**Hypothesis:**

Based on the last experiment, Merge Sort becomes faster than Insertion Sort at an N size of 1,100. The hypothesis is that a Hybrid-Merge Sort algorithm that utilizes Insertion Sort when the data size is at the cross-over between the two (1,100) will be the most efficient version of that algorithm.

**Method:**

To determine this, we can use the same set up from the last experiment, including environment and compiler flags. However, instead of the input size being the independent variable, we will hold input size to a constant and instead test the value K, which will determine at what data size in the recursive function we will use Insertion Sort. However, we will still test at different N levels to see what correlations arise, but it is not the independent variable.

An extra parameter is added to the mergeSort function, which is K. Within the function, we detect if the size of the vector being examined (right – left) is larger than or equal to K. If so, we allocate a vector which is that sub-vector, and then call insertSort on it. The newly sorted vector is then inserted back into the main vector. All the code can be accessed at the repository: <https://github.com/ephraimbennett/SortingExperiment>. The additional logic for the hybridization is also below:

A screen shot of a computer program

Description automatically generated

Once the code was working, tests were run at K values of 16, 256, 512, 1100, 1600, and 2400. These were chosen to understand a broad range of values, and specifically 1100 as the “crossover” for when Insertion becomes slower than Merge. All K values were tested with N=250,000 and N=125,000. Each K value was tested 5 times for its N level, and the average of these results was taken as our data points to graph.

The next step is to compare against the performance of pure Merge Sort and Insertion Sort, but the data for this was collected in the last experiment, so no new tests need to be conducted.

**Results:**

The results of the tests can be seen below, with one chart for each respective N value.

The fastest time for the Hybrid Merge Sort was at a K value of 256, after which the time continued to grow at a seemingly exponential rate, although not enough data is conclusive for that. At a K value of 256, the average speed of the program was 0.0148 seconds for N=250,000 and 0.0076 seconds for N=125,000.

The performance of Hybrid Merge can also be compared to regular Merge, at N=250,000

All these, however, are significantly faster than Insertion Sort at N=250,000. That time averages out to 4.406 seconds, which is slower by a factor of over 1000, and would not be represented well on a linear scale.

**Discussion:**

The results of this experiment indicate that the hypothesis is incorrect. The average Hybrid Merge that sets K to the “cross-over” point is not the fastest. A K value of around 256 is nearly twice as fast. There are many possible explanations, but the one which seems most clear has to do with the magnitude of speed difference between Merge and Insertion. Although Insertion will be faster for N <= 1100, that is in comparison to a “pure” Merge, where the array is split all the way until it is only one or two values. However, splitting to a smaller size but not all the way down to 1 (ie. 256) and then using insertion is the most efficient. This balances the performance of the linear aspect of merging, while taking advantage of Insertion’s fast time for small values. One way it could be looked at is that even though Insertion will be faster at N=1100, that is not when it best outperforms Merge, only the threshold where it becomes desirable. The greatest speed differential will provide the best results when swapping with Merge to fully optimize. Therefore, we should take the fastest version of Insertion to combine with our Merge algorithm.

**Conclusion:** Under the conditions tested, Hybrid Merge is the most efficient with a K value of 256 in comparison to K values of 16, 512, 1100, 1600, and 2400. Whether or not 256 is the most efficient K possible is not answered here, but it should be near the most efficient K value for this particular computer system.