Max Goldberg, Amir Al-Sheikh

Data Structures

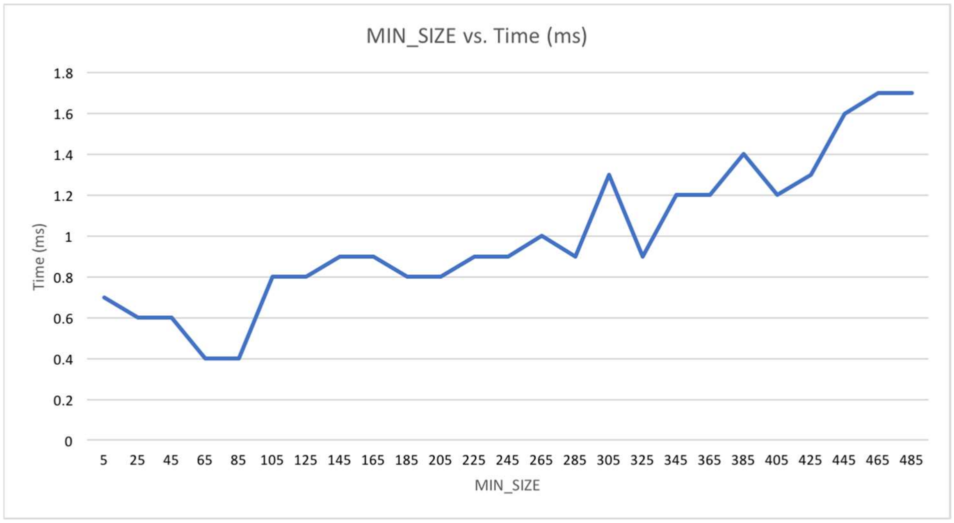
Homework 9

Quicksort

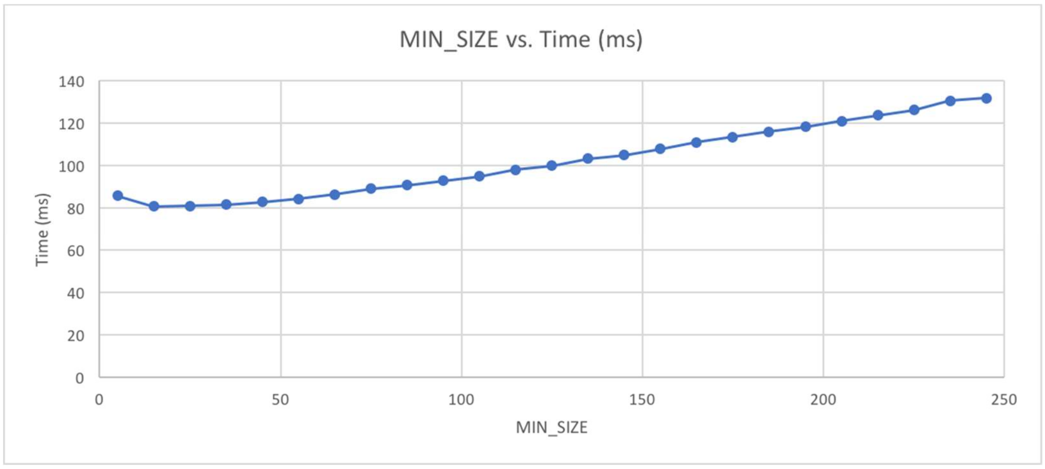
We ran three experiments with the intent to find the ideal subarray size (MIN_SIZE) to switch to insertion sort. Our first experiment went from 5 to 485 in increments of 20 with an array of size 10,000. Our second experiment went from 5 to 250 in increments of 10 with an array of size 1,000,000. Our third experiment went from 5 to 50 in increments of 1 with an array of size 10,000,000. The purpose of increasing the array size as the range decreases is to give a more accurate representation of the importance of small changes in MIN_SIZE. We don't believe there is a single MIN_SIZE where it is most efficient to switch to insertion sort, but rather a range of ideal MIN_SIZEs, depending on what state the array starts in.

We collected our data with a Writer object. Our data was written to a file using the Writer class and then imported into an Excel Spreadsheet. We ran each experiment ten times and aggregated the results. For each experiment, we ended up with several hundred data points, and we created charts representing each MIN_SIZE against time.

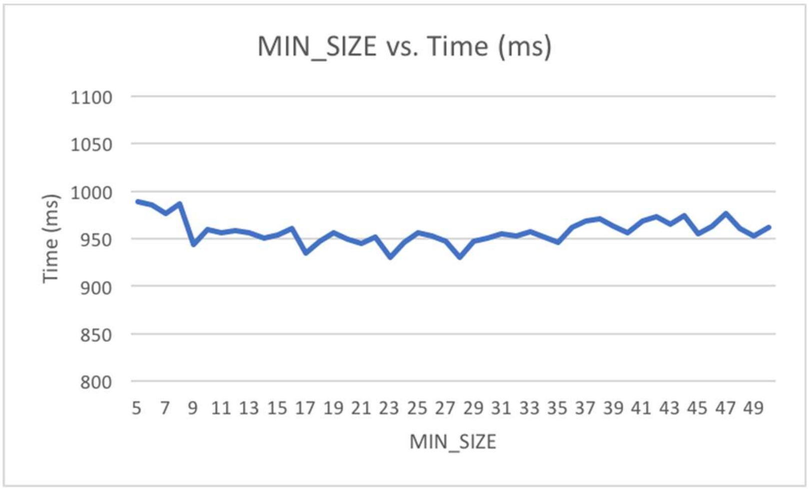
In our first experiment, we started with a broad investigation to try and find a range of ideal MIN_SIZEs. The data we collected ranged from a MIN_SIZE of 5 to 485. Once we noticed we were getting results in the one to two-millisecond range, we thought our results would be inconclusive. But, as we compiled our data, there seemed to be a general upward trend. This led us to believe that the best MIN_SIZE is under 250.



In our second experiment, we used the data from our first experiment to shrink the range of MIN_SIZEs we tested to between 5 and 250, and we increased our array size to 1,000,000 to increase the amount of time each sort takes, thus making results more distinct. These results show a much clearer trend than the previous experiment, while also giving us a more specific range to test.



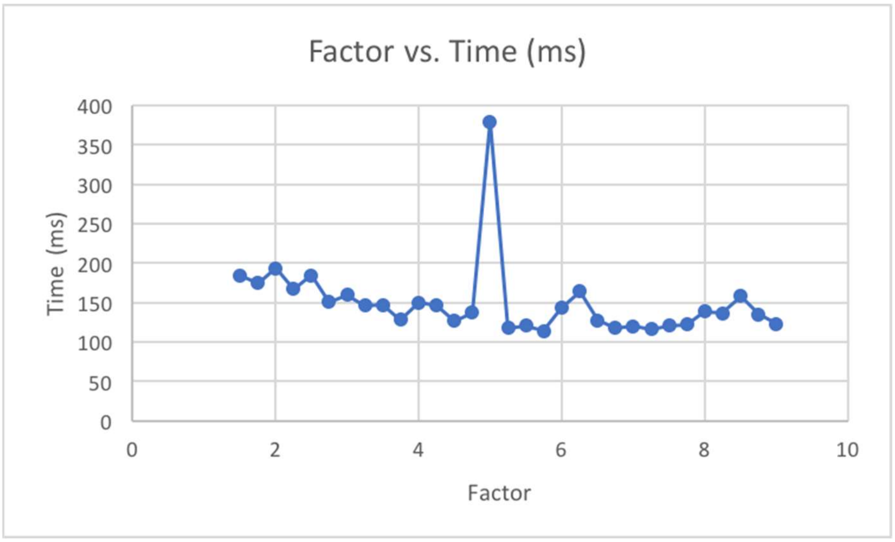
We created a third experiment, limiting MIN_SIZE even more and increasing the size of the array up to 10,000,000. This gave us a chart which does not show a clear best MIN_SIZE but tells us that a subarray size of 20-30 is a good time to switch to insertion sort.



Throughout all of our charts, there is a trend to have lower values with a MIN_SIZE below 50. While the third chart is slightly unsatisfying, it supports our hypothesis that there exists an ideal range of MIN_SIZEs rather than an ideal single value.

Shellsort

In our experiment, we tested how the efficiency of the shellsort method would change as we changed the factor used to modify the gap. The size of our array was 1,000,000. Given the fact that 2 and 2.25 are relevant factors, we ran our first test around them, going from 1.5 to 9 by increments of 0.25. Going in, we expected a factor of 2.25 to be very efficient. The data we collected did not align with our hypothesis. The minimum value was a factor of 5.75. We were also surprised at how inefficient a factor of 5 was. The chart below presents this data. After this experiment, there was no specific best region to zoom in on, as we did with quicksort.



When comparing both quicksort's and shellsort's most efficient implementation with an array size of 1,000,000, we noticed that quicksort was more efficient. Quicksort ran at 80.7 ms, while shellsort ran at 114.4 ms. The MIN_SIZE we found to be most efficient for quicksort was around the range of 10 to 20. It is important to note that this comparison is specifically for very random datasets, and each sorting method is better in different situations. This phenomenon also explains why different factors are better when the dataset starts in different states.