Gabriel Emerson

Programming Assignment

Comp 3270

For the programming part of this project, I decided to use Java language, mostly due to the fact it is where I have the most practice and have a very useful tool in JGRASP to help me get things right. I used JGRASP to create, edit, compile, debug, and run my program. I decided to use nanoseconds for my runtimes, this should help me to get more precise answers. In doing this I struggled to create my matrix and get the numbers out correctly, in the end I ended up having to use type Long instead of Int, however the number themselves are still integers.

For the first algorithm I had to look up the max function to determine what the exact cost of that function would be. I decided this function finds the first value, then second, then compares them, and returns one. This gives a cost of 4. However, we used a 3-input max function so for this Algorithm we will assume max function has a total cost of 7.

Alorithm1

Step	Cost of each execution	Total # of times executed
1	1	1
2	1	n + 1
3	1	$\Sigma_{i=1 \text{ to } n} (i + 1)$
4	1	$\Sigma_{i=1 \text{ to } n} (i)$
5	1	$\Sigma_{j=1 \text{ to } i}\ \Sigma_{i=1 \text{ to } n} (j + 1)$
6	6	$\Sigma_{j=1 \text{ to } i}\ \Sigma_{i=1 \text{ to } n} (j)$
7	7	$\Sigma_{i=1 \text{ to } n} (i)$
8	2	1

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

$T_1(n) = 1 + n + 1 + \Sigma_{i=1 \text{ to } n} (i + 1) + \Sigma_{i=1 \text{ to } n} (i) + \Sigma_{j=1 \text{ to } i}\ \Sigma_{i=1 \text{ to } n} (j + 1) +$

$6[\Sigma_{j=1 \text{ to } i}\ \Sigma_{i=1 \text{ to } n} (j)] + 7[\Sigma_{i=1 \text{ to } n} (i)] + 2$

$T_1(n) = 4 + n + n(n+1)/2 + n + n(n+1)/2 + [n(n+1)/2] * n + [n(n+1)/2] * n +$

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

$T_1(n) = 4 + 2n + (n^2+n)/2 + (n^2+n)/2 + n^3$ ….

$T_1(n) = O(n^3)$

Algorithm2

Step	Cost of each execution	Total # of times executed
1	1	1
2	1	n + 1
3	1	n
4	1	$\Sigma_{i=1 \text{ to } n} (i+1)$
5	6	$\Sigma_{i=1 \text{ to } n} (i)$
6	7	$\Sigma_{i=1 \text{ to } n} (i)$
7	2	1

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

$T_2(n) = 1 + n + 1 + n + \Sigma_{i=1 \text{ to } n} (i+1) + 6[\Sigma_{i=1 \text{ to } n} (i+1)] + 7[\Sigma_{i=1 \text{ to } n} (i+1)] + 2$

$T_2(n) = 4 + 2n + n(n+1)/2 + n + 6[n(n+1)/2] + 7[n(n+1)/2]$

$T_2(n) = 4 + 3n + (n^2+n)/2 + 6(n^2+n)/2 + 7(n^2+n)/2$

$T_2(n) = O(n^2)$

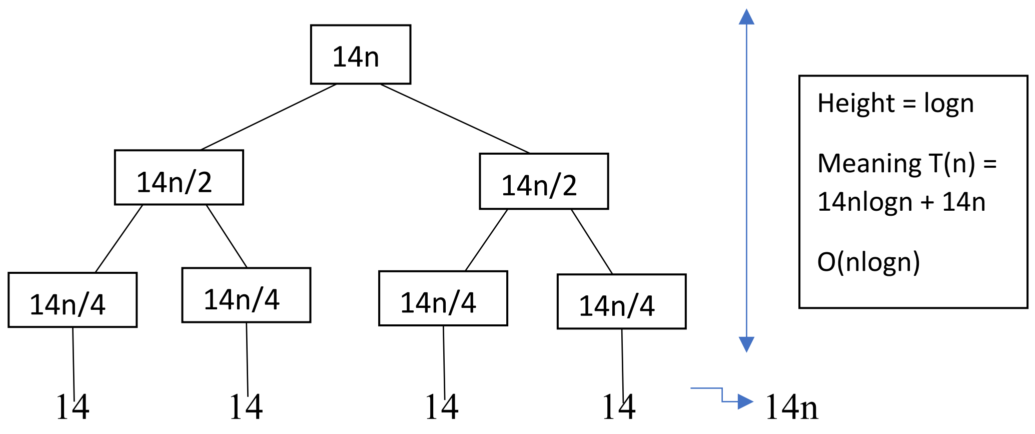
Algorithm3

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:



$T_3(n) = 14n\log n + 14n = O(n\log n)$

Algorithm4

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

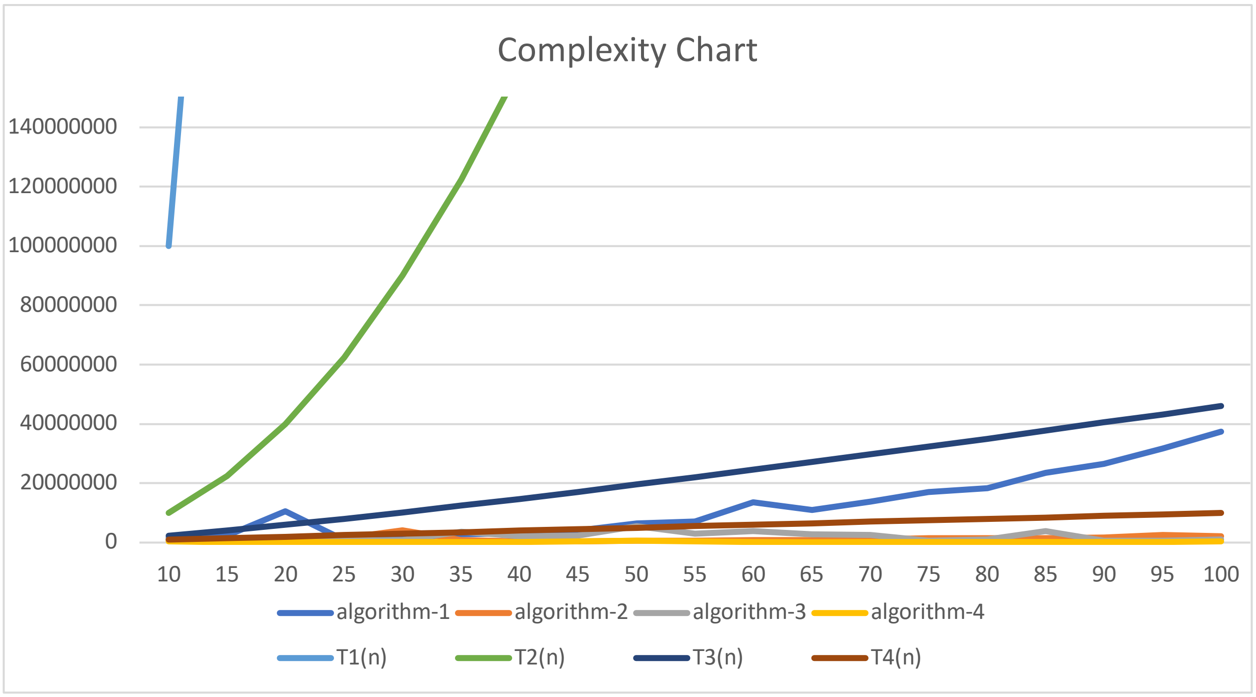
Multiply col.1 with col.2, add across rows and simplify

$T_4(n) = 1 + 1 + n + 1 + 10n + 7n + 2$

$T_4(n) = 5 + 18n$

$T_4(n) = O(n)$

Data Analysis



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

- ⇨ Algorithm-1 = $O(n^3)$
- ⇨ Algorithm-2 = $O(n^2)$
- ⇨ Algorithm-3 = O(nlogn)
- ⇨ Algorithm-4 = O(n)

From the data in the graph, we see that the quickest rising pattern is from T1(n). This is followed by T2(n), T3(n), then lastly T4(n). We assume that this will also be the case for calculated complexities, with T1(n) having the biggest curve since it is O(n^3). The second curve, T2(n) also makes sense since it is O(n^2). Overall, all of the curves seem to match up to the graph. This means the curves follow the theoretical data for times it would take to run each of the algorithms.