

Each sorting function is written

from code or pseudocode found

publicly online. The algorithms are

basic and there is no combination

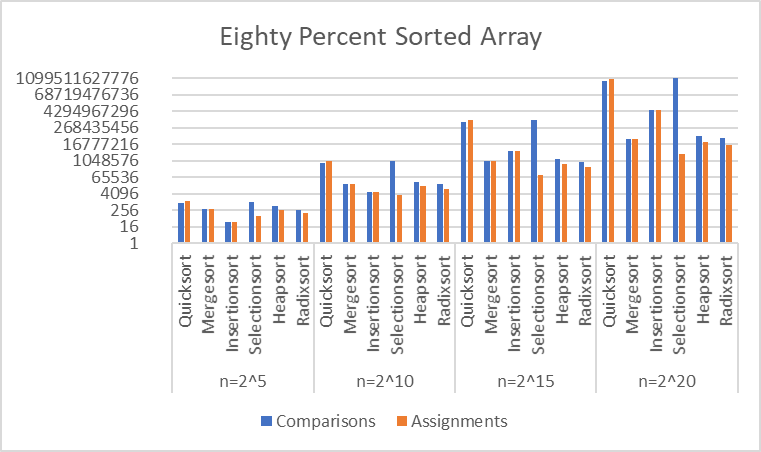
of algorithms. In practice, many

algorithms are made to be hybrid

or adaptive. For example, Timsort

is a Mergesort that uses

Insertionsort instead of merging

for small n.

Quicksort, Mergesort, Insertionsort,

Selectionsort, Heapsort, and

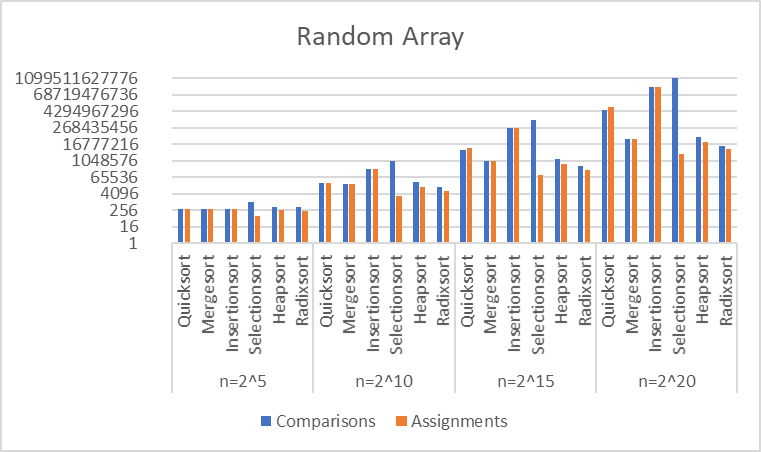
Radixsort are each run on the same

starting array.

There are five array types:

sorted, mostly-sorted, random,

slightly-sorted, and reverse-sorted.



Every combination of algorithm

and array type is tested for

array sizes of every power of 2

between 2^2 and 2^20, inclusive.

The graphs show only four of the

iterations, but are sufficiently

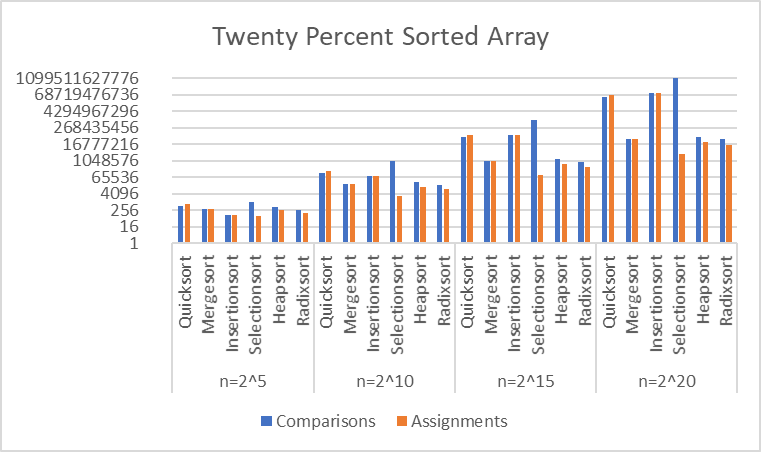
representative of the

ungraphed data.

As expected from the O(N) best case

of Insertionsort, Insertionsort was

the fastest in every sorted array test

and many mostly-sorted array tests.

Selectionsort always used more

comparisons than Insertionsort.

For small n, I expected the O(N^2)

algorithms, Insertionsort and

Selectionsort, to perform better.

However, the O(NlogN) algorithms

were on par with or better than the

O(N^2) algorithms. Selectionsort

took the most time for n=2^5.

Perhaps Selectionsort is slowest

because it used significantly

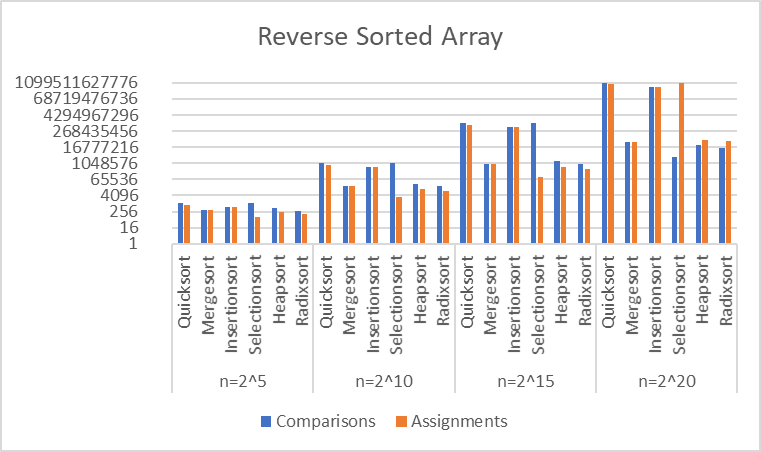
more comparisons than

assignments with respect to

other sorts.

End behavior analysis:

Mergesort and Heapsort performed similarly. Both are O(NlogN), but for n=2^20 Heapsort took ~65% of the time of Mergesort (calculated by the average time between array types).

Radixsort, O(N), performed slightly worse than both Mergesort and Heapsort, only being faster than Mergesort in the random array trial. Despite the seemingly amazing O(N) time complexity, it is no major improvement over O(NlogN) sorts.

Quicksort, Insertionsort, and Selectionsort are all worst-case O(N^2). This is especially reflected in the graphs of reverse-sorted arrays. Quicksort’s average-case O(NlogN) does not appear in the data. It’s best n = 2^20 trial (random array) puts it closer to the NlogN sorts than the N^2 sorts, but it is still significantly slower. Quicksort performed especially bad in the non-randomized arrays because my pivot selection was done by the largest index. This means that in sorted segments of the data, the partitions are of size n-1 and 0. An ideal pivot divides the data into n/2 and n/2, as Mergesort blindly does. A better pivot selection device would have shown greatly improved performance, especially in the sorted and reverse-sorted arrays. Quicksort is the algorithm that I would expect could be improved the most from my current implementation, but I don’t believe that it would ever beat Mergesort or Heapsort.

First place: Heapsort

Second place: Mergesort

Third place: Radixsort

Fourth place: Insertionsort (excellent performance for small n and mostly sorted arrays)

Fifth place: Quicksort

Last place: Selectionsort

