Ryan Harvey

Dr. Yilmaz

COMP 3270-002

19 November 2021

Programming Assignment: Algorithm Time Complexities

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.

I compiled and wrote my source code using IntelliJ IDEA.

**Algorithm-1**

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

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

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 | =1 to n(i + 1) |
| 5 | 6 | =1 to n(i) |
| 6 | 7 | =1 to n(i) |
| 7 | 2 | 1 |

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

T2(n) = O(n2)

**Algorithm-3**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed in any single recursive call |
| 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) = 4 when n = 0 and 11 when n = 1 | | |
| 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 | 17 | 1 |
| Steps executed when input is NOT a base case: 1 to 15 | | |
| Second recurrence relation: T(n>1) = 14n + 56 | | |
| Simplified second recurrence relation (ignore the constant term): T(n>1) = 14n | | |

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

MaxSum works on each side of the respective array it is calculating hence it can be described with n / 2. Therefore we can use the Recursion-Tree method(4.4)

Diagram, schematic

Description automatically generated

T3(n) = O(n log(n))

**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) = O(n)

The predicted time complexity curves do match the data collected from the source code. Looking at T1, T2, T3, and T4, the shape of their curves matches the order in which they were calculated on this document. T1 obviously grows much larger because it is O(n3) but not as fast as the O(n2) algorithm complexity T2. The same can be said for T3 and T4. Note that T4 stays completely flat because it is O(n), so the time it takes to compute the data set is exactly proportional to how much data there is.

Also, the last four columns are being scaled up by 1000 to fit inside the graph and make visually representing the data easier

It is also interesting to note the first four lines (the pure time) waver unlike the complexity lines. This is because of the randomness of the data. These lines should change very slightly every time you run the program while the complexity lines will stay almost exactly the same.

Chart, line chart

Description automatically generated