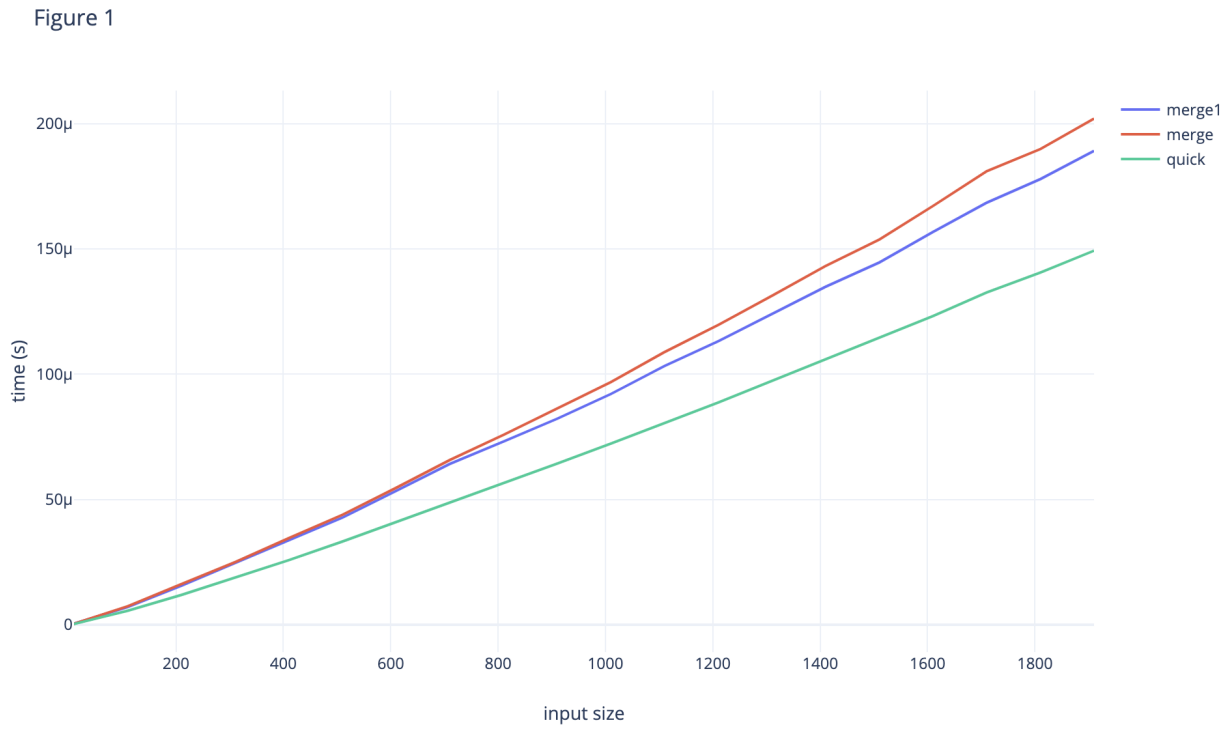
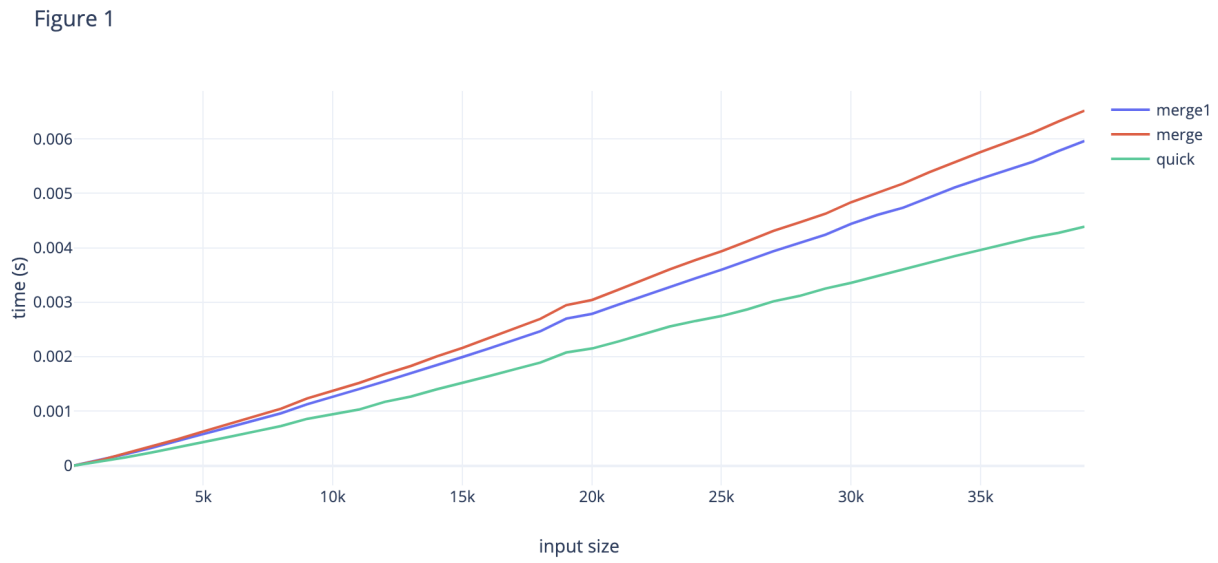
Figure 1

-s 10 -i 100 -m 2000 -t 10000 -w merge1,merge,quick -g random



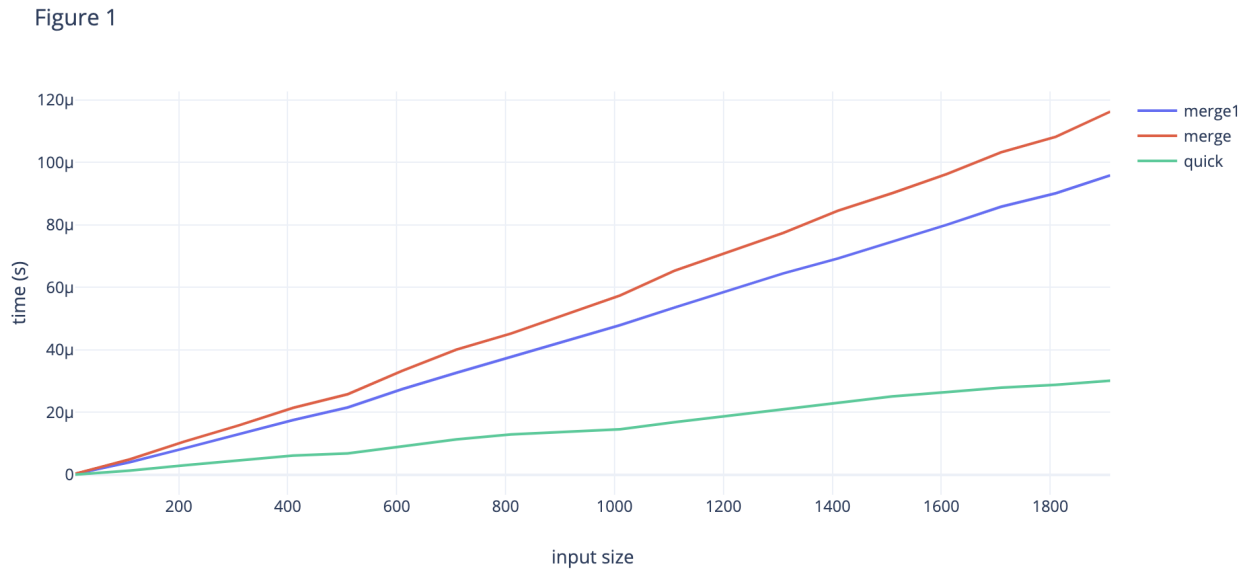
The point of this experiment is to see how the performance of mergeSort1 is compared to original mergeSort and quicksort sorting an array of random numbers of varying sizes. As expected, quick is the fastest. Merge1 is faster noticeably than the original merge even though merge1 implementation was focused on improving space.

-s 10 -i 1000 -m 40000 -t 1000 -w merge1,merge,quick -g random



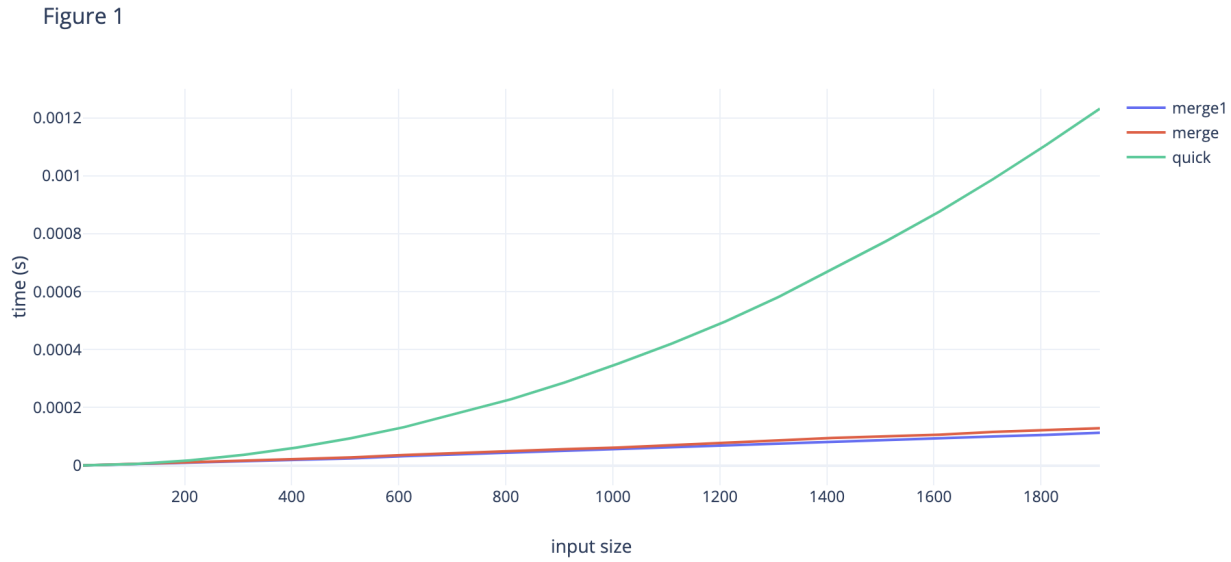
This was just an experiment to see if results would change if I increased the input, there's not much difference between 2k and 40k.

-s 10 -i 100 -m 2000 -t 10000 -w merge1,merge,quick -g ordered



The point of this experiment is to see how the performance of mergeSort1 is compared to original mergeSort and quicksort sorting an array of already ordered numbers of varying sizes. As expected, quick is the fastest. Merge1 is faster noticeably than the original merge even though merge1 implementation was focused on improving space.

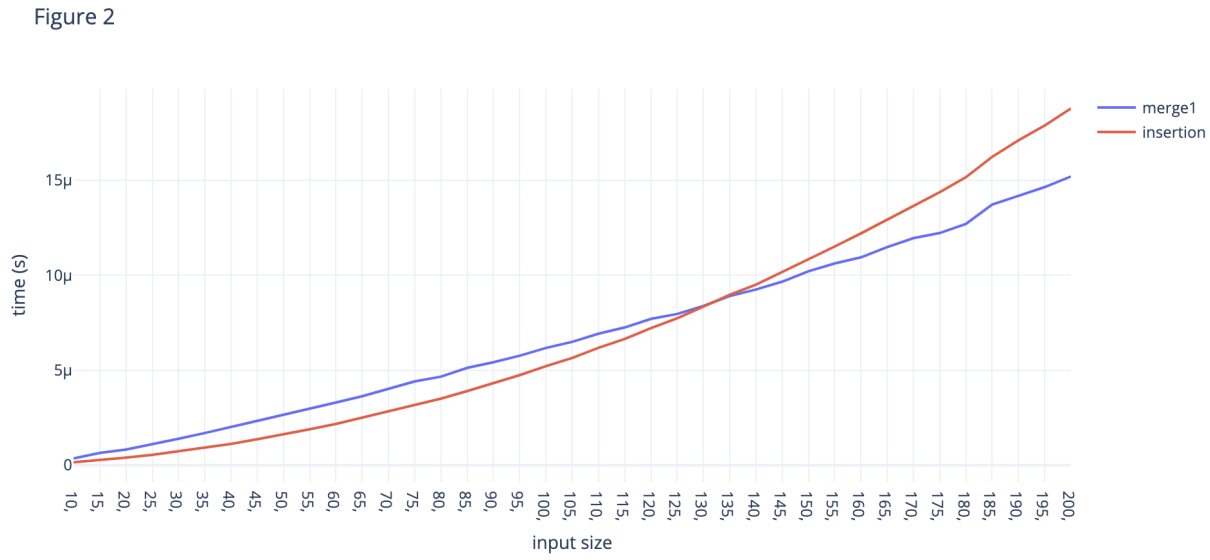
-s 10 -i 100 -m 2000 -t 10000 -w merge1,merge,quick -g evil



The point of this experiment is to see how the performance of mergeSort1 is compared to original mergeSort and quicksort sorting an array of random numbers of varying sizes for the worst case for each. As expected, quickSort follows its worst case time complexity of n^2. Merge1 is faster noticeably than the original merge even though merge1 implementation was focused on improving space. Both merges seem to be following n log base 2 n, which is expected.

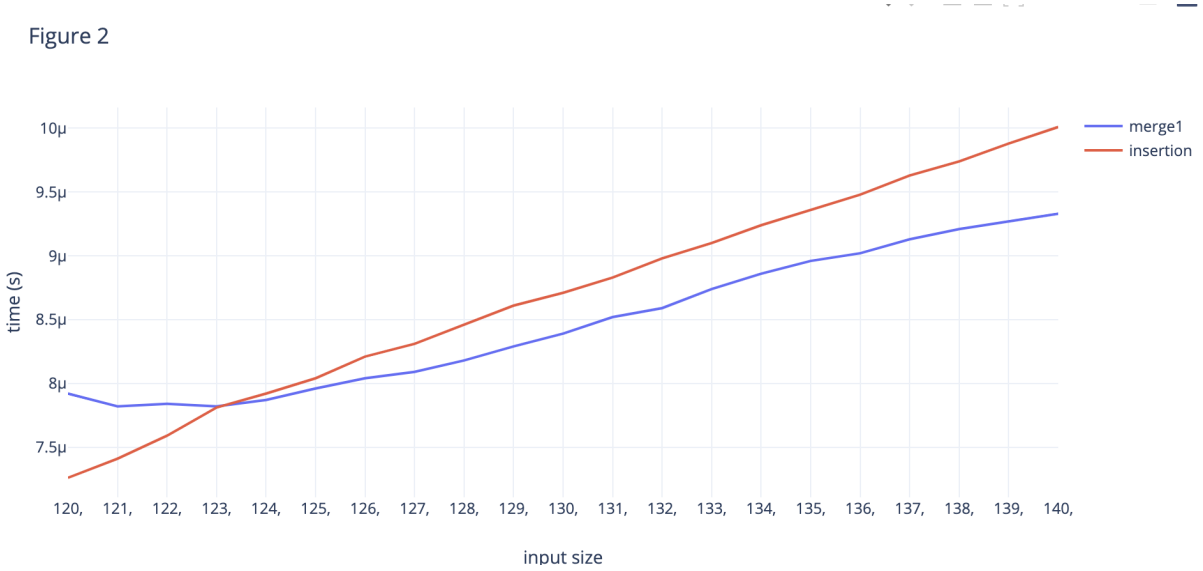
Figure 2

-s 10 -i 5 -m 200 -t 10000 -w merge1,insertion -g random



The point of this experiment is to find the point in which the sorting algorithm should switch from mergeSort to insertionSort because insertionSort is better for smaller array sizes. This narrowed it down to 130-145.

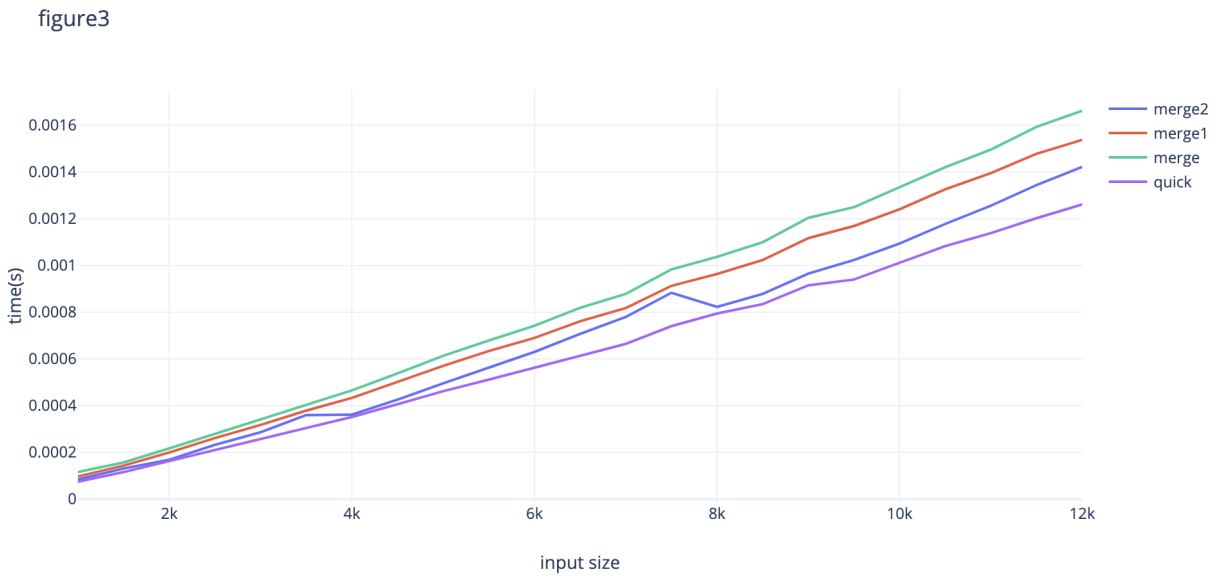
-s 120 -i 1 -m 140 -t 100000 -w merge1,insertion -g random



Continuing Experiment. I ran the same test again but with more trials and changed the interval of size difference down to one. The crossover point is at 123, so I picked 124 as the switching point because every array size smaller than 124 is when insertion sort is as fast or faster than merge sort.

Figure 3

-s 1000 -i 500 -m 12000 -t 1000 -w merge2,merge1,merge,quick -g random

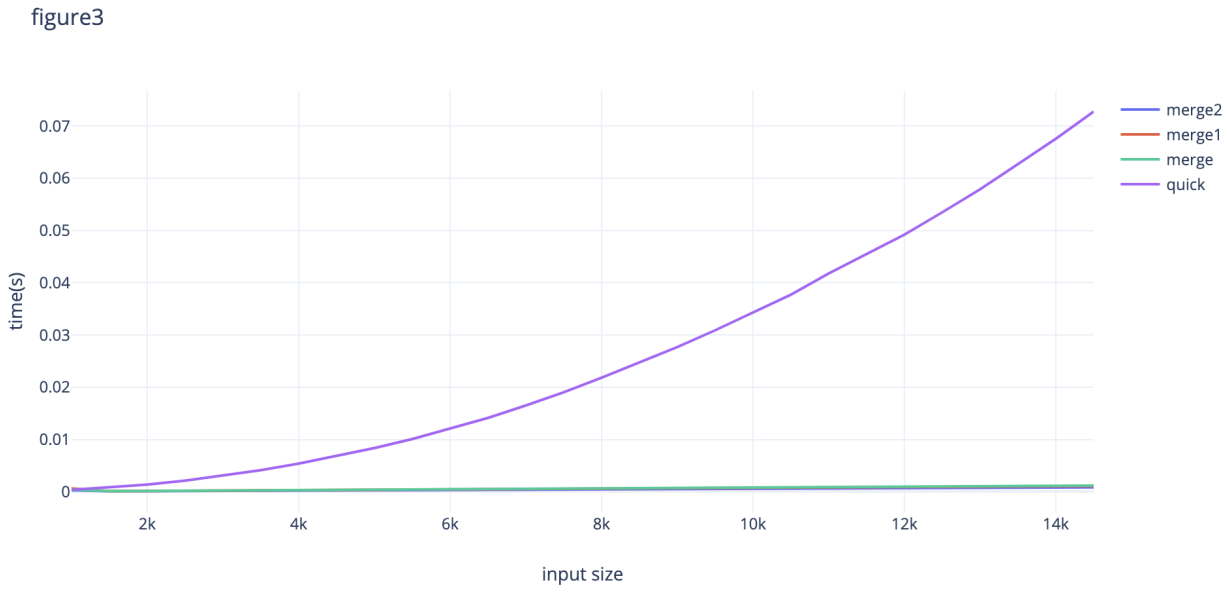


The point of this experiment is to see how the performance of mergeSort2 is compared to original mergeSort, mergeSort1, and quicksort sorting an array of random numbers of varying sizes. Merge2 uses insertion sort to sort when subarrays become less than the size of 124 because insertion sort is faster than merge for sizes smaller than that. As expected, quick is the fastest. Merge2 is faster than Merge1, and Merge1 is faster than merge2.

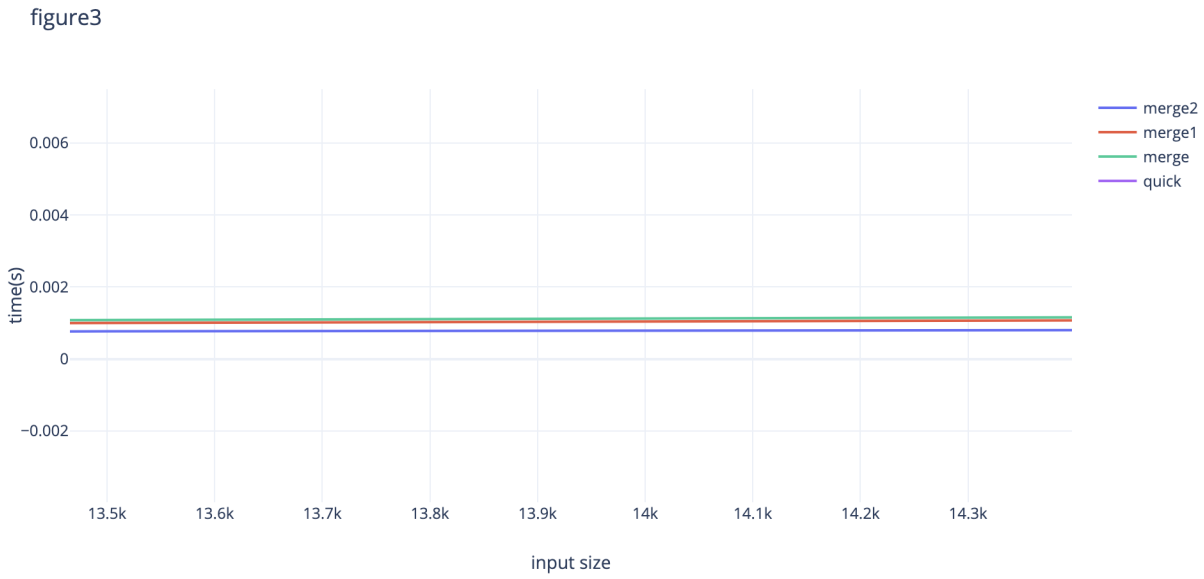
-s 1000 -i 500 -m 20000 -t 1000 -w merge2,merge1,merge,quick -g ordered

The point of this experiment is to see how the performance of mergeSort2 is compared to original mergeSort, mergeSort1, and quicksort sorting an array of ordered numbers of varying sizes. Merge2 uses insertion sort to sort when subarrays become less than the size of 124 because insertion sort is faster than merge for sizes smaller than that. As expected, quick is the fastest. Merge2 is faster than Merge1, and Merge1 is faster than merge2.

-s 1000 -i 500 -m 14500 -t 100 -w merge2,merge1,merge,quick -g evil



The point of this experiment is to see how the performance of mergeSort2 is compared to original mergeSort, mergeSort1, and quicksort sorting an array of varying sizes for the worst case of each. Quick follows the time complexity of n^2 while the mergeSorts seem to follow the worst case of nlogn, as expected.



This is a blew up version of the 3 merge sorts for the evil sort test. The speed order is still merge2 being the fastest, then merge1, then merge. Which is expected.

Figure 4

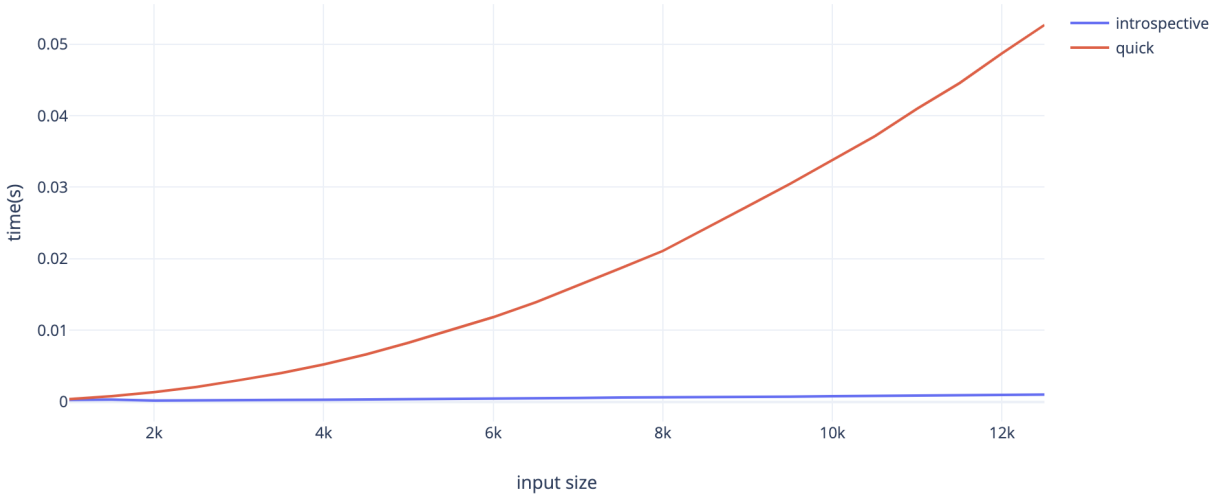
-s 1000 -i 500 -m 20000 -t 10000 -w introspective,quick -g ordered

The purpose of this experiment is to compare introspective sort to quicksort for order inputs. Introspective is almost as fast as quicksort which is expected because introspective is just quickSort except that it falls on a sorting algorithm of nlogn for worst case time complexity. So for ordered inputs it should be similar.

Figure 5

-s 1000 -i 500 -m 20000 -t 100 -w introspective,quick -g evil

figure4



The purpose of this experiment is to compare introspective sort to quicksort for evil inputs. Introspective falls on the sorting algorithm of nlogn for worst case time complexity in this case, so it makes sense that quickSort still follows n^2 while introspective is faster with a time complexity of nlogn.