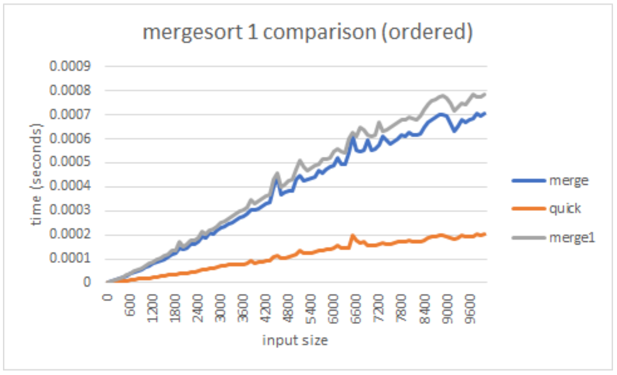
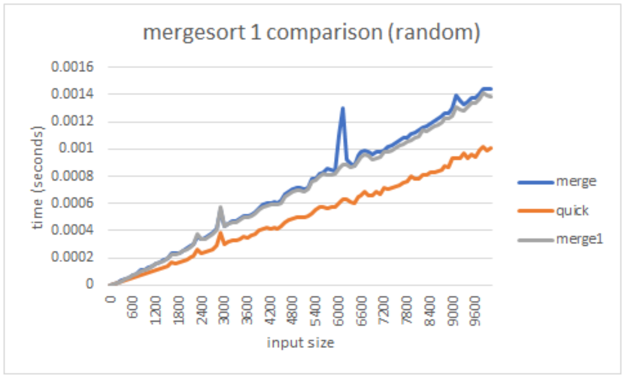
Dylan Moreno

Section MWF 10:30

PA03 Analysis

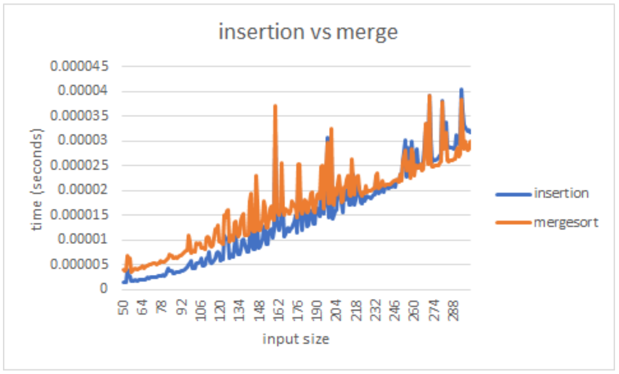
Figure 1: Improved Merges

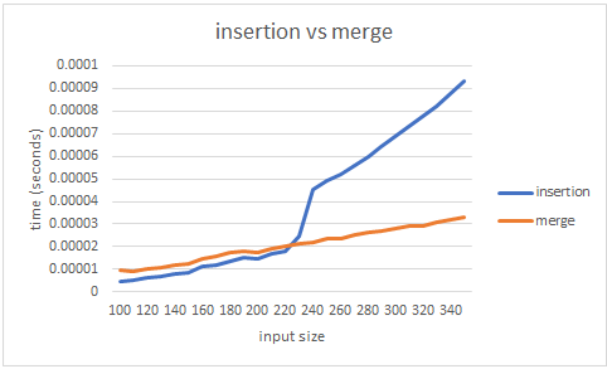




- I gave specifications of a range from 0 to 10,000, an increment size of 100, and each input size was run with 15 tests each. The first figure was run with ordered inputs, and the second figure was run with random inputs.
- The difference between mergesort and my improved mergesort 1 is that the latter uses a half as large auxiliary array. This is primarily for saving space, so the only speed improvement is that there are fewer array assignments (which is a tiny speed improvement as is).
- The general trends for both ordered inputs and random inputs are similar, though all sorts are slower for random inputs as compared to ordered ones. For random inputs, we see the expected result of my mergesort 1 being slightly faster than mergesort, though that is, in general, not the case for ordered inputs.
- The anomalous ‘spikes’ are most likely due to random error.

Figure 2: Threshold Selection

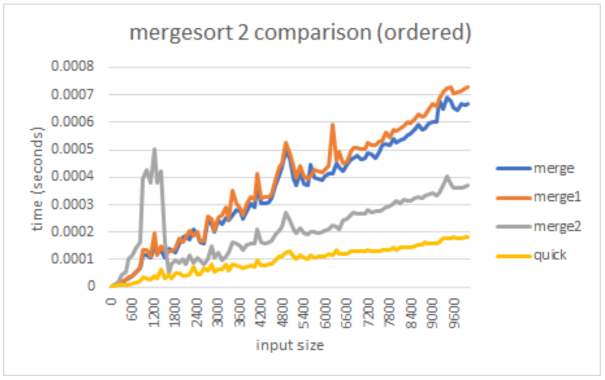


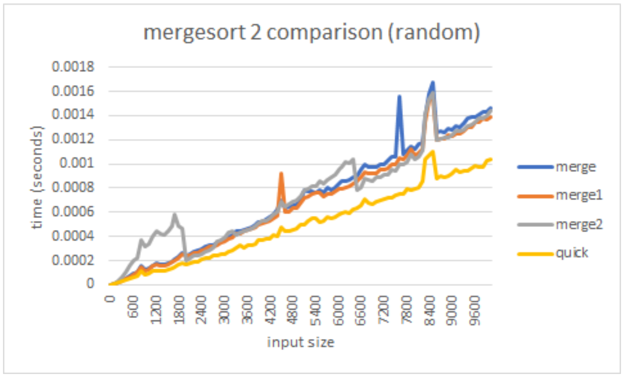


- The threshold is for use in my improved mergesort 2. It is the input size in which insertion sort stops being faster than mergesort. In usage, if a particular region of the array to be sorted is less than the threshold, my mergesort 2 will use insertion sort on that small region rather than continuing with mergesort.

  *****I determined that the best threshold is 254*****
- I ran 100 tests to find my threshold--narrowing down the input range until I was certain that 254 was the best option. Please note that I ran each test with random inputs and 1,000 trials per input size. I began by using input sizes ranging from 0 to 100,000 with an increment of 1,000. Once I examined the data, I narrowed the range down to 0 to 1,000 with an increment of 100. Then I narrowed the range down to 100 to 300 with an increment of 2. I continued this trend until I got to a range of 250 to 258. Once there, I ran 30 more tests and tallied which input size was the first, for that specific test, to have mergesort be faster than insertion sort. The input size with the most tallies, aka 254, is what I chose for my threshold.
- Due to random error and interfering processes, 254 was not the threshold in every test. It was only the most frequent threshold from my experiments. In some cases, the threshold was 253 or 255. In others, the threshold was ~100 or ~1,000.

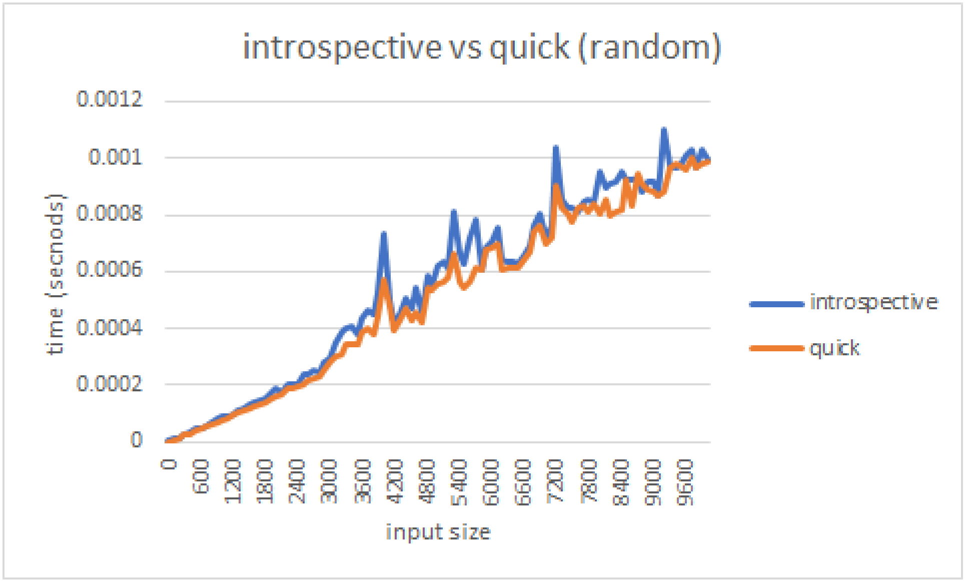
Figure 3: Switching Strategies





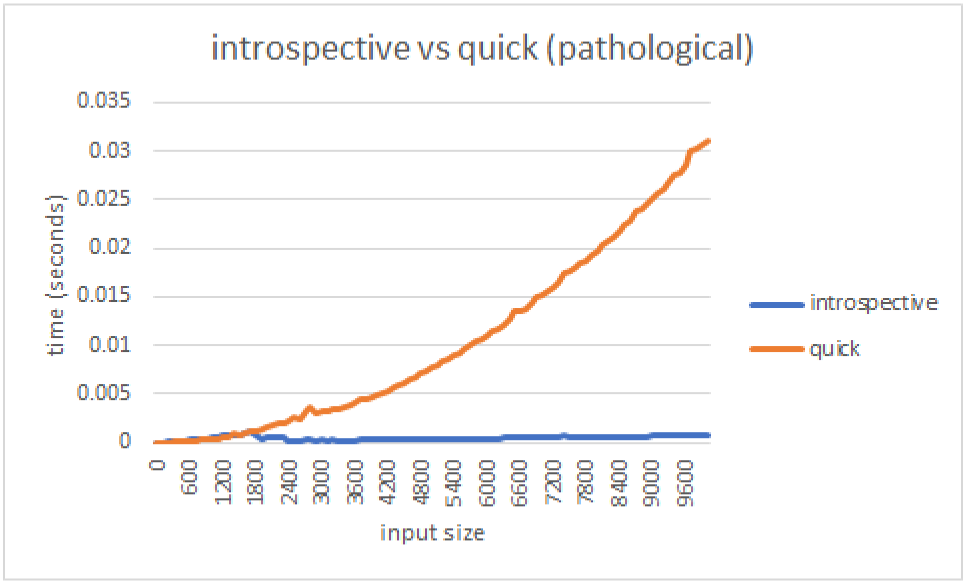
- I gave specifications of a range from 0 to 10,000, an increment size of 100, and each input size was run with 15 tests each.
- The point of my improved mergesort 2 is that it should be overall faster than both mergesort and my improved mergesort 1. This is because my new sorting algorithm switches to insertion sort for smaller regions of the input array. I do this because insertion sort is faster than mergesort 2 up until an input size equal to a threshold I explored in Figure 2.
- The main differences between ordered inputs and random inputs are that ordered inputs, expectedly, resulted in much faster sorts for all four of the algorithms being compared here.
- There were anomalous ‘spikes’ in the data that I would contribute to random chance.
- My improved mergesort 2 was expectedly the fastest sorting algorithm behind quicksort.

Figure 4: Introspective Sort (Random Inputs)



- I gave specifications of a range from 0 to 10,000, an increment size of 100, and each input size was run with 15 tests each. The tests were run with random inputs.
- Introspective sort is a combinational sorting algorithm, which, in this case, combines quicksort, mergesort, and insertion sort. Mergesort and insertion sort are combined by my improved mergesort 2 which I outlined in Figure 3. Quicksort is used primarily, but the algorithm switches strategies to mergesort when a pathological input is detected (see Figure 5).
- As expected, the time difference between introspective sort and quicksort was very minimal, with quicksort being slightly faster. The slight disadvantage to introspective sort is due to the extra check for if quicksort is sorting a pathological input--which, for random inputs, is very unlikely to occur.

Figure 5: Introspective Sort (Pathological Inputs)



- I gave specifications of a range from 0 to 10,000, an increment size of 100, and each input size was run with 15 tests each. The tests were run with pathological inputs. Pathological inputs are inputs which will result in the worst-case scenario for quicksort--when the recursion depth exceeds 2⌊log2n⌋.
- As expected, the time for quicksort is horrendous here, and introspective sort is exponentially faster. This is because introspective sort uses quicksort until the program detects that the inputs are pathological. If pathological inputs are detected, introspective sort stops using quicksort and switches to my improved mergesort 2 instead, which is almost as fast as quicksort, but has much better worse-case performance.