CECS 619
John Murray
Project 1

Summary

This study compared three sorting algorithms: Merge Sort and Quick Sort (with two variants). The two variants of Quick Sort differ in which element they choose for the pivot (the first element or the last element). Each algorithm was computed with an input list of size 2^n where n is any one of 4, 6, 8, 10, 14, or 16. Each algorithm is fed the same input list for size n and, for each n, 500 iterations were performed. On each computation the time was recorded and then averaged at the end.

The Code

The code was implemented in the Ruby programming language which results in a rather concise (as compared to most c-style languages) implementation. One important note is that, in the Ruby world, it is common to re-open and modify existing classes to add native-looking functionality. Both Merge Sort and Quick Sort were implemented in this fashion such that you could call them as:

```
a = [... random array …]
a.merge_sort
a.quick_sort
```

Where `a` is an instance of Ruby's `Array` class. One other Ruby-specific note is how comparisons are done. Ruby uses what is, informally, called the *rocket* operator (`<=>`) such that the code has the following effect:

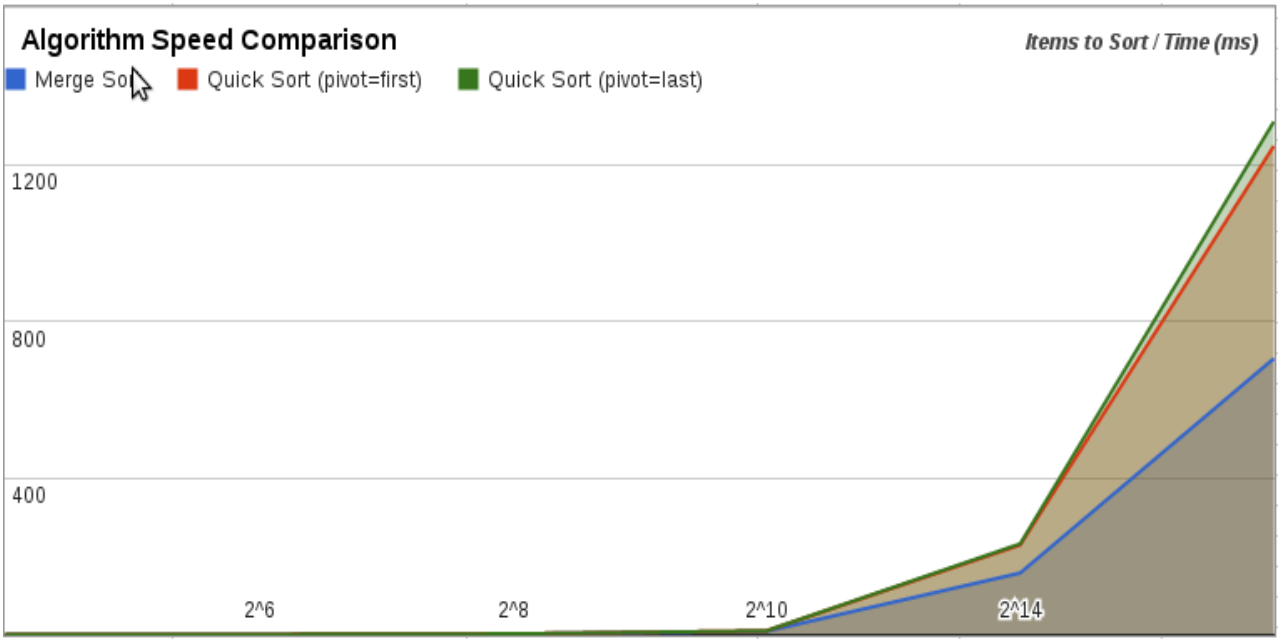
```
# a < b
a <=> b    #=> -1

# a = b
a <=> b    #=>  0

# a > b
a <=> b    #=>  1
```

You could see how this is used for sorting. With this, it is common practice to pass in a lambda, with an arity of two, that uses the same code as seen above such that we could do the following:

```
a = [... random array ...]
a.merge_sort {|a,b| a <=> b}    #=> sorted in ascending order  (eg. 1, 2,
3, ...)
a.merge_sort {|a,b| b <=> a}    #=> sorted in descending order (eg. 10, 9,
8, ...)
```

This is the method used in this Ruby implementation.

The Results



Notice that they are all *roughly* even until we process 2^16 elements (last-data-point) to which the Merge Sort algorithm runs around a half-second faster than both Quick Sort algorithms.

Also, one point of interest is to whether the pivot selection (first vs last) should have any impact on the results. The answer in this study is **no**. The algorithms are so close in their times that one cannot be deemed *conclusively* faster than the other.

So, although the two algorithms have the same asymptotic running times, I would have to go with merge-sort in a *real-world* setting as it is significantly faster on average.

The complete results table (the average of 500 iterations) is:

Items to be Sorted	Merge Sort	Quick Sort (pivot=first)	Quick Sort (pivot=first)
2^4	0.0948	0.0509	0.063
2^6	0.3853	0.3431	0.3412
2^8	1.8689	1.9039	1.8861
2^{10}	8.1148	9.4248	9.4971
2^{14}	156.8467	228.2297	231.3567
2^{16}	704.0989	1246.3766	1308.5267