Introduction to Computers and Programming
Homework 3 (Week 4)
Due date: 2020/10/22(Thur.)

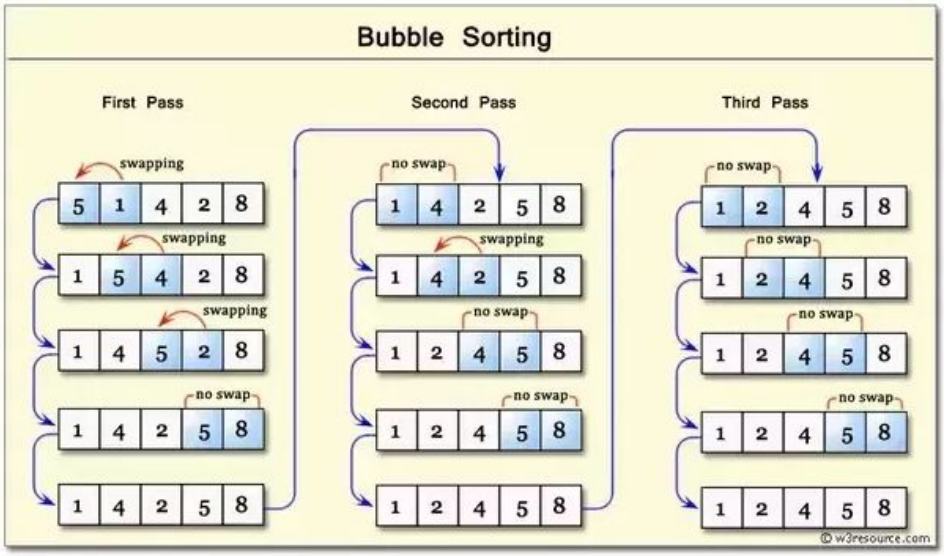
1. Sort Algorithm

1-1

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.

You can see the figure and pseudo code below.

Please implement your bubble sort by asking users to input 8 numbers and sort them in ascending order. You can **only use Bubble Sort** to implement the sorting algorithm. **You would be considered failed if you use another algorithm even if your result is correct.** Try to convince TA your implementation is Bubble Sort when demoing your code.



The pseudo code would look like this:

```
func bubblesort( var a as array )
   for i from 1 to N
        for j from 0 to N - 1
           if a[j] > a[j + 1]
              swap( a[j], a[j + 1] )
end func
```

P.S. Name your code to `109550XXX-hw4-1.c` when you submit to newE3.

1-2

Google “Insertion Sort” and do some research. Try to **explain** to TA how it works, and what the difference between “Insertion Sort” and “Bubble Sort” when demoing your results of 1-1. You don’t need to implement it or to write a report of Insertion Sort.

2. Anagrams

Write a program that tests whether two words are anagrams (permutations of the same letters):

```
Enter first word: smartest123
Enter second word: Mattress
The words are anagrams.

Enter first word: dumbest
Enter second word: stumble
The words are not anagrams.
```

Write a loop that reads the first word, character by character, using an array of 26 integers to keep track of how many times each letter has been seen. (For example, after the word *smartest* has been read, the array should contain the values 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 2 2 0 0 0 0 0 0, reflecting the fact that *smartest* contains one *a*, one *e*, one *m*, one *r*, two *s*'s and two *t*'s.) Use another loop to read the second word, except this time decrementing the corresponding array element as each letter is read. Both loops should **ignore any characters that aren't letters**, and both should **treat upper-case letters in the same way as lower-case letters**. After the second word has been read, use a third loop to check whether all the elements in the array are zero. If so, the words are anagrams.
Hint: You may wish to use functions from `<ctype.h>`, such as `isalpha` and `tolower`.

P.S.1 The maximum number of digits in one word we input is **16**.
P.S.2 Name your code to `109550XXX-hw4-2.c` when you submit to newE3.