Programming Assignment

Comp 3270

Haden Stuart / has0027

*I certify that I wrote the code I am submitting. I did not copy whole or parts of it from another student or have another person write the code for me. Any code I am reusing in my program is clearly marked as such with its source clearly identified in comments.*

For the coding portion of this assignment I selected to solve the program using the java language. I used the JGRASP IDE to write and compile my code. As the instructor suggested, I used nanoseconds to calculate my runtimes for enhanced precision. This meant that my time values would need to instead be Long type instead of Int type, so I hope that will not affect my grade since my matrix is not of type Int.

Before starting the charts, the max functions must be analyzed. Since the **Math.max** function does the following: finds the first value, finds the second value, compares them, then returns one of the two, we will just assume it has a **cost of 4**. For the three-input **max** function I implemented below the algorithms, we will assume it has the best-case **cost of 7**.

**Alorithm-1**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed |
| 1 | 1 | 1 |
| 2 | 1 | n + 1 |
| 3 | 1 | Σi=1 to n (i + 1) |
| 4 | 1 | Σi=1 to n (i) |
| 5 | 1 | Σj=1 to i  Σi=1 to n (j + 1) |
| 6 | 6 | Σj=1 to i  Σi=1 to n (j) |
| 7 | 7 | Σi=1 to n (i) |
| 8 | 2 | 1 |

Multiply col.1 with col.2, add across rows and simplify

T1(n) = 1 + n + 1 + Σi=1 to n (i + 1)+ Σi=1 to n (i) + Σj=1 to i  Σi=1 to n (j + 1) +

6[Σj=1 to i  Σi=1 to n (j)] + 7[Σi=1 to n (i)] + 2

T1(n) = 4 + n + n(n+1)/2 + n + n(n+1)/2 + [n(n+1)/2] \* n + [n(n+1)/2] \* n +

6[n(n+1)/2] \* n + 7[n(n+1)/2]

T1(n) = 4 + 2n + (n2+n)/2 + (n2+n)/2 + n3 ….

T1(n) = O(n3)

**Algorithm-2**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed |
| 1 | 1 | 1 |
| 2 | 1 | n + 1 |
| 3 | 1 | n |
| 4 | 1 | Σi=1 to n (i +1) |
| 5 | 6 | Σi=1 to n (i) |
| 6 | 7 | Σi=1 to n (i) |
| 7 | 2 | 1 |

Multiply col.1 with col.2, add across rows and simplify

T2(n) = 1 + n + 1 + n + Σi=1 to n (i +1) + 6[Σi=1 to n (i +1)] + 7[Σi=1 to n (i +1)] + 2

T2(n) = 4 + 2n + n(n+1)/2 + n + 6[n(n+1)/2] + 7[n(n+1)/2]

T2(n) = 4 + 3n + (n2+n)/2 + 6(n2+n)/2 + 7(n2+n)/2

T2(n) = O(n2)

**Algorithm-3**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed |
| 1 | 4 | 1 |
| 2 | 11 | 1 |
| Steps executed when the input is a base case: 1 or 2 | | |
| First recurrence relation: T(n=1 or n=0) = T(0) = 4, T(1) = 11 | | |
| 3 | 5 | 1 |
| 4 | 2 | 1 |
| 5 | 1 | n/2 + 1 |
| 6 | 6 | n/2 |
| 7 | 7 | n/2 |
| 8 | 2 | 1 |
| 9 | 1 | n/2 + 1 |
| 10 | 6 | n/2 |
| 11 | 7 | n/2 |
| 12 | 4 | 1 |
| 13 | 4 | (cost excluding the recursive call) 1 |
| 14 | 5 | (cost excluding the recursive call) 1 |
| 15 | 11 | 1 |
| Steps executed when input is NOT a base case: 1 to 15 | | |
| Second recurrence relation: T(n>1) = 50 + 14n | | |
| Simplified second recurrence relation (ignore the constant term): T(n>1) = 14n | | |

4 + 11 + 5 + 2 + n/2 + 1 + 3n + 7n/2 + 2 + n/2 + 1 + 3n + 7n/2 + 4 + 4 + 5 + 11 = 50 + 6n + 8n = 50 + 14n

Solve the two recurrence relations using any method (recommended method is the Recursion Tree). Show your work below:

14n

Height = logn

Meaning T(n) = 14nlogn + 14n

O(nlogn)

14n/2

14n/2

14n/4

14n/4

14n/4

14n/4

14 14 14 14 14n

T3(n) = 14nlogn + 14n = O(nlogn)

**Algorithm-4**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed |
| 1 | 1 | 1 |
| 2 | 1 | 1 |
| 3 | 1 | n + 1 |
| 4 | 10 | n |
| 5 | 7 | n |
| 6 | 2 | 1 |

Multiply col.1 with col.2, add across rows and simplify

T4(n) = 1 + 1 + n + 1 + 10n + 7n + 2

T4(n) = 5 + 18n

T4(n) = O(n)

**Data Analysis**

The algorithms for this assignment were calculated to have the following complexities:

* Algorithm-1 = O(n3)
* Algorithm-2 = O(n2)
* Algorithm-3 = O(nlogn)
* Algorithm-4 = O(n)

From the data provided above in the graph, we can see that the T1(n) curve has the quickest rising pattern, followed by the T2(n) curve, then T3(n), and finally T4(n). From the calculated complexities we would assume the time data from each of the algorithms would follow this pattern with the first algorithm’s complexity (aka T1(n)) having the most prominent curve of the bunch since it is O(n3). The second curve also stands out as having a steep curve since the second algorithm is O(n2). Overall, these curves appear to follow the theoretical data for the times it would take to run each of the provided algorithms.