a. Hypothesis

From the problem description and my experience with hash tables I would expect that the Unordered multiset would be faster with insertions. I know that hashing values to insert is usually much faster than using sorting methods to insert values so in this case I would expect the unordered multiset to be faster. I expect that any number of insertions under 1000 will probably take less than a tenth of a second and not show significant results for either method because of the speed of processing speed of modern-day computers. But I do estimate that at some point over 10,000 and below 100,000 insertions that there will be a significant difference in insertion times and that the multiset will take over 3 seconds to run.

b. Methods

I decided to use C++ to test hash tables and balance binary trees because of the already implemented multiset and unordered multiset in the standard library. I figure by using these I would avoid any possible error from creating my own or using another language and that I would get the best estimate of time from them. I used Visual Studio which uses the Microsoft C++ Compiler. I originally was using the default optimization flag but then I switched to /O2 maximum optimization with default basic runtime checks. I changed these settings in the property page of the solution explorer in visual studio. My code repeatedly initializes N number of random numbers and times how long it takes to insert all the random values into a multiset and unordered multiset repeatedly incrementing N until one of the insertions takes more than 3.0 seconds. I originally just multiplied n by 10 each repetition but realized that significant changes in time only start to show around 100,000 and an exponential increase of n does not give me a lot of data to analyze. I changed my code to just increment N by 100,000 each repetition after N first reached 100,000 until either reached > 3.0 seconds.

c. Results

Shown below (Figure 1) are the results from my experiment using default optimization. The x values on the graph are the N values I used ranging from 10 to 900,000. The y values represent the time the insertion took for the corresponding N value for a hash table and balanced binary tree. The results are from running my code in Visual Studio using default optimization. The hash table time are in blue and balanced binary tree times are in orange.

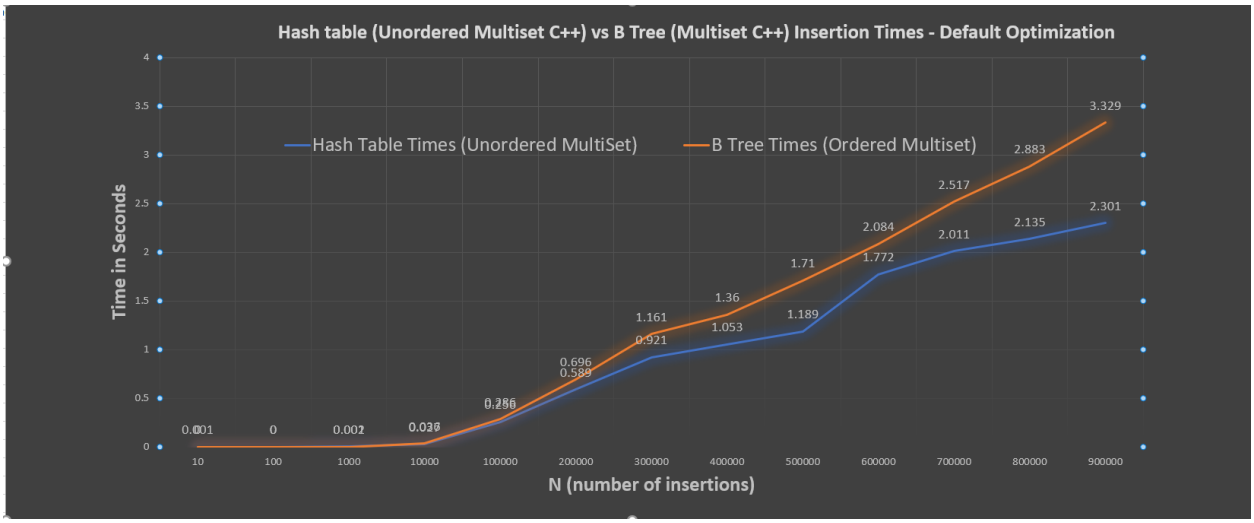

Figure 1

Shown below (figure 2) are the results from my experiment using Visual Studio and O2 Maximum optimization (favoring speed) and default runtime checks. Because of the increase in speed, the x values range in values from 10 to 1,600,000. The corresponding y values represent the time of the insertions.

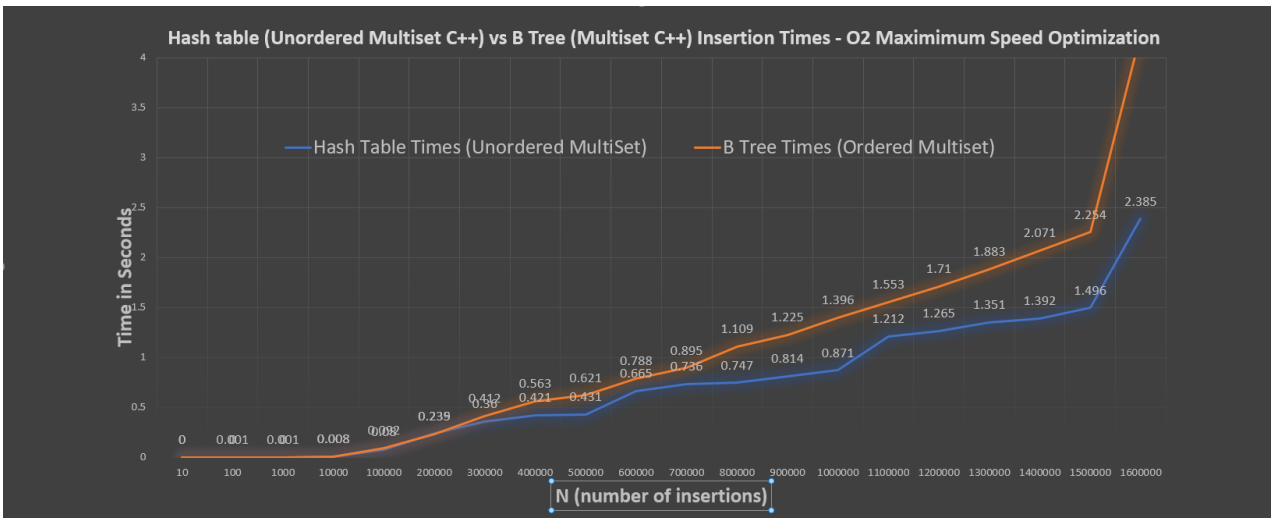

Figure 2

d. Discussion

The overall expectation of the experiment was confirmed by in the results because the hash table was faster at insertions. Major differences in time were not significant until I had tested over 100,000 insertions which was larger than I expected. One thing that surprised me was how big of an effect using the O2 optimization flag had. Using the same code but with just O2 optimization almost doubled the number of insertions both data structures could do within 3 seconds. I was surprised by

the major jump from 2.234 seconds to 4.2 seconds from 1.5 million to 1.6 million insertions, but I expect it is likely just are rare case and an average would increment more evenly. I expected a larger difference in time though considering that std::multiset insertion takes logarithmic time and std::unordered_multiset is constant time on average.

e. Conclusions

From the data shown in my graph and my own testing I can conclude that for N < 100,000 insertions the difference in time is minimal and indistinguishable between the data structures but for an N > 100,000, a hash table is faster than a balanced binary tree using std::multiset and std::unordered_multiset in C++.