Vince Velocci – Lab 3 Extra Credit Assignment

Analysis of the ABS:

When varying the scale factor for increasing the maximum capacity, one can see that as the scale factor increases from 1.5 to 100, the total number of resizes approaches a step function. For all scale factors, as you increase N, you also increase the total number of resizes. This is clearly expected: If you add many items to the ABS, the ABS will fill up more often, requiring more resizes. The same thing applies when removing items. The total number of resizes increases rapidly as you increase N but then the rate of increase decreases as N gets very large. This makes sense because if N is large, then the first resizes of the ABS will have (exponentially) expanded the size of the ABS so much that future resizes are unnecessary. For each scale factor, the total number of resizes reaches a constant value once N is large enough. For both the ABS and the ABQ, the number of resizes done during the pop/dequeue task was roughly twice the number of resizes done during the push/enqueue task.

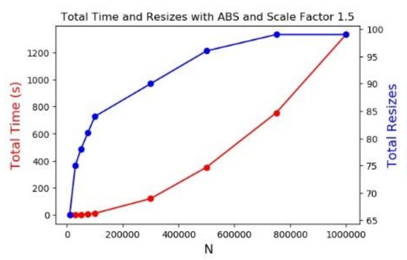
One can see that for a scale factor of 10, the total number of resizes reaches a constant value of 19 rather quickly. For a scale factor of 100, the number of resizes reaches a constant value of 10 almost as soon as we begin increasing N. This makes sense because having a large scale factor allows you to increase the size of your ABS so much that the ABS needs to resize itself less often. We can see this in the graphs. As scale factor increases, the constant value of total resizes becomes less and less. For a scale factor of 1.5, the total number of resizes approaches a value that's close to 100 as N increases. For a scale factor of 10, that value becomes only 19. For a scale factor of 100, the total number of resizes reaches a constant value of only 10. If you multiply your maximum capacity by 100, the size of your ABS becomes large very rapidly, making future requirements for resizes occur less often.

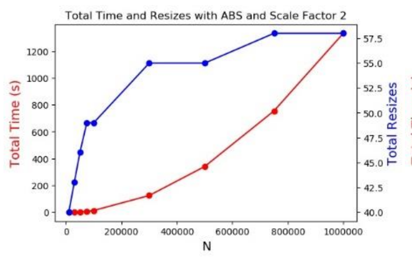
In contrast, one can see that the total time taken to push the items onto the stack and pop them off behaves the same as we raise N no matter what the scale factor is. The values are also about the same no matter what the scale factor is. The total time taken follows a quadratic upward trend with N (in contrast to the total number of resizes which jumps to a value and stays at the value). Adding a trendline to the plots in Excel (though the plots themselves were made with the matplotlib Python library), I found that a third order polynomial in the variable N provides a good fit to the data.

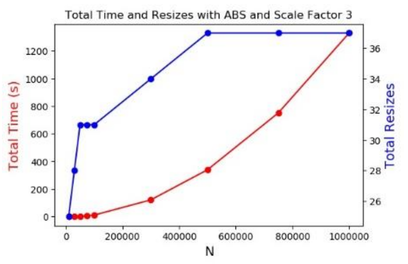
Since the total time taken is independent of the scale factor, it seems like the optimal value for scale factor would be 100 since it results in the least total number of resizes. Having a smaller number of resizes is beneficial since this results in a lower number of times the program needs to copy values each time it resizes the stack.

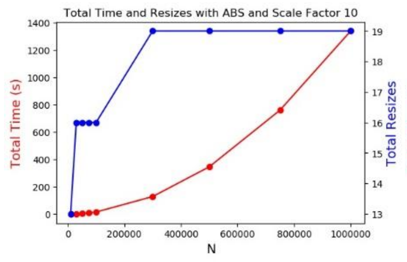
Analysis of the ABQ:

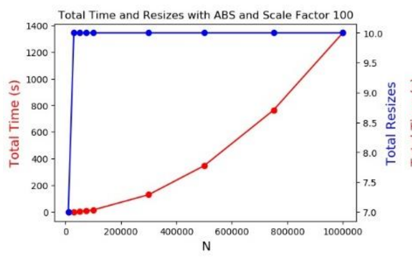
For the ABQ, the total number of resizes is the same as for the ABS for the same values of scale factor and N. This is not surprising as both the ABS and ABQ follow the same rule for determining when to resize the linear data structure. The only difference between the two is the position of the data element that gets removed when either pop() or dequeue() is called. Resizing only occurs when the size crosses a threshold, a threshold which is the same for both ABS and ABQ. Since the procedure for resizing is virtually identical to both (i.e. same amount of memory is allocated and the same number of elements are copied over) and since it should take the same amount of time to push and enqueue elements, neither is it surprising to see that the total time taken for the ABQ is more or less identical to the time taken by the ABS. Differences may be attributed to fluctuations in the speed of the computer at the time the program was run. Therefore, the same conclusions apply to the ABQ as written above for the ABS. One caveat: When analyzing the total time for the ABQ, I did notice that as the scale factor increased, the number of total time measurements that were less than the respective measurement for the ABS decreased. For example, for scale factor 1.5, only one value of N results in a total time for the ABQ being greater than that of the ABS for the same N and scale factor. For a scale factor of 2, there are two values of N that result in a total time for the ABQ being greater than that of the ABS. For a scale factor of 10, that number jumps to 4. For a scale factor of 100, that number jumps to 7. However, the time differences between the ABS and the ABQ are very small, so I don't know if there is any significance to the observation I just mentioned. Below are the graphs generated (using the matplotlib Python library) from the analysis:

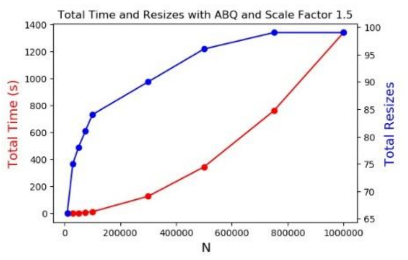

Total Time and Resizes with ABS and Scale Factor 1.5
Total Time (s)
Total Resizes
N

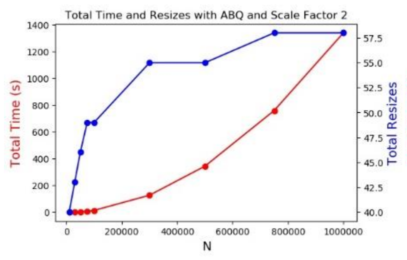

Total Time and Resizes with ABS and Scale Factor 2
Total Time (s)
Total Resizes
N

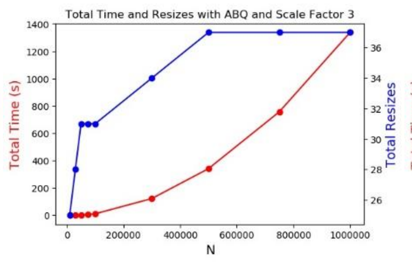

Total Time and Resizes with ABS and Scale Factor 3
Total Time (s)
Total Resizes
N

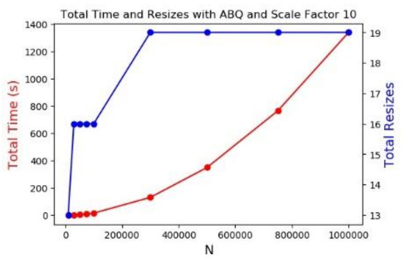

Total Time and Resizes with ABS and Scale Factor 10
Total Time (s)
Total Resizes
N

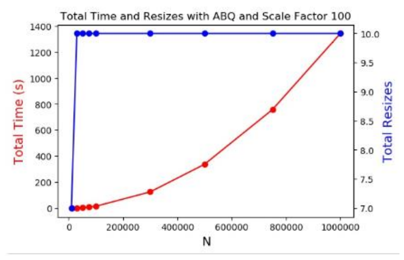

Total Time and Resizes with ABS and Scale Factor 100
Total Time (s)
Total Resizes
N


Total Time and Resizes with ABQ and Scale Factor 1.5
Total Time (s)
Total Resizes
N


Total Time and Resizes with ABQ and Scale Factor 2
Total Time (s)
Total Resizes
N


Total Time and Resizes with ABQ and Scale Factor 3
Total Time (s)
Total Resizes
N


Total Time and Resizes with ABQ and Scale Factor 10
Total Time (s)
Total Resizes
N


Total Time and Resizes with ABQ and Scale Factor 100
Total Time (s)
Total Resizes
N