Robert Myers

CMPSC122

Assignment 11

1 November 2021

Assignment 11

1. The BLUE algorithm has a time-complexity of . The outer “i” loop runs . The inner loop runs times with respect to i. This is calculated by the following: When i=0, j runs 5 times. When i=1, j runs 4 times, when i=2, j runs 3 times and so on. This loop evaluates (n)+(n-1)+(n-2)+…+3+2+1. This inner loop is the leading contributor to the time complexity (upper bound) --
2. The GREEN algorithm has a time complexity of . This is calculated by the outer loop having a time-complexity of O(n). The inner “j” loop is executed when i=0, n times, when i=1, (n-1) times, when i=2, (n-2) times. The j loop evaluates n+(n-1)+(n-2)+(n-3)+…3+2+1 times. This is the same as 1+2+3+…+(n-2)+(n-1)+n which evaluates to or in O-notation O(n2). For the inner-most “k” loop, when i=1, this loop runs 1 time, when i=2, the loops runs 4 times, when i=3, the loop runs 10 times, when i=4 the loop runs 20 times, when i=5 the loop runs 35 times. 1,4,10,20,35 which is a tetrahedral number . We can ignore the “I” and “j” loops because they represent a lower bound. The leading contributing factor for this algorithm would be the “k” loop which has a time complexity of
3. The RED algorithm has a time complexity of . This is due to a singular loop that runs *n* times. This if/else if statement inside this loop only contributes a smaller order to the time complexity therefore it can be ignored to calculate the overall time complexity.
4. Computer Specifications
   1. OS – MacOS Monterey v12.0.1
   2. Processor - Apple M1 Pro
   3. RAM - 16 GB
5. BLUE – Run Times Text

   Description automatically generated
6. GREEN – Run TimesText

   Description automatically generated
7. RED – Run TimesText

   Description automatically generated
8. BLUE
   1. In my initial prediction I anticipated that this would achieve a time complexity of . I believe this to be correct when running this algorithm in the driver program. Because no significant variation showed in the Elapsed Time of the algorithm outside of the last array (n=2048), I decided to add a running total to the algorithm which involved declaring a variable sumOfJ and adding sumOfJ++ inside of the inner most for-loop. This loop ran a total of 2080 times when n=64. This is roughly . This contributes to the worst case for this algorithm leaving this with time complexity.

GREEN

* 1. My initial prediction was that this algorithm would have a time complexity of . The number of operations for each subsequent n3 value is 8 times the previous n3 value. This should coincide with the Elapsed Time which is proven through the chart below. This behavior is most apparent when the n size is 512, 1024 and 2048. When n = 512, the Elapsed Time is .1. When n is doubled the increase in number of operations increases 8 fold, which is reflected in the elapsed time of .91. When n=2048, the elapsed time increases 8 fold again to 8.04 which is consistent with an algorithm of a time complexity

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 64 | 128 | 256 | 512 | 1024 | 2048 |
| n3 | 262,144 | 2,097,152 | 16,777,216 | 134,217,728 | 1,080,045,576 | 8,589,934,592 |
| Elapsed Time | .001 | .01 | .01 | .1 | .91 | 8.04 |

RED

* 1. I predicted that this algorithm would run in linear time . The results are consistent with this as there is no noticeable increase in Elapsed Time between n sizes of 64 and 2048. In every increase in n, the elapsed time increases in parallel with the n size.