Mohriz Hassan
12301385

Sorting Algorithms

In this report we will discuss 3 different types of sorting algorithms. What is a sorting algorithm?

Sorting is ordering a list of objects. We can distinguish two types of sorting. If the number of objects is small enough to fits into the main memory, sorting is called *internal sorting*. If the number of objects is so large that some of them reside on external storage during the sort, it is called *external sorting*. **(Reference 1).**

"The trouble with organizing a thing is that pretty soon folks get to paying more attention to the organization process than what they're organizing for."

Laura Ingalls Wilder

Selection sort, bubble sort and quicksort are common methods to order arrays of numbers. We will look at each of these to understand how they rearrange arrays to give the desired output.

For learning purposes, we will investigate how the following array is sorted using each method. (see attached .cpp file for defined SIZE). In this case our array is set so size 7.

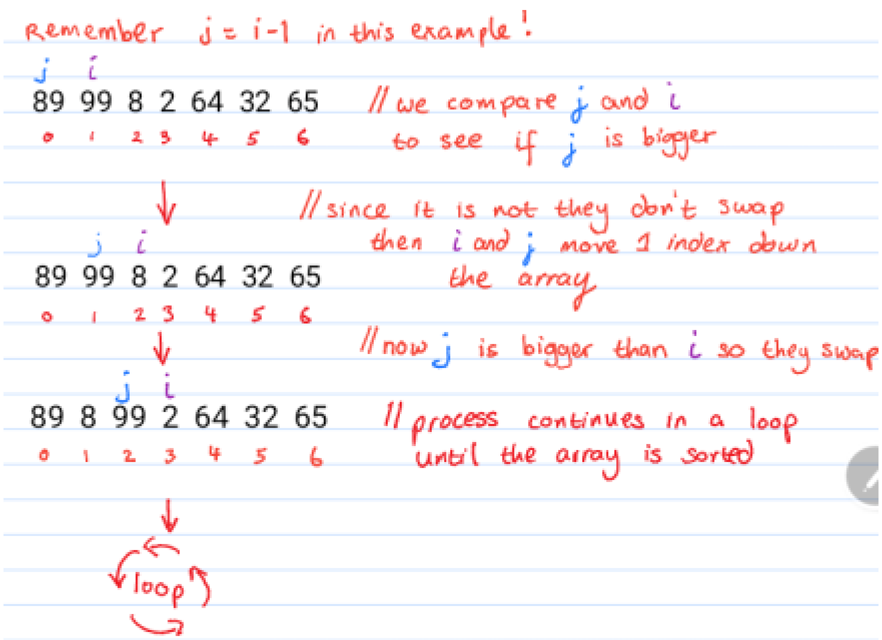
89 99 8 2 64 32 65

The key to effectively sorting anything is choosing the right method of implementation!!!

(reference 2)

Bubble Sort

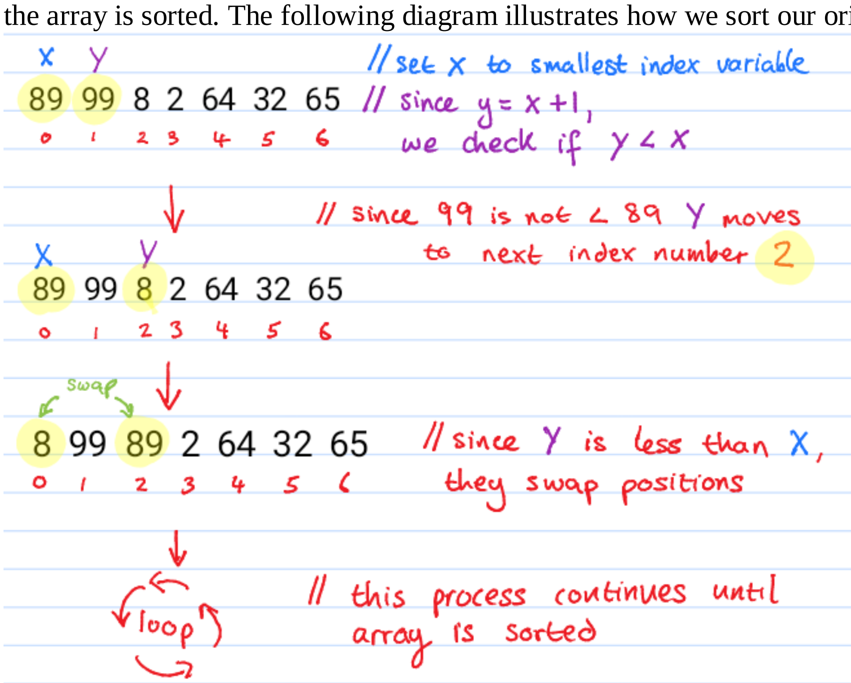
Bubble Sort is a straightforward sorting algorithm and is probably the most basic. It works by repeatedly exchanging adjacent elements, if necessary. It then continues to loop around the array and when no more exchanges are required, the file is sorted. The following diagram illustrates how we sort our original array.
For convenience we will say that the j = [i-1] in the if(bubbleSortArray[i]<=bubbleSortArray[i-1]) of the bubble sort function used in the code attached. This will allow easier illustration below.



This algorithm has several advantages. It is simple to write, easy to understand and it only takes a few lines of code. The data is sorted in place so there is little memory overhead and, once sorted, the data is in memory, ready for processing. The major disadvantage is the amount of time it takes to sort. The average time increases almost exponentially as the number of table elements increase. Ten times the number of items takes almost one hundred times as long to sort.

Selection Sort

When ordering an array (ascending order) Selection Sort will allocate the lowest element to the smallest index of the array. The second smallest element is then allocated to the second smallest index of the array. Generally, this process is looped over and over until the array is sorted. The following diagram illustrates how we sort our original array.



The Selection Sort programme written can be found in the attached code file. The void **selectionSort(int arraySelection[])** function implements the above steps perfectly and returns a sorted array in ascending order.

We may be concerned about the efficiency of our algorithm and its implementation as a program. The efficiency of an algorithm depends on the number of major computations involved in performing the algorithm. The efficiency of the program depends on that of the algorithm and the efficiency of the code implementing the algorithm.

The Selection Sort spends most of its time trying to find the minimum element in the "unsorted" part of the array. It clearly shows the similarity between Selection sort and Bubble Sort. Bubble Sort chooses the maximum remaining elements at each stage, but wastes some effort imparting some order to the "unsorted" part of the array thus Selection Sort is a superior sorting method.

Quick Sort

Quick sort works by

1. Picking a partition point somewhere in the sortable range.
2. Move the smaller values left and larger values right (of the partition point)
3. Repeat this process for the unsorted range to the left and right

The quick sort written in the attached code takes an array of ints and two indexes. It will not sort the whole array, but only the part of it between the two indexes, ignoring anything that is outside them. This means the same function can sort the whole array if you pass the first and last indexes, or just a sub array if you pass a left value that is not the index of the first element of the array and/or a right value that is not the index of the last element.

The pivot used is the central element (it could as well have used any other element). It partitions the array into the less than (or equal to) pivot subarray and the greater than (or equal to) pivot subarray, leaving an element equal to the pivot between the two partitions. Then it recursively calls itself to sort the two partitions, but only does it if it is necessary (hence the ifs before the recursive calls).

Quick sort is an efficient algorithm for larger sets of data, it is less efficient that other sorting algorithms on smaller sets of data.

Once we run our original array through each sort we get the following table of results! (outputted from the code)

```
----------------------------------------------------------------------------
                               RESULTS

Array before sorting:    89      99      8       2       64      32      65

Sorted via Bubble Sort:  2       8       32      64      65      89      99
Sorted via Select Sort:  2       8       32      64      65      89      99
Sorted via Quick Sort:   2       8       32      64      65      89      99

Counter for BubbleSort:    30
Counter for SelectionSort: 37
Counter for QuickSort:     13
----------------------------------------------------------------------------
```

Algorithm Name	***Big-O Complexity***		
	Best	***Worse***	***Average***
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Quicksort	$O(n*\log(n))$	$O(n^2)$	$O(n*\log(n))$

References

Reference 1: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Sorting%20Algorithms/sorting.html

Reference 2: http://www.csc.kth.se/~snilsson/fast-sorting/