints them out. The array contains 6 integers which are entered from the keyboard.
  1. Enter array elements from the keyboard
  2. Sort the array elements using function selectionSort()
  3. Output the sorted array elements on the screen

Exercise (2): Fill in missing parts to complete the program

```
#include "stdio.h"
#define ARRAY_SIZE  6

//add function prototype here
                    ???

void main(void)
{
int myarray[ARRAY_SIZE];
int i = 0;

printf("Please input the array
elements:\n");
                    ???

//call the selectionSort function here
                    ???

printf("The array elements after sorting
are:\n");
                    ???

}
```

```
 void selectionSort(int list[], int last)
{
  int current, walker, temp, min;
  for (current = 0; current < last; current++)
  {
     min = current;
     for (walker = current + 1; walker <= last;
walker++)
     if (list[walker] < list[min])
               min = walker;
     /*smallest selected: exchange with current
element*/
     temp = list[current];
     list[current] = list[min];
     list[min] = temp;
   }
}
```

Summary of Lecture #24

- Sorting problem is a problem to sort/arrange a sequence of numbers into non-decreasing or non-increasing order
- Bubble sort works by repeatedly comparing adjacent elements and swapping adjacent elements that are out of order
- Selection sort works by repeatedly selecting the smallest/largest remaining element

Things To Do

- Review lecture notes
- Run and test the programs in Exercises (1) and (2) on Slides 15 and 21 (refer to the solution file for the complete programs)

Next Topic

- Strings and pointers