



PRIFYSGOL
BANGOR
UNIVERSITY

Imperative Programming in C

Laboratory 9: Sorting Algorithms

Dr. Llyr ap Cenydd

Exercise 1 - Bubble Sort

One of the most common tasks we need to perform on an array of data is to sort it. To do this, we need to use a sorting algorithm.

One of the most basic and brute force methods of sorting is called the bubble sort. The bubble sort works like this:

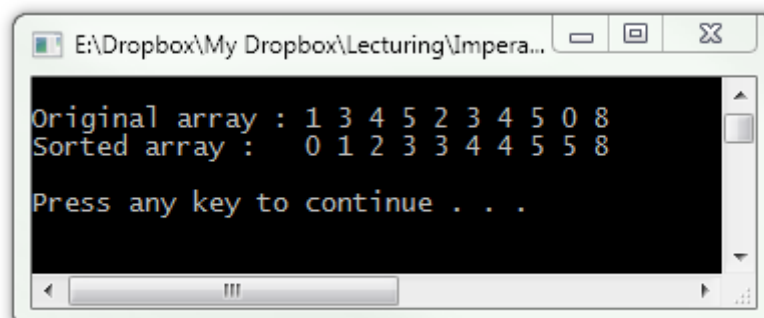
1. Loop through each element in the array, from start to end
2. Compare each pair of elements
3. Switch the two elements positions if necessary
4. Repeat this process until the array is sorted

The worst case scenario here is that the array is in reverse order, in which case the algorithm will need to loop through the array once for every item in the array.

For this exercise, your task is to write a bubble sort algorithm that sorts this integer array of 10 elements:

```
int array[10] = {1,3,4,5,2,3,4,5,0,8};
```

Your output should resemble the following:



```
E:\Dropbox\My Dropbox\Lecturing\Impera...  
Original array : 1 3 4 5 2 3 4 5 0 8  
Sorted array : 0 1 2 3 3 4 4 5 5 8  
Press any key to continue . . .
```

Hints

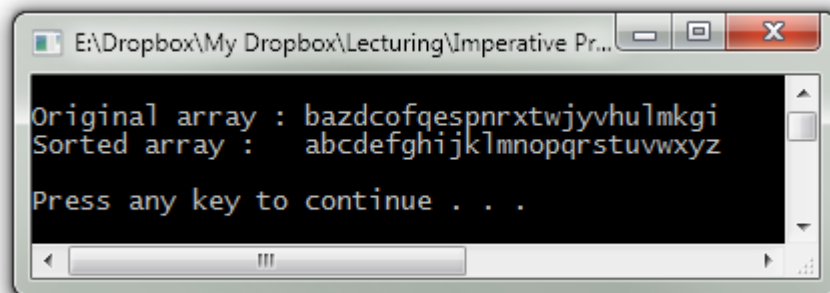
- You can use two for-loops for this exercise
- You will need to define a single temp value that can be used to temporarily hold data while swapping two items
- Assume the worst case scenario, and that you need to perform a bubble sort iteration n number of times (where n is the size of the array)

Exercise 2 - Alphabetical

Modify your algorithm from Exercise 1 so that it works with characters. Your algorithm should take the following array and sort it so it forms the alphabet:

```
char array[27] = "bazdcofqespnrxwtjyvhuImkgi";
```

Example output:



Exercise 3 - Modified Bubble Sort

It can be quite wasteful to iterate through an array once for every item in the array, as we are assuming the worst case scenario where the array needs to be completely reversed - what if we only need to swap two items in the array to sort it?

A better version of bubble sort, called the modified bubble sort, uses a simple flag to check if the array is sorted every time the algorithm loops through. The pseudo code looks like this:

1. Set flag to 0
2. Loop through each element in the array, from start to end
3. Compare each pair of elements
4. Switch the two elements positions if necessary (Set flag to 1)
5. At the end of the loop

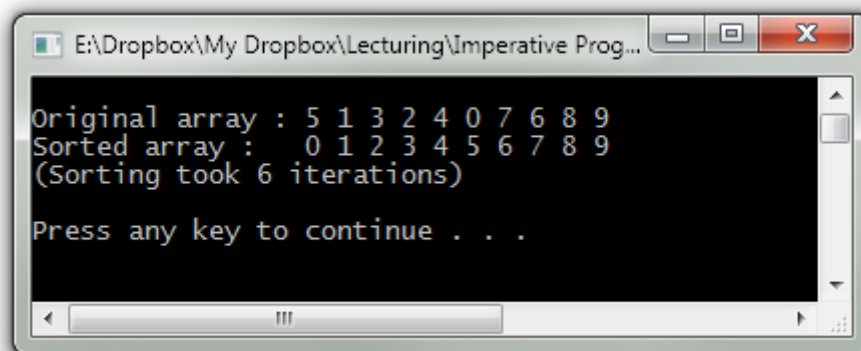
If flag is 1, go back to 1 (repeat this process)

If flag is 0, break (the list is now sorted)

Modify your bubble sort algorithm from Exercise 1 so that it uses the modified bubble sort algorithm. Check your algorithm works using the following array, which should require 6 iterations though before it is known to be sorted.

```
int array[10] = {5,1,3,2,4,0,7,6,8,9};
```

Your output should resemble the following:



```
E:\Dropbox\My Dropbox\Lecturing\Imperative Prog...
Original array : 5 1 3 2 4 0 7 6 8 9
Sorted array : 0 1 2 3 4 5 6 7 8 9
(Sorting took 6 iterations)

Press any key to continue . . .
```

Hints

- A flag can be represented using an integer (0=no, 1=yes). In C99 or C++ we could use the boolean true/false, but it's the same process
- You shouldn't need to define a counter to know how many times you've iterated

Exercise 4 – Selection sort

Selection sort is another simple search algorithm. It works by selecting the smallest element in the array, and placing it at the front of the array. The process is then repeated for every other element in the array, placing them in the second, third index etc.

This sorting algorithm can be faster than bubble sort, simply because with every iteration it is looking at a smaller and smaller part of the array (there's no need to iterate over parts of the array already sorted).

For this exercise, your task is to implement a selection sort program.

Hints

- Using a nested for loop based on x and y, setting the initial condition of the inner for-loop to be y = x allows you to iterate over smaller and smaller sections of the array.

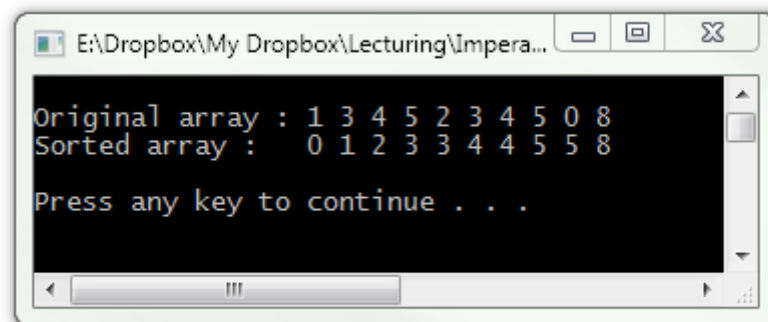
Challenge – Insertion sort

Insertion sort is one of the fastest in-place sorting algorithms. It works similarly to selection sort, but is more efficient especially in small and almost sorted data sets. Insertion sort is also the closest search method to the one most humans would naturally use.

The pseudo-code is as follows:

1. Loop through the unsorted elements in the list
2. Compare each element with sorted part of list until you find where it fits
3. Insert the element at this position, and move all elements between old and new indices by 1

Your task for this challenge exercise is to write an insertion sort algorithm. When complete, your output should resemble the following:



```
E:\Dropbox\My Dropbox\Lecturing\Impera...  
Original array : 1 3 4 5 2 3 4 5 0 8  
Sorted array : 0 1 2 3 3 4 4 5 5 8  
Press any key to continue . . .
```

Appendix

[Online C Programming Resources](#)

[Complete C Reference Library](#)

C Programming IDE's

[Dev-C++](#) (Windows)

[Code::Blocks](#) (Windows, Mac, Linux)

[Visual Studio/C++ Express](#) (Windows)

[Netbeans C/C++](#) (Windows, Mac, Linux)

[Codelite](#) (Windows, Mac, Linux)



PRIFYSGOL
BANGOR
UNIVERSITY