# 2XB3 Lab3

Yi Luo
400254211
luoy94@mcmaster.ca
L02

Frank Yang
400243777
yangf51@mcmaster.ca
L02

Harrison Chiu
400261400
chiuh@mcmaster.ca
L02

February 2, 2021

**Abstract**

This paper analyzes different time complexities of quicksort. Specifically it looks into the time complexities of various quicksort implementations, including multiple pivots, different pivots, small lists, and worst case performance.

## Quicksort Inplace

## Quicksort Multi-pivot

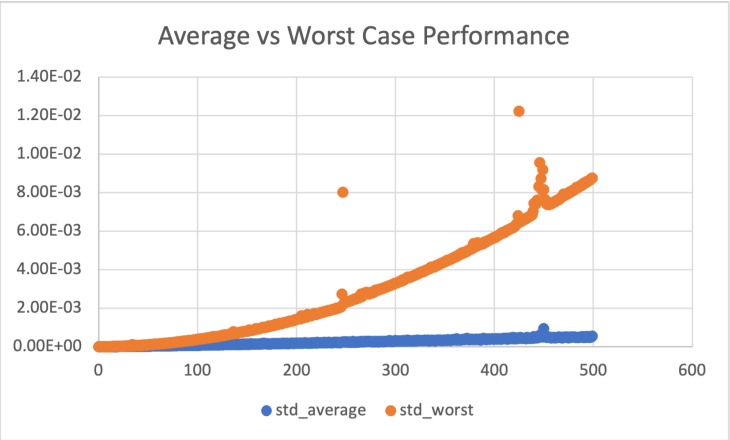
## Worst case

The worst complexity of quicksort is $n^2$. It happens under the following circumstances. The given array is already sorted or reversely sorted with the leftmost or the rightmost element chosen as the pivot. The given array's elements are the same.

All of them would lead to a situation where the divided arrays are totally unbalanced such that one array has only one element while the other has $n-1$ element with $n$ being the size of the parental array. As a result, the time complexity would be as such:

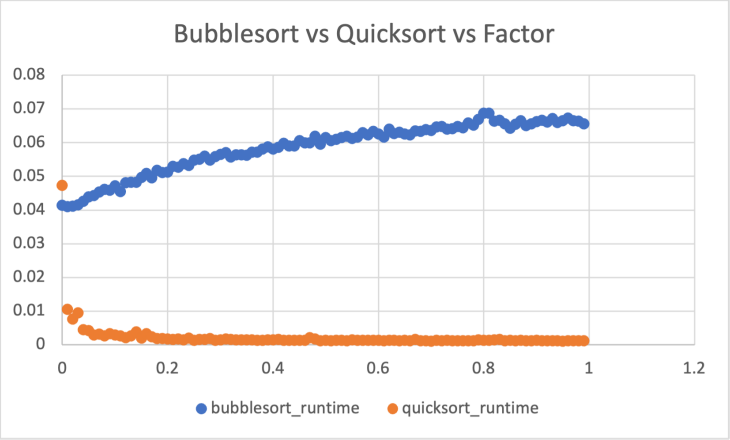
$$n + (n-1) + (n-2) + (n-3) + \ldots + 2 = n * (n+1)/2–1 = \Theta n^2$$

A graph is attached showing the average case performance vs the worst case performance vs n.



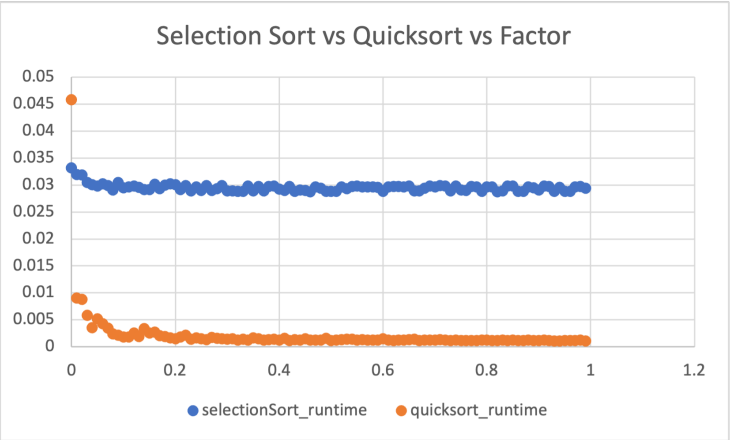

The function `create_near_sorted_list` creates near-sorted-list based off some input factor. The lower the factor, the closer to being sorted the created list is.

Bubble sort will out-perform quicksort when a given list is close to being sorted since it only takes one or few iterations for the bubble sort to terminate, yielding a linear complexity($\Theta n$). Attached is a graph comparing their performance with factor ranging from 0 to 0.99. Note that the lists sorted are of length 996 instead of 1000 as specified since it would otherwise generate `RecursionError`.



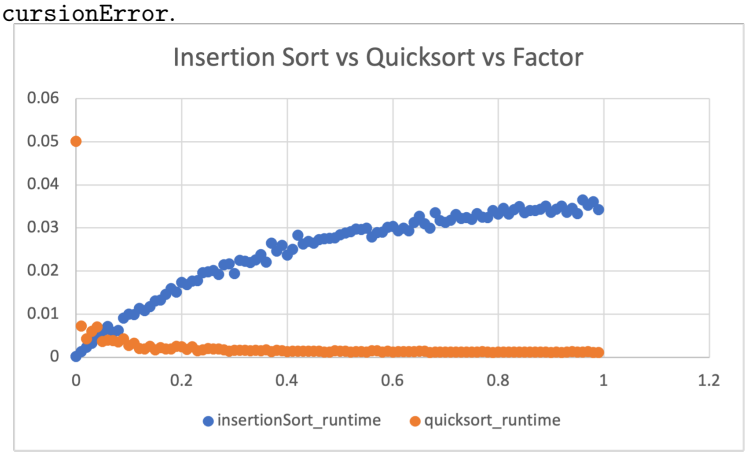

Bubble sort out-performs only when factor is extremely close to zero.

Selection sort always has a complexity of $n^2$. Attached is a graph comparing their performance with factor ranging from 0 to 0.99. Note that the lists sorted are of length 996 instead of 1000 as specified since it would otherwise generate `RecursionError`.




Selection sort out-performs only when factor is extremely close to zero.

Insertion sort will out-perform quicksort when a given list is close to being sorted, yielding a linear complexity($\Theta n$). Attached is a graph comparing their performance with factor ranging from 0 to 0.99. Note that the lists sorted are of length 996 instead of 1000 as specified since it would otherwise generate `RecursionError`.




Insertion sort out-performs when factor approximately is smaller than 0.05.

## Small lists