A. Introduction

This laboratory exercise required the implementation of merge sort and quicksort for sorting String and long elements in an array and the runtime for each sort for each element.

Container classes that implement quicksort and merge sort were implemented and the performance of these sorts are studied in this lab.

B. Design Approach

The lab required the creation of an abstract Person class with private instance fields as first name and last name. An abstract SortedList class was required to serve as a container for Person objects.

From the Person class, two classes, Worker and Contact were inherited from these classes. The Worker class had an extra field variable called an ID which serves a comparison element between Worker objects. The Contact class added and extra instance field of String type called number which was represented the number of the Contact object. The Contact object used names as a comparison element between Contact objects.

From the SortedList there are two classes that are inherited from it: the ContactList class which serves as a Contact container class and the WorkerList class which serves as a container for the Worker object.

A RandomStringGenerator class is created that generates 4-letter words as names, 8 digit numbers as ID, and generates phone numbers with use of methods. This class is essential in instantiating ContactList and WorkerList objects for testing.

The final class is the ExperimentController which tests all these sorts and writes the data into a .csv file, given parameters such as the file name for the data to be written into, the number of times for averaging data and the number of items to be sorted.

The container classes were implemented using Java's ArrayList object which ensured easy and fast access to object elements with the use of indexes. Sorting over such an object makes it easier compared to LinkedList objects, which uses references and thus access and retrieval of objects is much slower compared to ArrayList.

Insertion might be slower in ArrayList implementation but the gains in time saved in easy and fast access to object elements outweigh its cost of insertion.

C. Method

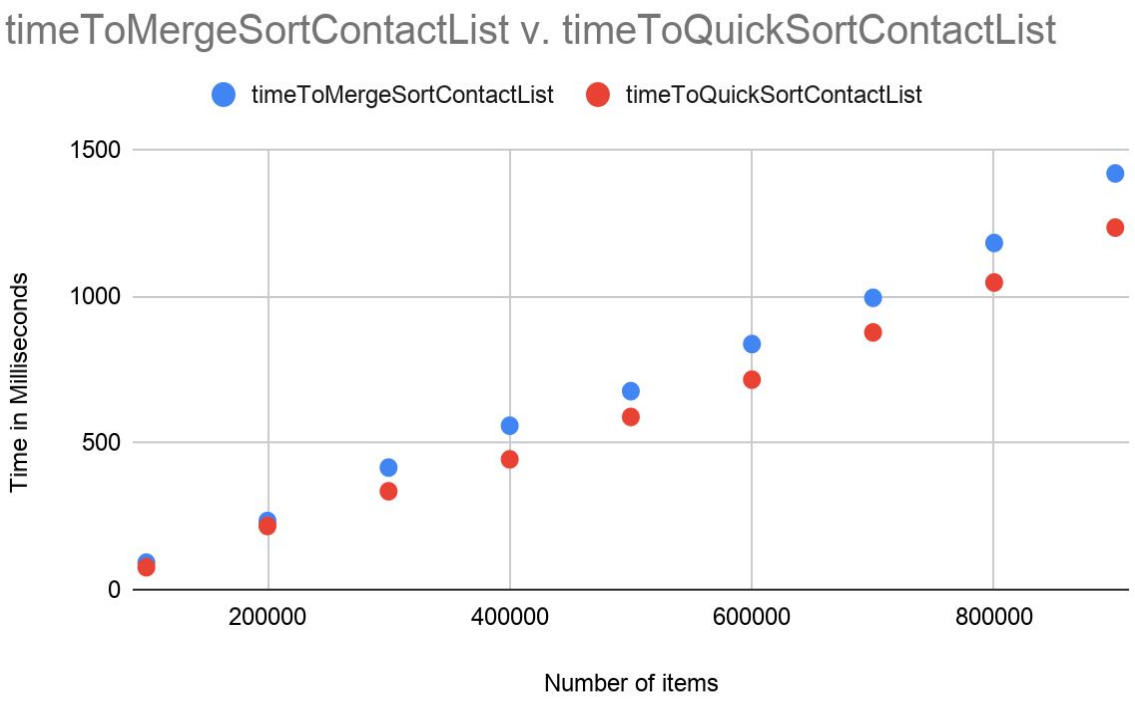
Runtime from experiments show the average runtime for both merge sort and quicksort as log(N) by O(N). Thus, low number of items to compare would be to low for the Java millis clock to capture. Thus, values for sorting were selected to be from 100,000 to 900,000. The same seed value was used by the RandomString Generator class to ensure both sorts are sorting through the exact number and objects in the container objects.

The experiment was run for an average of 20 times for each sort for each container. This is a good amount to average values to avoid atypical data points to skew collected data.

D. Data & Analysis

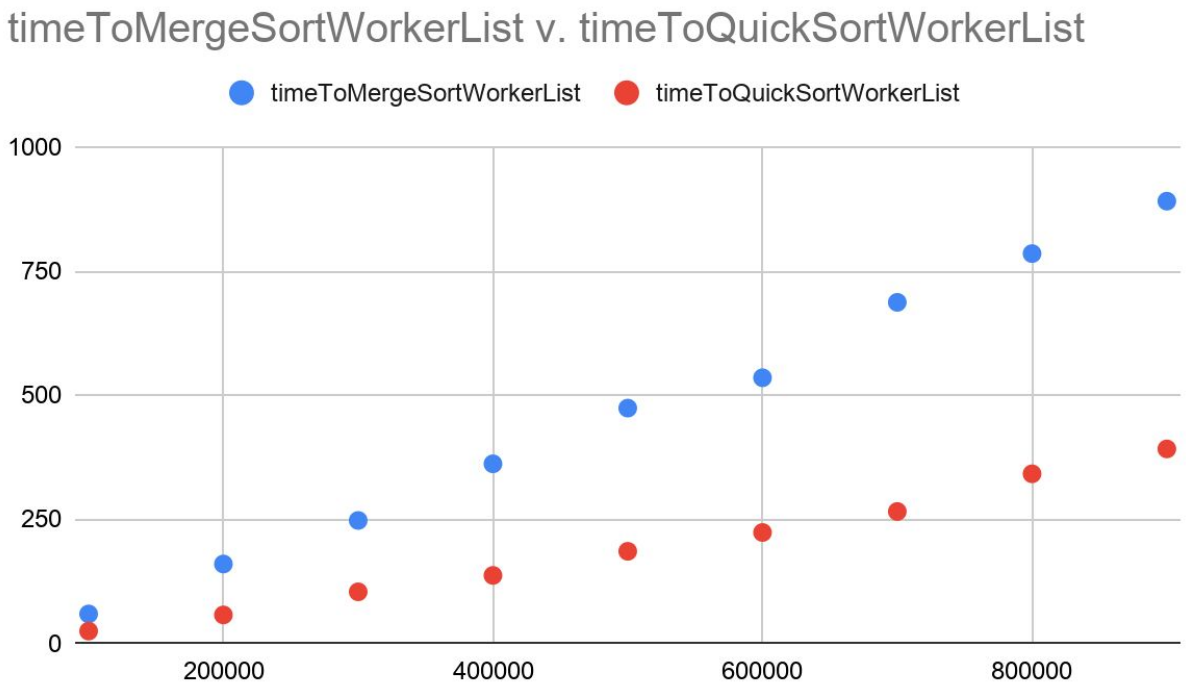
Implementation of merge sort and quicksort is different in ContactList and WorkerList due to the difference in comparison elements for both classes (String for Contact, long for Worker) thus, it would be erroneous in comparing sorts between container classes.

1. ContactList Class Container



For the ContactList container, merge sort and quicksort runtime are fairly close for small values of N. These values diverge as N increases. Quicksort algorithm was much sorting faster than merge sort for larger values of N.

2. WorkerList Class Container

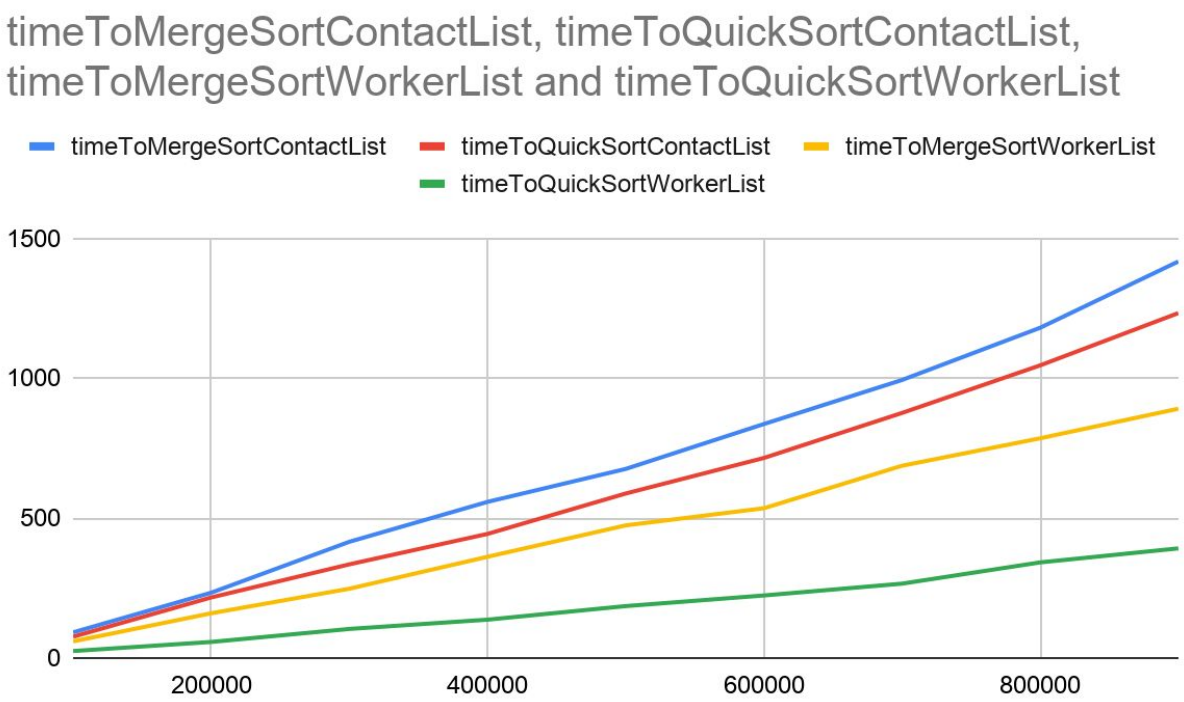


For the WorkerList container, the divergence of merge sort and quicksort runtime are very evident for low values of N. The runtime values diverge greatly as N increases. Quicksort algorithm was a much faster sorting algorithm than merge sort for larger values of N.

Sorting algorithms merge sort and quicksort have a big O(N) notation of log(N), but for the implementation of these sorts in the ContactList and WorkerList container classes, the runtime of these sorts diverge and quicksort seems a much faster sort algorithm than merge sort. Why the discrepancy?

The most likely reason might be the difference in the implementation of quicksort and merge sort. Merge sort uses space resources during sorting unlike quicksort. Storing and retrieval of these data from temporary spaces can consume time and add to the overall runtime.

3. Comparing all sorts



The merge sort and quicksort values for the WorkerList object is much more faster than both sorts for the ContactList sort. This might have to do with the Java API and how comparison is done.

For long comparison,

```
public static int compare(long x, long y) {
        return (x < y) ? -1 : ((x == y) ? 0 : 1);
    }
```

This comparison is much faster compared to the comparison of Strings which in the Java API is,

```
public int compare(String s1, String s2) {
            int n1 = s1.length();
            int n2 = s2.length();
            int min = Math.min(n1, n2);
            for (int i = 0; i < min; i++) {
                char c1 = s1.charAt(i);
                char c2 = s2.charAt(i);
                if (c1 != c2) {
                    c1 = Character.toUpperCase(c1);
                    c2 = Character.toUpperCase(c2);
```

```
                if (c1 != c2) {
                    c1 = Character.toLowerCase(c1);
                    c2 = Character.toLowerCase(c2);
                    if (c1 != c2) {
                        // No overflow because of numeric promotion
                        return c1 - c2;
                    }
                }
            }
        }
        return n1 - n2;
    }
```

Comparing the two blocks of code, comparison of longs is much more faster than Strings.

Why is the divergence between sorts for WorkerList is much more larger than Contact List?

The reason might be the difference in the implementation of Worker and Contact classes. Worker class objects have to store data of a larger size compared to Contact class objects. For instantiation, a Worker class object needs an 8-digit instantiation of field variable id. Though Contact class objects need to store Strings as field variables, in the ExperimentController class, the strings are of length of 4 compared to the large long value of the Worker class.

Storing and retrieving objects containing longs from temporary storage during merge sort for WorkerList is much more intensive than for retrieving strings of length 4 for ContactList. This explains the bigger divergence in sorts for WorkerList objects and ContactList objects.

E. Conclusion

Big O(N) notation does not account for the full picture of the runtime of an algorithm. It depends on the objects sorted, code implementation of user and language's source code implementation. Quicksort is the best candidate for sorting both WorkerList and ContactList classes, given the implementation of these classes is done using Java ArrayList class.

F. References

Lysecky, R. V., & Lysecky, F. V. (2013). *Data Structures and Algorithms*. Los Gatos, CA. Zyante Inc.

Oracle Corporation. (n.d.). Java API Long Source Code. Retrieved from http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/f92ab6dbbff8/src/share/classes/java/lang/Long.java.

Oracle Corporation. (n.d.). Java API String Source Code. Retrieved from http://hg.openjdk.java.net/jdk7u/jdk7u6/jdk/file/8c2c5d63a17e/src/share/classes/java/lang/String.java.

Oracle Corporation. (2018, October 6). Java API String. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/lang/String.html.

Oracle Corporation. (2018, October 6). Java API Long. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/lang/Long.html.