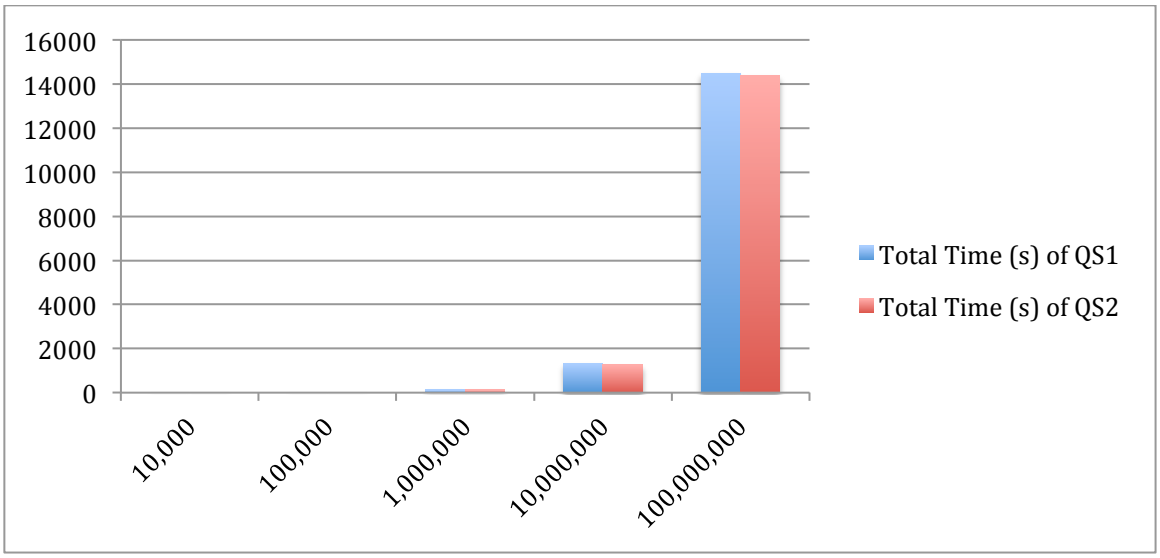
Paul Diaz
CS 146
Project1.B: Quicksort

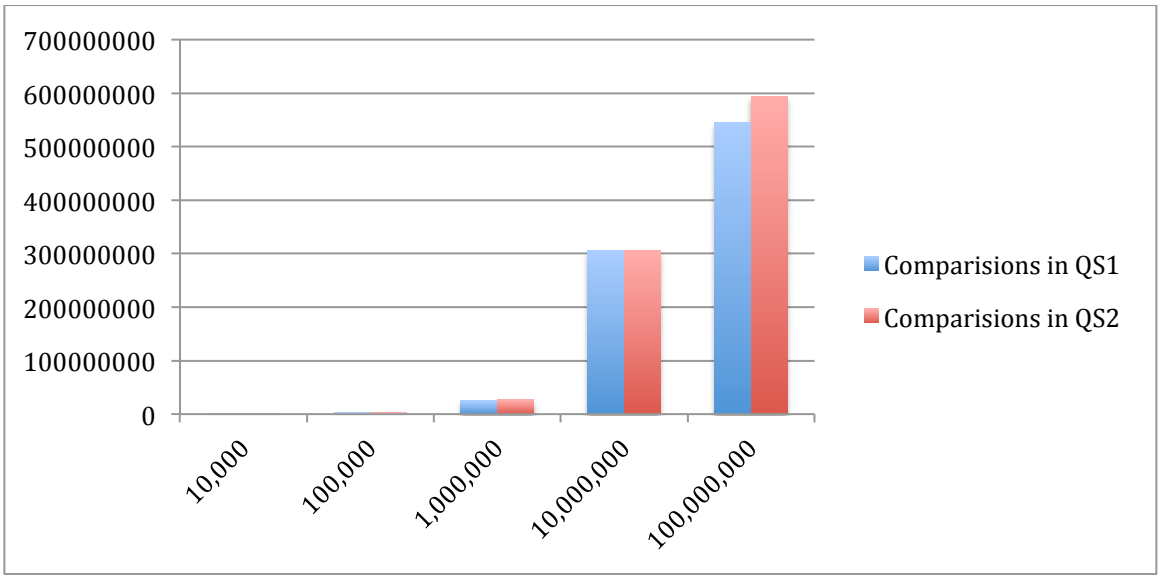
## Introduction

The purpose of this project is to test the behavior of two types of quicksort; regular quick sort, where the last index will be the pivot, and a quick sort where we randomly select the pivot before partitioning the array. Moreover, on this project, the numbers of comparisons of array elements are also collected.

| Size of Array | 10,000 | 100,000 | 1,000,000 | 10,000,000 | 100,000,000 |
|---|---|---|---|---|---|
| Total Time (s) of QS1 | 6 | 22 | 122 | 1317 | 14502 |
| Total Time (s) of QS2 | 7 | 21 | 122 | 1264 | 14398 |
| Comparisons in QS1 | 167826 | 2120379 | 25225932 | 305272206 | 545656053 |
| Comparisons in QS2 | 158416 | 2129004 | 26676084 | 305448714 | 593466544 |




Note: Total time of function execution using method System.currentTimeMillis();




Note: Total number of comparisons

**Conclusion**:

Based on our table and graphs, we see that the execution time in quicksort1 and quicksort2 are very close; however, it appears that the execution time of quicksort2 is slightly faster than quicksort1. This result is expected because in quicksort1, worst case could happen where the selected pivot is the largest among the rest of the elements in the sub-array; where as in quicksort2, worst case could also happen but the probability of worst case from happening is slightly rarer that quicksort1. I expected that the more comparisons the program does, the more time it will take to execute the method; however, based on our table, it appears that this is not the case. In this project, we proved that by choosing the pivot randomly instead of a fixed pivot, the running time of quicksort is slightly better.