Jason Collier 214008258 WRA201

Sorting Algorithms: Comparing Sort Times in Various Circumstances

Abstract

The objective of this experiment was to determine which algorithms (selection sort, recursive selection sort, double-selection sort and bucket sort) worked best under different circumstances. To have made conclusions, processing times of the algorithms, based on the range of circumstances, were measured and plotted versus array lengths on four graphs, one representing a general case and three others representing possible worst case scenarios for the four algorithms. Based on these results, it was concluded that for arrays of all types, for arrays smaller than two thousand five hundred elements, all four algorithms have similar processing times. As the arrays get bigger, especially for arrays bigger than seven thousand five hundred elements in size, the bucket sort algorithm is most efficient. The algorithms are more efficient on already sorted (ascending) arrays, followed by reversed arrays, then arrays full of duplicates and, lastly, by randomly ordered arrays.

Introduction

All programmers know that there are always many ways to obtain a certain solution. This fact is no different when it comes to sorting lists. Sorting algorithms are important, both in in code and in everyday life. Sorted lists, whatever the type, make it faster and easier to locate items. The right algorithm must be used for the right type of list to speed up the process further.

Four algorithms are discussed here. They are the selection sort algorithm, the recursive selection sort algorithm, the double-selection sort algorithm and the bucket sort algorithm. Because the different algorithms have different advantages and disadvantages, their processing times are recorded as accurately as possible on different types of arrays (that will represent general and worst case scenarios) of different sizes.

These results are discussed and conclusions are made as to which algorithms work best under specific circumstances.

Related Work

The four algorithms described in this report are all well researched. The sections below give a description of each algorithm, give the performance of each algorithm and detail the advantages and disadvantages of each algorithm according to other authors. The performance will be described using Big O notation, which, according to Bell (2009), describes the performance at the worst case scenario and can represent the execution time or memory used by an algorithm.

Basic Selection Sort

The selection sort algorithm works by finding the smallest item in the unsorted portion of the list (which is the whole list for the first iteration) and replaces it with the first item of the unsorted portion for an ascending list, unless that first item is in fact the minimum (Lakra and Divya, 2013).

According to Mishra and Garg (2008), this algorithm has a worst case performance of order N squared, and an average case performance of order N squared too. It is thus very slow when processing big lists, one of its disadvantages, and should only be used on data structures that can fit into the processor's main memory.

Compared to the three other basic sorting algorithms, this algorithm is faster than the bubble sort algorithm, but slower than the insertion sort algorithm. The main advantage of a selection sort algorithm is its simplicity (Lakra and Divya, 2013).

Recursive Selection Sort

The recursive selection sort runs in much the same fashion as its iterative counterpart. Sahgal (2015) says that, because programming languages spend more time maintaining the call stack when executing recursive algorithms than performing necessary calculations, there is more overhead associated. This means that recursive algorithms usually take longer to run than iterative algorithms.

The disadvantage to using the recursive version of the selection sort algorithm is the extra time it takes to process lists, which is especially notable on lists containing more than two thousand five hundred elements. The advantage is that recursive algorithms generally look neater and are easier to follow for other programmers (Sahgal, 2015).

Double-Selection Sort

The double-selection sort algorithm selects two elements at either end of the list, then searches the rest of the list for a minimum and maximum element, thereafter swapping them with each of the end values, depending if the list is being sorted in ascending or descending order. The algorithm then selects the second and the second last elements and searches the rest of the unsorted portion for another minimum and maximum value, swapping them with the second and second last values. This process continues with the unsorted portion decreasing by two on each pass, while the sorted portions on either end increase by one each on each pass (Lakra and Divya, 2013).

The double-selection sort uses $3\frac{N^2}{8} + O(N)$ comparisons versus the selection sort algorithm which uses $\frac{N^2}{2} + O(N)$ comparisons, for a theoretical twenty-five percent improvement in performance. The actual performance increase is unfortunately not as big (Lakra and Divya, 2013).

Apart from the obvious advantage that comes with the performance increase, Lakra and Divya (2013) say that big improvements are only practically seen in large lists. Unfortunately, even though this algorithm is more efficient than its basic counterpart for big lists it is still not efficient enough for its use to be justified on large lists (of two thousand five hundred elements and more).

Bucket Sort

According to Pollice, Selkow and Heineman (2008), the bucket sort algorithm takes an input list of integers and puts them into a certain amount of "buckets". Each bucket represents a range of integers, so each element in the list is placed into a bucket, in the correct position so each buckets remains ordered, based on which range it falls into.

This algorithm has a worst, best and average case performance of O(N) (Pollice, Selkow and Heineman, 2008), making this algorithm, theoretically, the fastest sort of the four spoken about in this report. It sorts lists with more evenly distributed data faster than lists with an uneven distribution of data.

The advantage of using this algorithm is its speed of sorting large lists, especially lists with a uniform distribution of data. Unfortunately, the bucket sort algorithm, as a result of the less taxing processing, uses more memory when sorting, which can be disadvantageous. It can also only be used on integer lists.

Experimental Design

Four types of arrays (*ArrayLists* in particular) are used to test these algorithms (selection sort, recursive selection sort, double-selection sort and bucket sort), the first being a randomly sorted array, the second an array that is already sorted in ascending order, the third an array sorted in reversed order, and lastly, an array full of duplicates. The first mentioned array represents a general case, while the other three types represent possible worst case scenarios for the specified algorithms.

Each element in each array is an alpha-numeric element containing six numerical values and three characters, in any order.

For each type of array, the above specified algorithms are tested on arrays with different amounts of elements (eleven arrays of different sizes between ten and fifty-thousand). The time taken (using the *Stopwatch* class), for each algorithm to process each array is recorded and plotted versus the amount of elements in each array. Four different graphs are created; one for each type of array.

Based on these, conclusions can be made with regard to which algorithms should be used under specific circumstances, and the advantages and disadvantages of the different algorithms are deduced. These findings are compared to the related work.

Results

Random Arrays

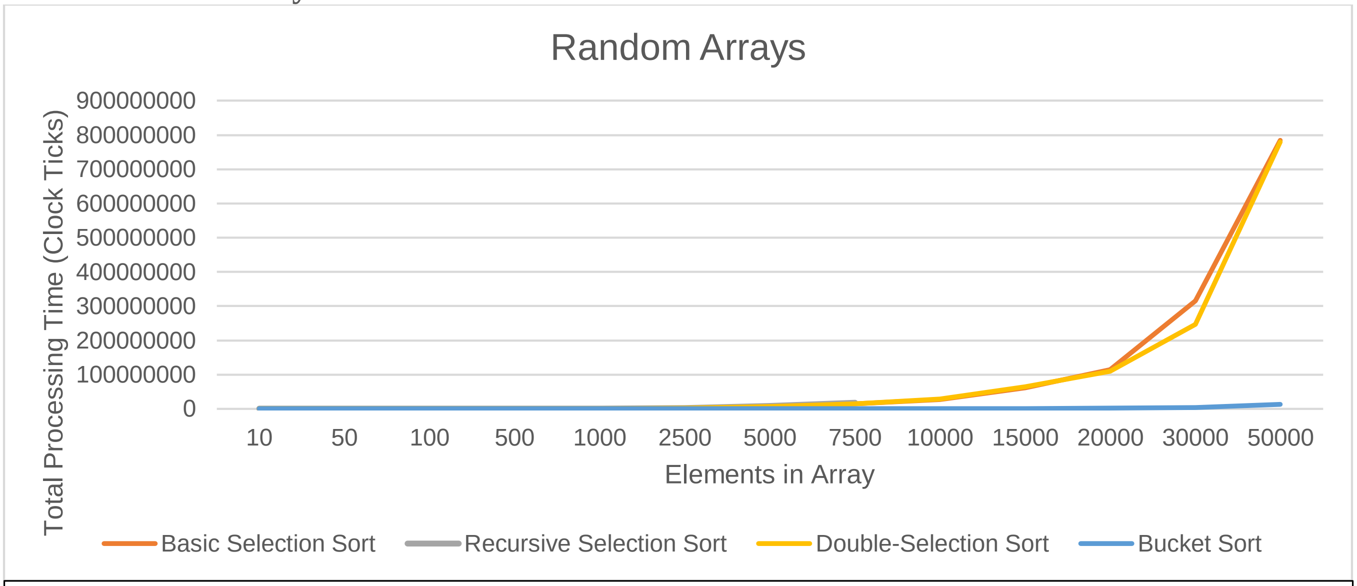

Figure 1: The total processing time for the specified algorithms on random arrays

All four of the sorting algorithms (basic selection sort, recursive selection sort, double-selection sort and bucket sort) perform worst when sorting completely random arrays, as opposed to sorting already sorted (ascending) arrays, arrays sorted in reverse order or arrays full of duplicate values, as can be seen from figures 1 to 4.

Comparing these four algorithms based on random arrays, it is noted, from figure 1, that the bucket sort algorithm is quickest, with the double selection sort way off in second, followed closely by the basic selection sort algorithm and then by the recursive selection sort algorithm.

According to figure 1, the slower three algorithms are very efficient for arrays consisting of up to about 2500 elements. There is a slight increase in processing times for arrays consisting of between, roughly, 2500 and 7500 elements. For the basic selection sort algorithm and the double-selection sort algorithm, the processing time increases drastically for arrays consisting of more than 7500 elements.

The only slight increase in processing time of the bucket sort algorithm comes for arrays with more than 30000 elements.

Sorted Arrays

The selection sort algorithm, the recursive selection sort algorithm and the double-selection sort algorithm all sorted the already sorted arrays faster than any type of arrays. The bucket sort algorithm sorts already sorted lists faster than the other algorithms, but it takes longer to sort already sorted arrays than it does to sort any other type of array. The double-selection sort algorithm is the next fastest sorter for this case (except for an array of 50000 elements which is freak result), followed by the basic selection sort algorithm and then by the recursive selection sort algorithm. Figure 2 below depicts the above description.

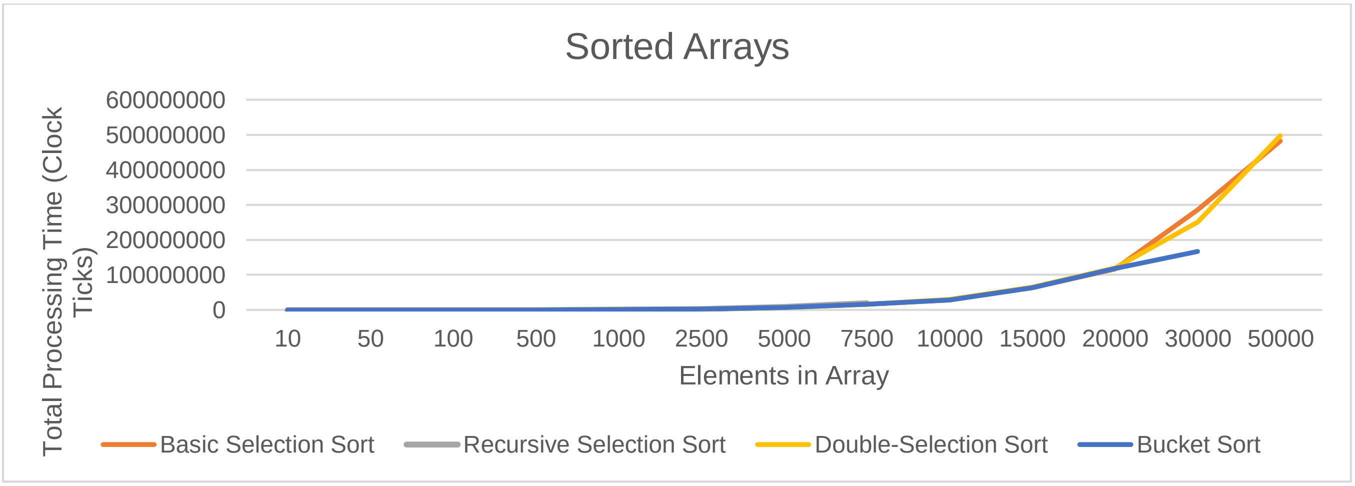

Figure 2: The total processing time for the specified algorithms on sorted (ascending) arrays

Reversed Arrays

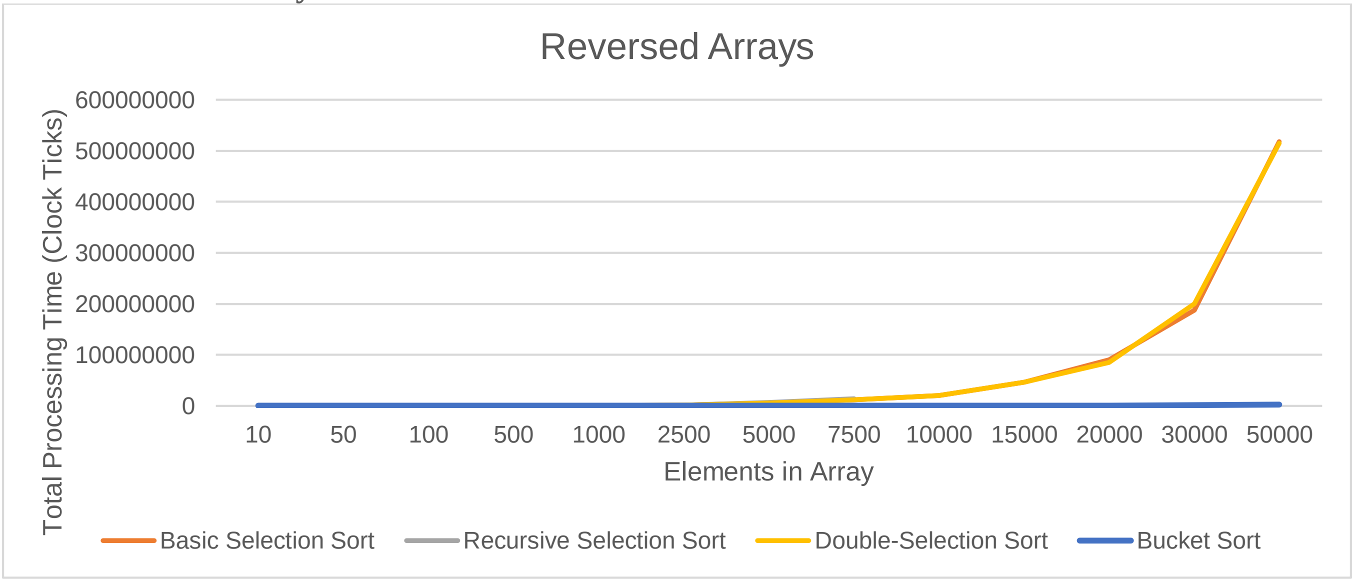

Figure 3: The total processing time for the specified algorithms on arrays sorted in reverse order

The slower three algorithms sort reversed arrays slightly slower than they sort ordered arrays, with the double-selection sort algorithm being slightly faster than the basic selection sort algorithm, followed by the recursive selection sort algorithm, as can be seen in figure 3.

There is not much difference in time taken for the bucket sort to sort a reversed array compared to a random array, but it is still the fastest of the four sorts for this case.

Arrays with Duplicates

The three slower algorithms take longer to sort arrays full of duplicates than they do to sort reversed and ordered arrays, though they do still sort arrays with duplicates faster than random arrays.

There is not much difference in time taken for the bucket sort to sort a reversed array compared to a random array, but it is still the fastest of the four sorts for this case. Figure 4 below depicts the above description.

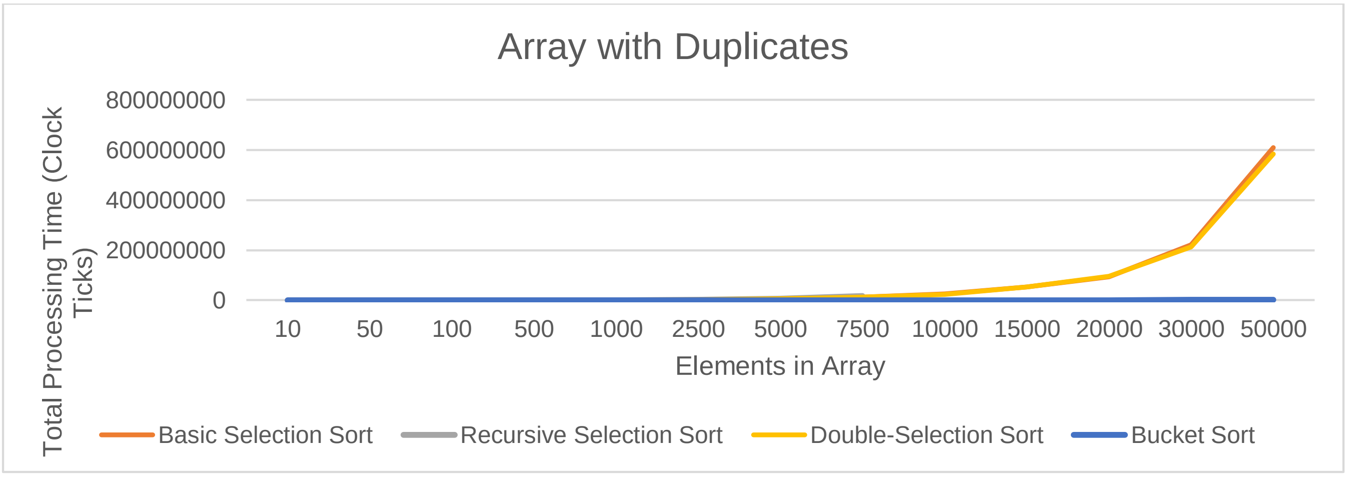

Figure 4: The total processing time for the specified algorithms on arrays full of duplicates

Discussion

The general case, using randomly sorted arrays, is used to compare the algorithms to one another, as this provides the fairest comparison based on their average performances (some algorithms will perform better or worse than their usual average performances when tested on worst case scenarios).

The recursive selection sort algorithm cannot process arrays bigger than seven thousand five hundred elements in size without throwing a stack overflow exception. It can still be seen that the recursive algorithm has the worst performance of all the algorithms. This observation is justified by Sahgal (2015) who says that recursive algorithms run slower than their iterative counterparts as a result of the extra overhead.

Both the selection sort and double selection sort algorithms are iterative in nature and are therefore better performers than their recursive counterpart. The double selection sort algorithm sorts from both ends of the list, whereas the basic selection sort algorithm only sorts from one end, making the double selection sort the most efficient of these three algorithms. The results support the theoretical performances given in the related work.

The bucket sort algorithm is the fastest of all four algorithms in all cases. The bucket sort algorithms perform at a similar efficiency in all cases (justifying the theoretical performance of O(N) given by Pollice, Selkow and Heineman (2008), except on sorted arrays. This exception is likely due to the fact that, to allow bucket sort to work on *strings*, the first character of each string is treated as a numerical value and this value is used to place each string into its correct bucket. This work-around did not affect the other cases much, but in the case of the already sorted arrays, the first character of each string in the array was the same (these strings were manually generated like this), meaning that the elements were not properly distributed into the buckets, greatly slowing the sort time.

Comparing the general case to the three possible worst case scenarios, it can be seen that all four algorithms are slowest when sorting randomly arranged arrays (all be it a small difference for the bucket sort algorithm). This is due to the random distribution of elements in these arrays. Other than for the bucket sort algorithm, the other three perform best on already sorted arrays because less swaps need to be made. They perform slightly slower on reversed arrays because swaps have to be made on every iteration and slowest on arrays full of duplicates.

Conclusion

For all cases, the recursive selection sort algorithm is the slowest performing, but easiest to understand making it useful for very small lists. This algorithm is followed closely in performance by the basic and double-selection sort algorithms, which are also best used on small lists. The basic selection sort, the simplest and most widely understood of these three algorithms, is the best option for sorting small lists. The bucket sort algorithm is the best performer of all these algorithms, but because of its complexity and memory requirements, should only be used on large lists (of more than 2500 elements).

References

1. Bell, R., 2009. *A beginner's guide to Big O notation.* [online] Available at: <https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/> [Accessed 01 May 2016].
2. Mishra, A.D. & Garg, D., 2008. Selection of Best Sorting Algorithm. *International Journal of Intelligent Information Processing*, [online] Available at: <http://gdeepak.com/pubs/Selection%20of%20best%20sorting%20algorithm.pdf> [Accessed 01 May 2016].
3. Lakra, S. & Divya, 2013. Improving the performance of selection sort using a modified double-ended selection sorting. *International Journal of Application or Innovation in Engineering & Management*, [online] 2(5) Available at <http://www.ijaiem.org/Volume2Issue5/IJAIEM-2013-05-31-098.pdf> [Accessed 01 May 2016]
4. Sahgal, V. 2015. *Recursion Versus Iteration*, [online] Available at: <http://www.programmerinterview.com/index.php/recursion/recursion-versus-iteration/> [Accessed 02 May 2016].
5. Pollice, G., Selkow, S. & Heineman, G.T., 2008. *Algorithms in a Nutshell*. California: O'Reilly Media, Inc.