Samuel Adams and Brian Shin

sja0029@auburn.edu

bcs0058@auburn.edu

COMP 3270 PROGRAMMING ASSIGNMENT

**Algorithm-1**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed |
| 1 | 1 | 1 |
| 2 | 1 | n + 1 |
| 3 | 1 |  |
| 4 | 1 |  |
| 5 | 1 |  |
| 6 | 6 |  |
| 7 | 4 |  |
| 8 | 2 | 1 |

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

T1(n) = 1 + (n + 1) + () + () + () + 6 () + 4 () + 2

4 + n + (n(n+1)/2 + 1) + n(n+1)/2 + ((n(n+1)(n+2)/6) + n) + 6(n(n+1)(n+2)/6) + 4(n(n+1)/2)

5 + n + (n2+n / 2) + (n2+n / 2) + 4(n2+n / 2) + 7(n(n+1)(n+2)/6) + n

5 + 2n + 6(n2+n / 2) + 7(n(n+1)(n+2)/6)

5 + 2n + 6(n2+n / 2) + 7(n(n2+ 3n + 2)/6)

5 + 2n + 6(n2+n / 2) + 7(n3 + 3n2 + 2n)/6)

5 + 2n + 6(n2+n / 2) + 7(n3 + 3n2 + 2n)/6)

5 + 2n + 3(n2 + n) + 7/6(n3 + 3n2 + 2n)

5 + 2n + 3n2 + 3n + 7/6(n3 + 3n2 + 2n)

5 + (5n + 7/3n) + (3n2 + 7/2n2) + 7/6n3

5 + 22/3n + 13/2n2 + 7/6n3

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 |  |
| 5 | 6 |  |
| 6 | 4 |  |
| 7 | 2 | 1 |

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

T2(n) = 1 + (n + 1) + n + ( ) + 6() + 5( + 2

5 + n + n + 12(

5 + 2n + 12(n(n+1)/2)

5 + 2n + 12(n2+n)/2

5 + 2n + 6(n2+n)

5 + 8n + 6n2

T2(n) = O(n2)

**Algorithm-3**

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

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

A piece of paper with writing on it

Description automatically generated with medium confidence

T3(n) = O(nlogn)

**Algorithm-4**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed |
| 1 | 1 | 1 |
| 2 | 1 | 1 |
| 3 | 1 | n + 1 |
| 4 | 8 | n |
| 5 | 4 | n |
| 6 | 2 | 1 |

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

T4(n) = 1 + 1 + n + 1 + 8n + 4n + 2

5 + 13n

T4(n) = O(n)

Total Graph of all Algorithms

This graph shows the empirical and theoretical data of each Algorithm.

Algorithm 1 Chart

Our predicted complexity for Algorithm 1 was O(n3 ) however through testing it seems that the Algorithm performed much better than that on the lower Input sizes. On the higher input sizes, the Algorithm did see a sharp increase, so it would be safe to say that the two curves would look similar past input sizes of 100.

Algorithm 2 Chart

Our predicted complexity for Algorithm 2 was O(n2 ). Both of the curves appear to rise at similar rates, especially at the beginning.

Algorithm 3 Chart

Our predicted complexity for Algorithm 3 was O(nlogn). The Empirical complexity definitely met our expectation, as the graph is quite similar to that of a nlogn graph.

Algorithm 4 Chart

Our Predicted complexity for Algorithm 4 was O(n) and it seems to have met that expectation. The Time taken is very low, and the two graphs are very similar.