**Algorithm-1**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed |
| 1 | 1 | 1 |
| 2 | 1 | n+1 |
| 3 | 1 | (n2/2)+(3n/2) |
| 4 | 1 | (n2/2)+(n/2) |
| 5 | 1 | (n3/2)+(5n/2) |
| 6 | 6 | (n3/2)+(3n/2) |
| 7 | 4 | (n2/2)+(n/2) |
| 8 | 2 | 1 |

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

T1(n) = 1 + n + 1 + (n2/2) + (3n/2) + (n2/2) + (n/2) + (n3/2) + (5n/2) + 6(n3/2) + 6(3n/2) + 4(n2/2) + 4(n/2) + 2

= **(7n3/2) + 3n2 + (33/2)n + 3 = 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 | (n2/2)+(3n/2) |
| 5 | 6 | (n2/2)+(n/2) |
| 6 | 4 | (n2/2)+(n/2) |
| 7 | 2 | 1 |

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

T2(n) = 1 + n + 1 + n + (n2/2)+(3n/2) + 3n2 + 3n + 2n2 + 2n + 2 = n2/2 + 3n/2 + 7n + 4 = **n2/2 + 17n/2 + 4 = O(n2)**

**Algorithm-3**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed in any single recursive call |
| 1 | 3 | 1 |
| 2 | 3 | 1 |
| Steps executed when the input is a base case: 1- 2 | | |
| First recurrence relation: T(n=1 or n=0) = 12 | | |
| 3 | 5 | 1 |
| 4 | 2 | 1 |
| 5 | 1 | (n/2)+1 |
| 6 | 6 | n/2 |
| 7 | 4 | n/2 |
| 8 | 2 | 1 |
| 9 | 1 | (n/2)+1 |
| 10 | 6 | n/2 |
| 11 | 4 | n/2 |
| 12 | 4 | 1 |
| 13 | 5 | (cost excluding the recursive call) 1 |
| 14 | 6 | (cost excluding the recursive call) 1 |
| 15 | 5 | 1 |
| Steps executed when input is NOT a base case: 1, 3-15 | | |
| Second recurrence relation: T(n>1) = 2T(n/2) + 12n + 37 | | |
| Simplified second recurrence relation (ignore the constant term): T(n>1) = 2T(n/2) + 12n | | |

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

Diagram

Description automatically generated

T3(n) = 12n ((log2n) + 1) = **12nlog2n + 12n = 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 = (n+1)+12n+4 = **13n+5 = O(n)**

After taking the data from SarahPham\_phw\_output.txt and displaying the data on excel. We get this graph (next page):

Table

Description automatically generated

Chart, line chart

Description automatically generated

The above chart shows the rate of growth for each algorithm. It also shows the rate of the growth of the calculated T(n) based on counting the cost of each line and the total # of executions. It shows that my calculations of T(n) are either not accurate or the average running time (measured by the system clock) may not be as accurate. This chart shows that T1(n) and algorithm-1 are the least efficient. Then the chart shows that algorithm-2 is the second least efficient. The most efficient being algorithm-3, which was as expected. However, the T1(n) curve is potentially way higher than the actual measured running time for algorithm-1. This shows that my calculations must be inaccurate, or my running time measurements were inaccurate due to my computers internal clock (However I did set N = 500 to prevent that).