**COMP3270-002 Programming Assignment**

Fall 2021

Will May (wdm0032@auburn.edu)

**Tables provided in instruction document:**

**Algorithm-1**

|  |  |  |
| --- | --- | --- |
| Step | Cost of each execution | Total # of times executed |
| 1 | 1 | 1 |
| 1 | 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

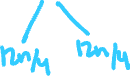
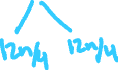
= n2/2 + 17n/2 + 4

= 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 to 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:



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)

**Data collected from program output:**

**Table

Description automatically generated**

Chart, line chart

Description automatically generated

**Explanation:**

The data above shows the rate at which each algorithm runs/grows. From the graph data, it is noted how Algorithm-1 and T1(n) are the least efficient out of all the other algorithms and complexity orders. This was expected, as the overall time complexity is O(n3). Algorithm-2 and T2(n) were the second least efficient due to their overall time complexity of O(n2). This can also be seen directly from the graph data. Algorithm-4 and T4(n) were the third least efficient algorithm and complexity order due to the time complexity being O(n). This can also be seen from the graph data, where Algorithm-4’s curve is very horizontal or linear. Finally, Algorithm-3 and T3(n) were the most efficient with a time complexity of O(nlogn), where the graph data also supports this time complexity.