Sorting an array

Pontus Hector

Fall 2022

Introduction

This is the fourth assignment in the course ID1021. In this task one explores different ways of implementing algorithms that sorts arrays and their difference in time efficiency.

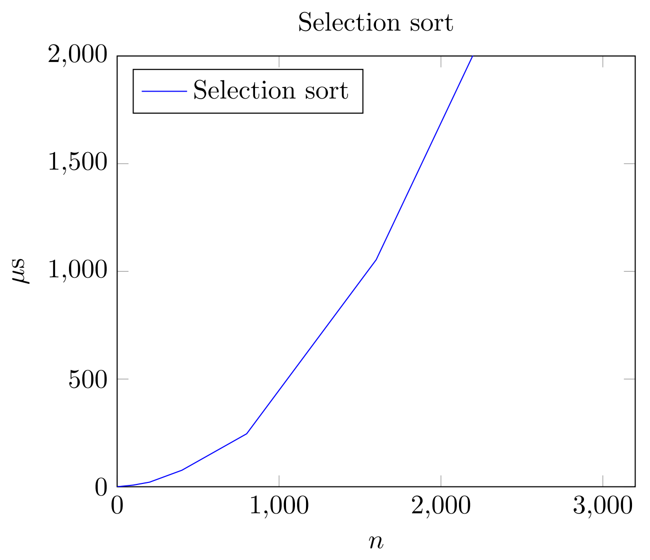
Selection sort

Selection sort is a sorting algorithm where the first sequential element is compared to the rest of the array and the smallest element in the array is then swapped with the first element. Then the second sequential element is compared to the rest of the unsorted array, and so on, repeatedly finding the smallest element from the unsorted part and putting it at the end of the sorted part of the array. The following measurements were obtained through benchmarking:

Iterations	**Array size (n)**	**Average runtime**
1000000	100	7.76μs
1000000	200	21.3μs
1000000	400	76.7μs
1000000	800	246μs
1000000	1600	1054μs
1000000	3200	3595μs

Table 1: Selection sort

These values are then graphically plotted to enhance the picture of what time complexity this algorithm is of.



These measured values confirms that the time complexity is quadratic (O(n^2)). This sorting algorithm is simple to implement but considerably inefficient on larger sized arrays.

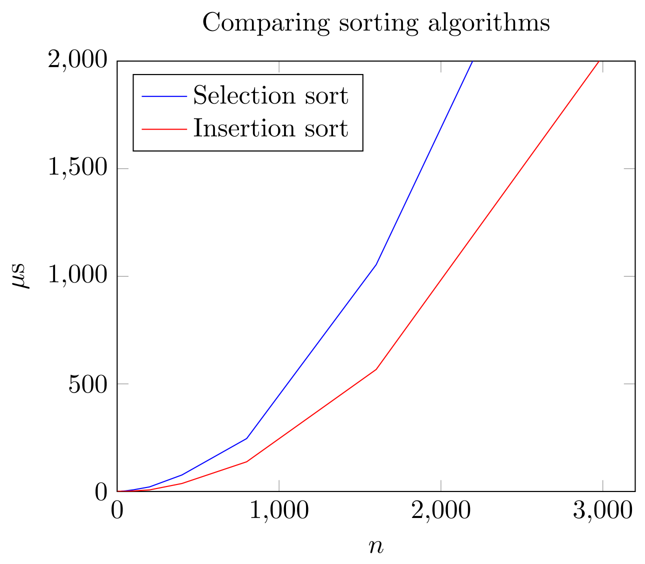
Insertion sort

The insertion sort algorithm sequentially selects one element and compares it to the previous elements in the array. Each previous element that is larger than the currently selected element is shifted right, and the selected element is *inserted* where it temporarily belongs. This is repeated until all the elements of the array has been compared and inserted on the sequentially correct place. The following measurements were obtained through benchmarking:

Iterations	**Array size (n)**	**Average runtime**
1000000	100	2.43μs
1000000	200	7.28μs
1000000	400	37.1μs
1000000	800	138μs
1000000	1600	567μs
1000000	3200	2232μs

Table 2: Insertion sort

These values are then graphically plotted to enhance the picture of what time complexity this algorithm is of.



These measured values confirms that the time complexity is quadratic (O(n^2)). This sorting algorithm performs better than the selection sort, however the time complexity of this algorithm is yet quadratic which renders it rather inefficient for larger sized arrays.

Merge sort

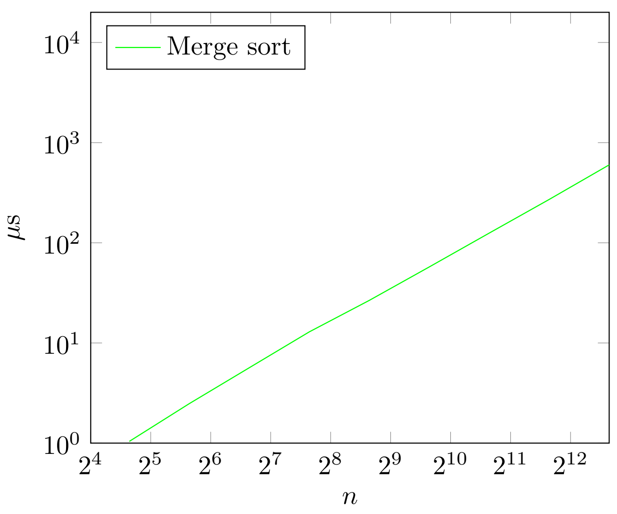
The merge sort algorithm is a "divide and conquer" sorting algorithm that continuously divide and break down the array into pieces (smaller arrays) until it cannot be further divided. These pieces are then sequentially built back up in a sorted order and *merged* into larger and larger sorted arrays until a finally sorted and fully sized array has been built back up. The following measurements were obtained through benchmarking:

Iterations	**Array size (n)**	**Average runtime**
1000000	100	5.66μs
1000000	200	12.9μs
1000000	400	26.6μs
1000000	800	57.4μs
1000000	1600	125μs
1000000	3200	271μs

Table 3: Merge sort

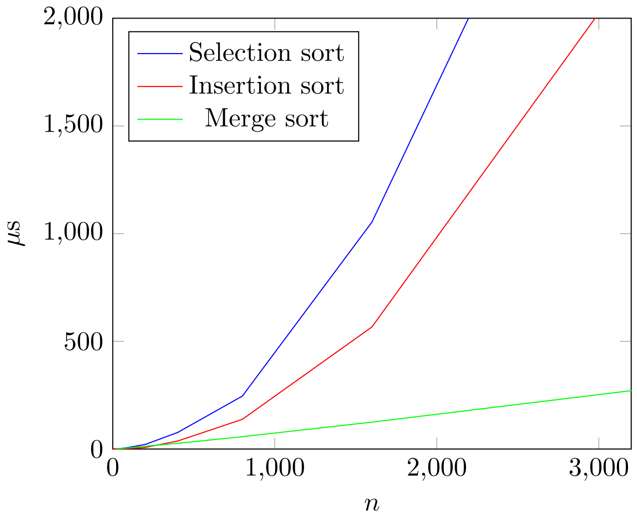
These values are then graphically plotted to enhance the picture of what time complexity this algorithm is of.

Merge sort



At first glance this algorithm may seem to be of linear time complexity, but when considering the actual measurements one can determine that the time complexity is O($nlog(n)$). Excluding the linear part from the measured values results in a rest term that is not constant, but rather increasing in correlation to the linear part and therefore is the time complexity O($nlog(n)$).

Comparing sorting algorithms



Compared to the previous sorting algorithms, merge sort performs considerably better when operating on larger arrays. The few cons it has is that its a bit more difficult to implement and that it has plenty more lines of code than the previous algorithms, which makes it perform worse than both of the previous algorithms on smaller sized arrays.