Chapter 1.23 - #4

Algorithm d contains two while loops and a few statements that need to be analyzed. There are two statements outside the while loops that will take constant time. The first while loop goes from 0 to n-1, but the iterator, “i”, is iterated by 2. This causes the first while loop to take n/2 + 1 time. The second while loop iterates from 0 to n/2. This will also have a time of n/2 + 1. The statements inside the two while loops are executed only when the test passes which is n/2 times. When the execution steps are added together, they sum to 3n + 4. This gives a big-ohh execution time of O(n).

The count variable was removed from the code for timed testing.

In my testing, I ran values from 100 to 4900 in increments of 200. With an O(n), I would expect the time plot to be linear. Each value of “n” was run 5 times and the timed average was used in the plot. The first half of the testing was linear, but a steep drop off in time occurred in the later values of the testing. I first thought this was due to changes in other programs running on my computer, but even after closing applications the drop off stayed present in the results.



Chapter 1.24 - #5

This function performs the transpose of a square matrix. The algorithm can be visualized as iterated over the bottom half of a square matrix and swapping the values with the symmetric position. For example, the value in (1,2) is switched with the value in (2,1). There are two for loops that will accomplish this task. The first for loop iterates over the n-rows in the matrix. The second will iterate from 0 to i. Since i goes to n, the summation is from 1 to n. This yields the inner for loop with (n)(n + 1)/2 steps. There are 3 steps plus the for-loop check. The inner for loop with total at 4(n)(n + 1) / 2 steps. The extra check for the inner for loop will be checked n times and one more step is added for the final check of the outer for loop. The final step count is (4n^2 + 8n + 2) / 2 which yields big-ohh of O(n^2).

The count variable was removed from the code for timed testing.

In my testing, I ran values from 10 to 200 in increments of 10. With an O(n^2), I would expect the time plot to be quadratic. Each value of “n” was run 5 times and the timed average was used in the plot. The data plotted approximately quadratic but was separated into three groups.



Chapter 1.25

This function takes two matrices, a and b, multiplies adjacent cells from matrix a and matrix b. This is done n times before moving onto the next index. The algorithm is one for loop for the rows, another for loop for the columns, and then a third for loop is added to perform the multiplication n times. This yields a final big-ohh of O(n^3).

The count variable was removed from the code for timed testing.

In my testing, I ran values from 10 to 200 in increments of 10. With an O(n^3), I would expect the time plot to be cubic. The data turned out to fit this assumption well. The cubic function varies a little in the beginning while the n-values are low because of variation in other programs running on the system. Once the program runs for longer n-values, the average is more accurate.



Chapter 1.26

This function also performs a multiplication of two matrices. More variables are passed into the function to define the matrices so that total cost function is based on all three of these variables. The inner for loop will run mn(n+1). The middle for loop will run m(p+1). The outer most for loop will run m times. The final check that for loops perform gives them the +1. The big-ohh analysis of this function will then be O(mpn).

The count variable was removed from the code for timed testing.

In my testing, I ran values from 10 to 200 in increments of 10. With an O(mnp), I would expect the time plot to be close to cubic for square equal square matrices. The data turned out to fit this assumption well.

