Jeremy DePoyster

Professor Nicholas Penney

University of Florida

COP3503 Summer 2020 Online

**Lab 3 Extra Credit**

**Setup:**

For this project, I implemented a print function that output the type, scale, N, Push time, Pop time, and Resize counts for each operation. This allowed easy copy-and-paste to a .csv file for data analysis and simple chart-rendering.

In ‘main.cpp’ I built a Test method using the following structure:

Test( ABQ\_or\_ABS, N, scale\_factor )

This way, I was able to loop through arrays of the test N’s and scale factors easily in the test file. The test method would then push/enqueue and pop/dequeue the appropriate number of N’s at the correct scale factor. I implemented a print method that output the type, scale, N, Push time, Pop time, and Resize counts for each operation, with each data point delimited by a comma. This allowed easy copy-and-paste to a .csv file for data analysis and simple chart-rendering.

Initially, the program crashed due to the floating point comparisons of ratios in the pop / dequeue segments, but thanks to fellow student Akash Patel in the class Slack channel, I was able to create an “if” statement that would round floats within a 0.00001 difference. Similarly, my initial Lab solution was re-creating the Stack or Queue on each pop / dequeue, but I rewrote this to only re-create when a resize was in order. This was achieved using an int “counter” variable that would increment when dequeuing, using this as the “front” of the queue, rather than re-copying the entire queue each time. This allowed me to run the test with N’s in the millions as required, rather than thousands, also accomplished by utilizing x64 rather than x86 debugging in Visual Studio.

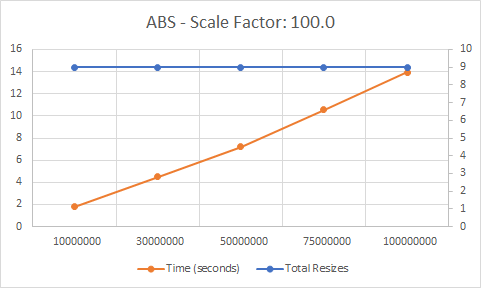
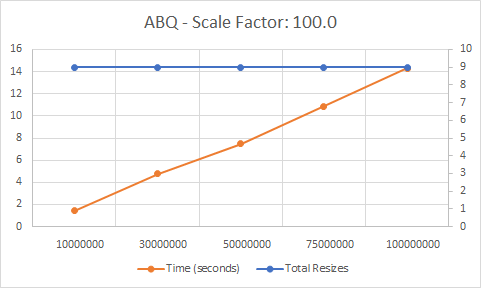
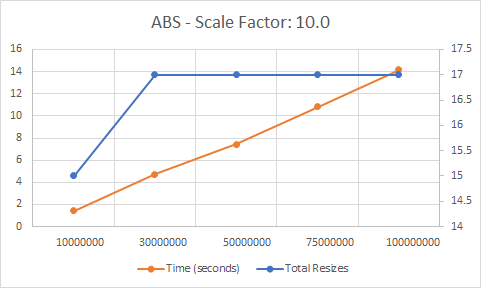
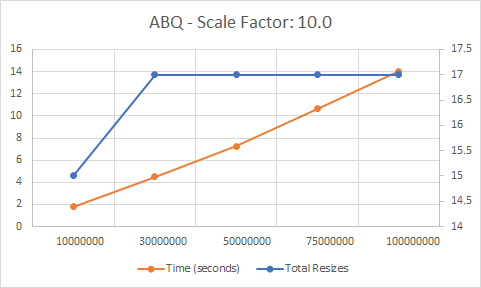
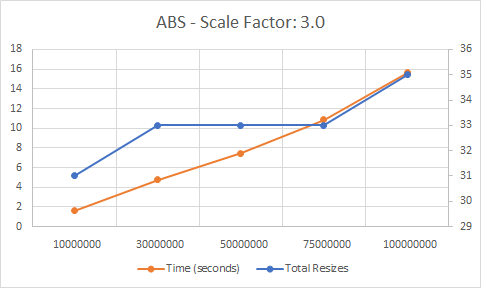
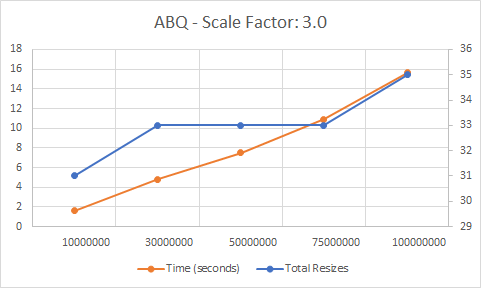
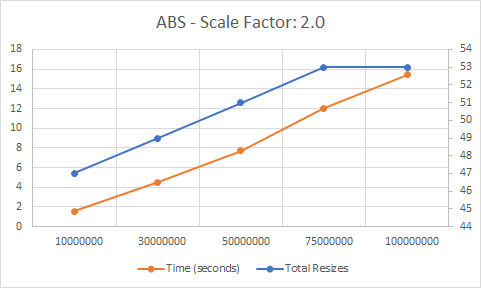
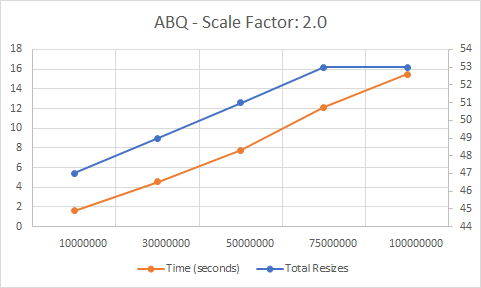
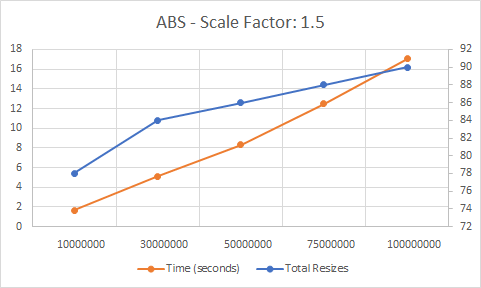
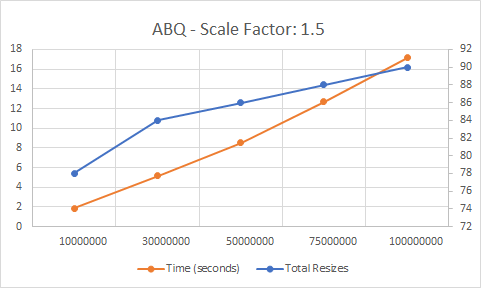
**Findings:**

1. Based on the charts, the higher the N, the longer the time, but it seems to be on an almost direct trajectory, with a similar slope regardless of which structure was used. I think that because of the timing lines from the chart, both the ABQ and ABS are likely O(n) in Big-O complexity because of their consistent slope regardless of N input.
2. Changing the scale factor had a significant impact on how many resizes were necessary, ranging from 90 resizes at a 1.5 scale factor with 100 million N’s, to just 9 resizes at the 100 scale factor with 100 million N’s.

Interestingly, changing scale factor did not have a large impact on the time necessary to push and pop large numbers of integers. At scale factor 1.5 the time required varied from nearly 2 seconds to 17 seconds in both models, while a scale factor of 100 only offered less than half of a second to a second in savings on both models. This indicates that N is a much bigger factor than scale factor in time required to render.

1. At the lower scale factors of 1.5 and 2, were pretty evenly trending upwards, but once reaching scale factors of 10 and 100, the number of resizes were not only largely reduced (from 50-90 down to just 9 resizes at 100), but the higher scale factors saw no difference in the number of resizes between N sizes. I think that a significantly larger N size than tested would have been necessary to see a higher scale factor change resizes.
2. The best scale factor in both models seems to be 100. This is because it requires less resizes overall, as well as nearly 2 seconds of savings on the higher end. This savings should be even more significant as more N’s are introduced, as long as the memory is not necessary for other operations. In that case, 10 could be the best choice?
3. Unless I did something incorrectly, both my ABS and ABQ had very similar characteristics, even nearly identical, in this testing. Perhaps this indicates that the processing required for popping off the top or dequeuing from the front is similar, and other factors like the scale factor have a bigger impact on performance.

Charts for reference are accompanied on the following pages.



A picture containing building

Description automatically generated