Programming Assignment:

Theoretical and Empirical Analysis of 4 MSCS Algorithms

Austin Osborn

Algorithm-1

Step	Cost of each execution	Total # of times executed
1	1	1
2	1	n+1
3	1	[n(n+1)/2] + 1
4	1	n(n+1)/2
5	1	(n(n+1)(n+2)/6) + 1
6	5	(n(n+1)(n+2)/6)
7	5	(n(n+1))/2
8	2	1

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

T.(n) = $n^3 + \frac{13}{2}n^2 + \frac{13}{2}n + 6$

Algorithm-2

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

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

T.(n) = $6n^2 + 8n + 4$

Algorithm-3

Step	Cost of each execution	Total # of times executed in any single recursive call
1	4	1
2	8	1
Steps executed when the input is a base case:		MAX(4,11) ⇝ 11
First recurrence relation: T(n=1 or n=0) =		MAX(4,11) ⇝ 11
3	5	1
4	2	1
5	1	(n/2) + 1
6	6	n/2
7	5	n/2
8	2	1
9	1	(n/2) + 1
10	6	n/2
11	5	n/2
12	4	1
13	1	(cost excluding the recursive call) 1
14	1	(cost excluding the recursive call) 1
15	6	1
Steps executed when input is NOT a base case:		12n + 24
Second recurrence relation: T(n>1) =		2T(n/2) + 12n + 24
Simplified second recurrence relation (ignore the constant term): T(n>1) =		2T(n/2) + 12n

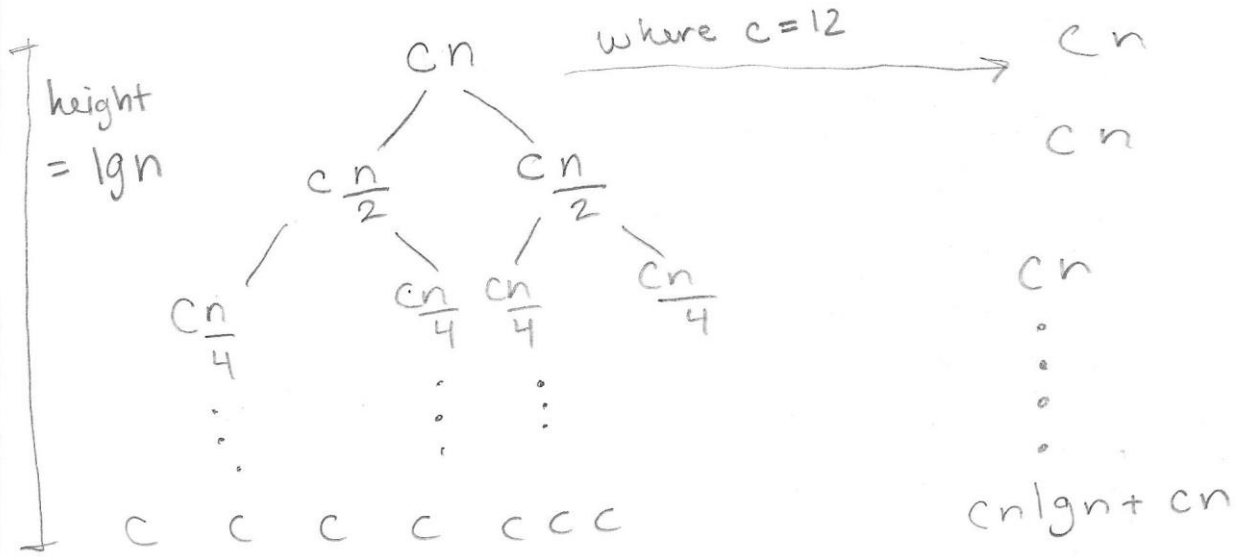
Solve the two recurrence relations using any method (recommended method is the Recursion Tree). Show your work below:

Recursion tree attached below

T.(n) = $12n\lg n + 12n + 11$

Algorithm 3 Recursion tree

$T(n>1) = 2T(n/2) + 12n$



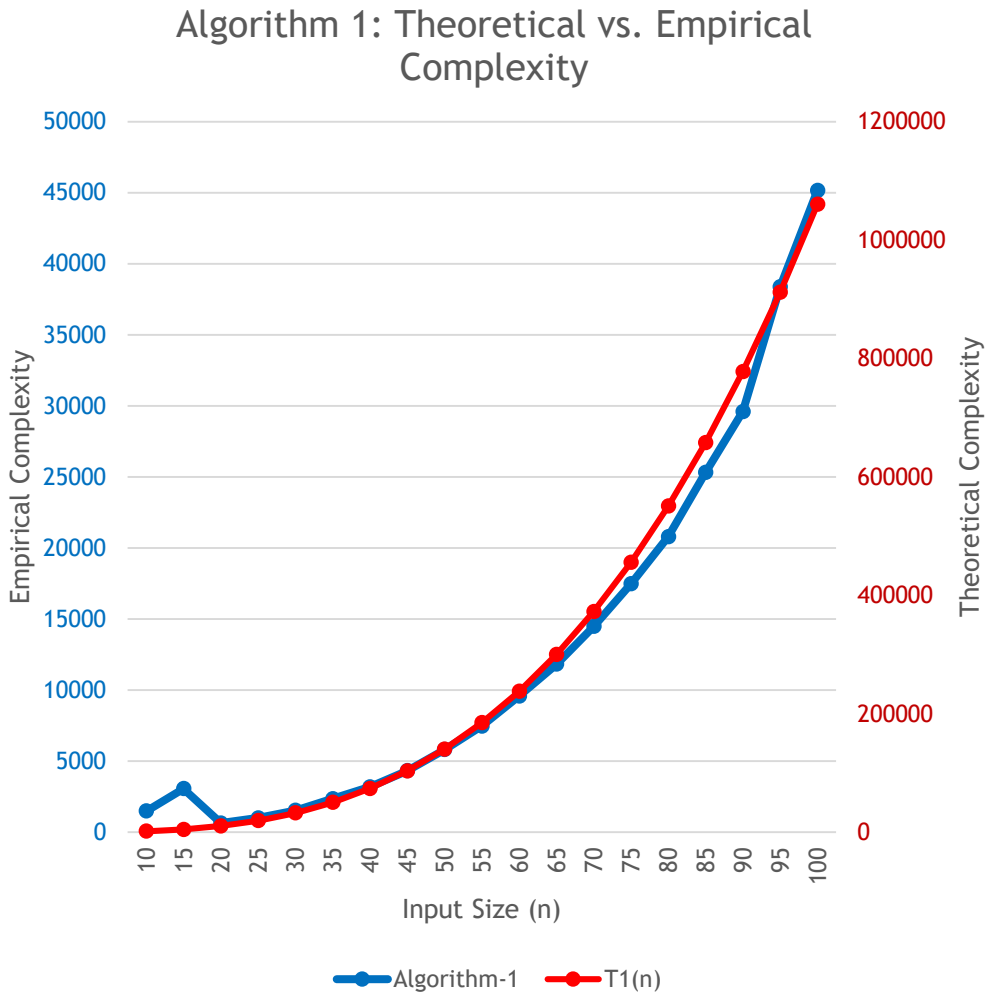
$= 12n\lg n + 12n$

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	5	n
6	1	1

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

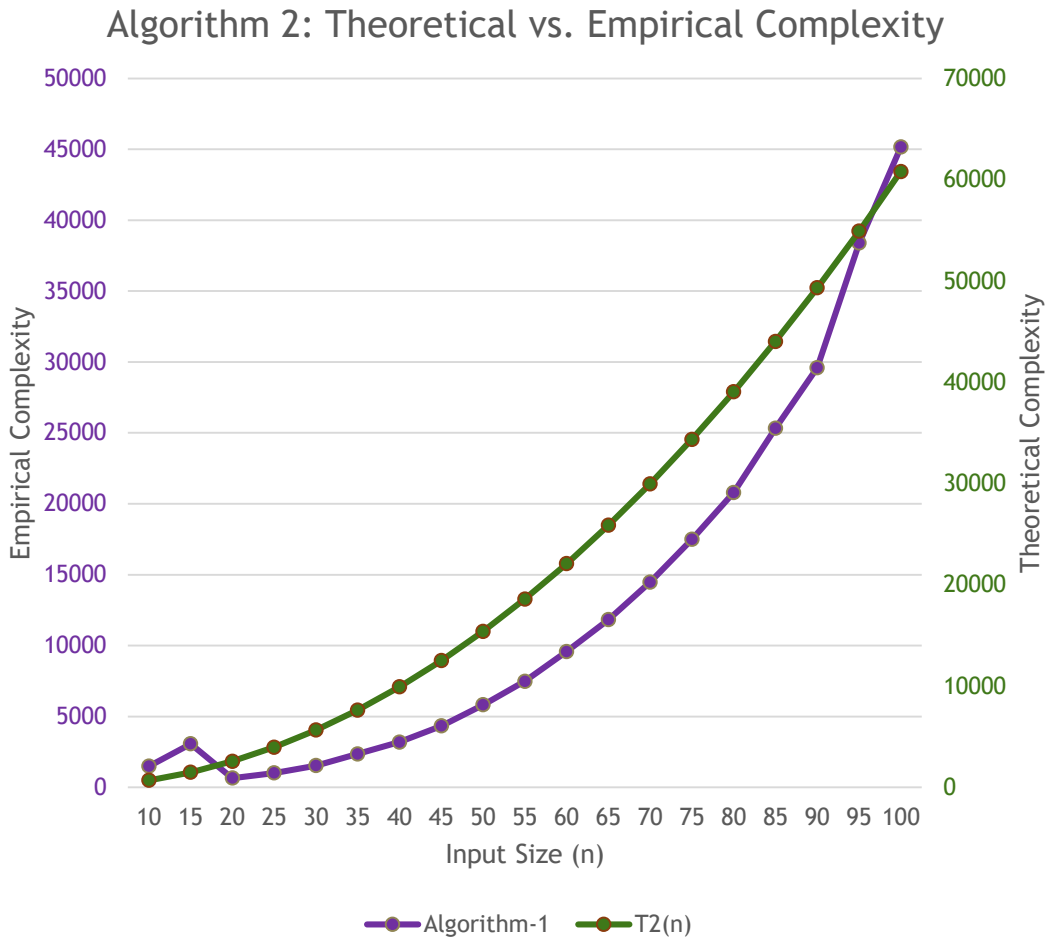
$T_4(n) =$ $14n + 4$



Algorithm 1: T(n) = $n^3 + \frac{13}{2}n^2 + \frac{13}{2}n + 6$

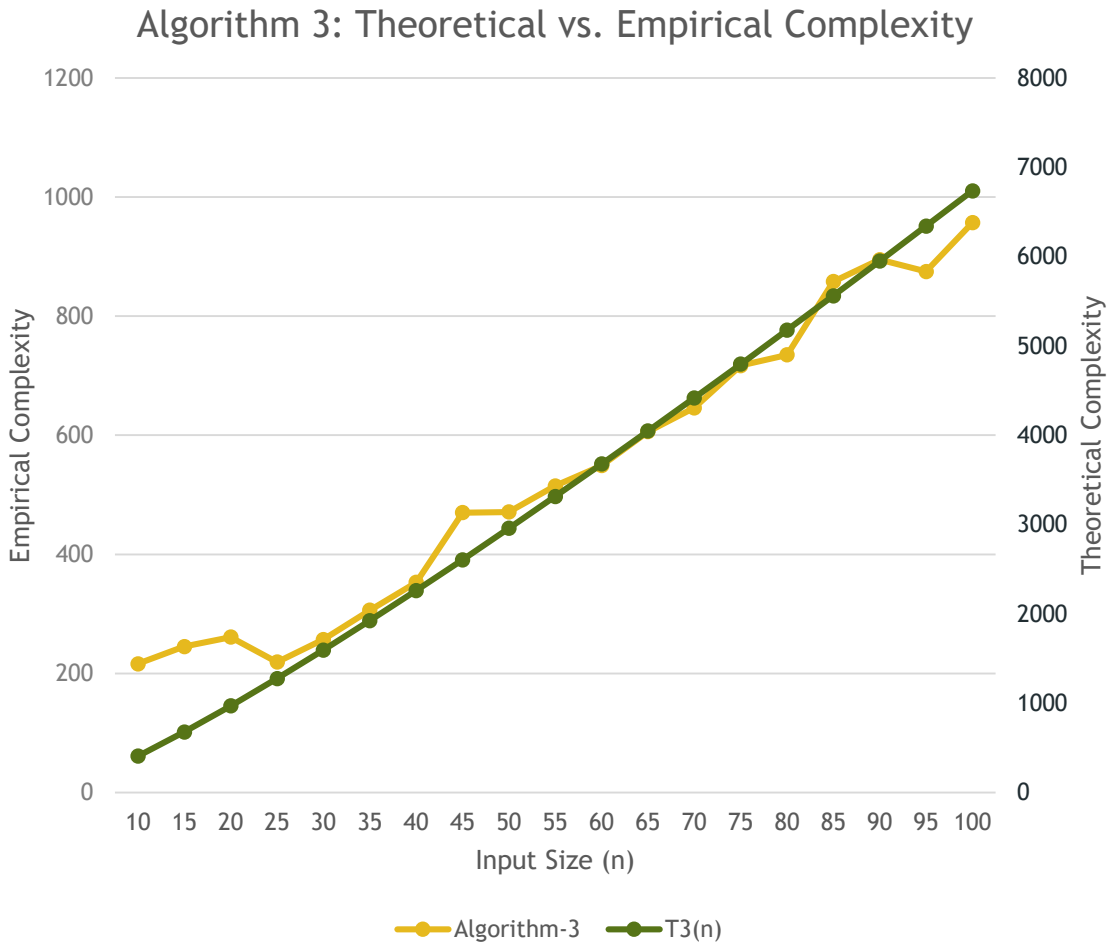
The worst performing algorithm of the group.

Has polynomial complexity of degree 3, meaning it does not scale for large data sets.



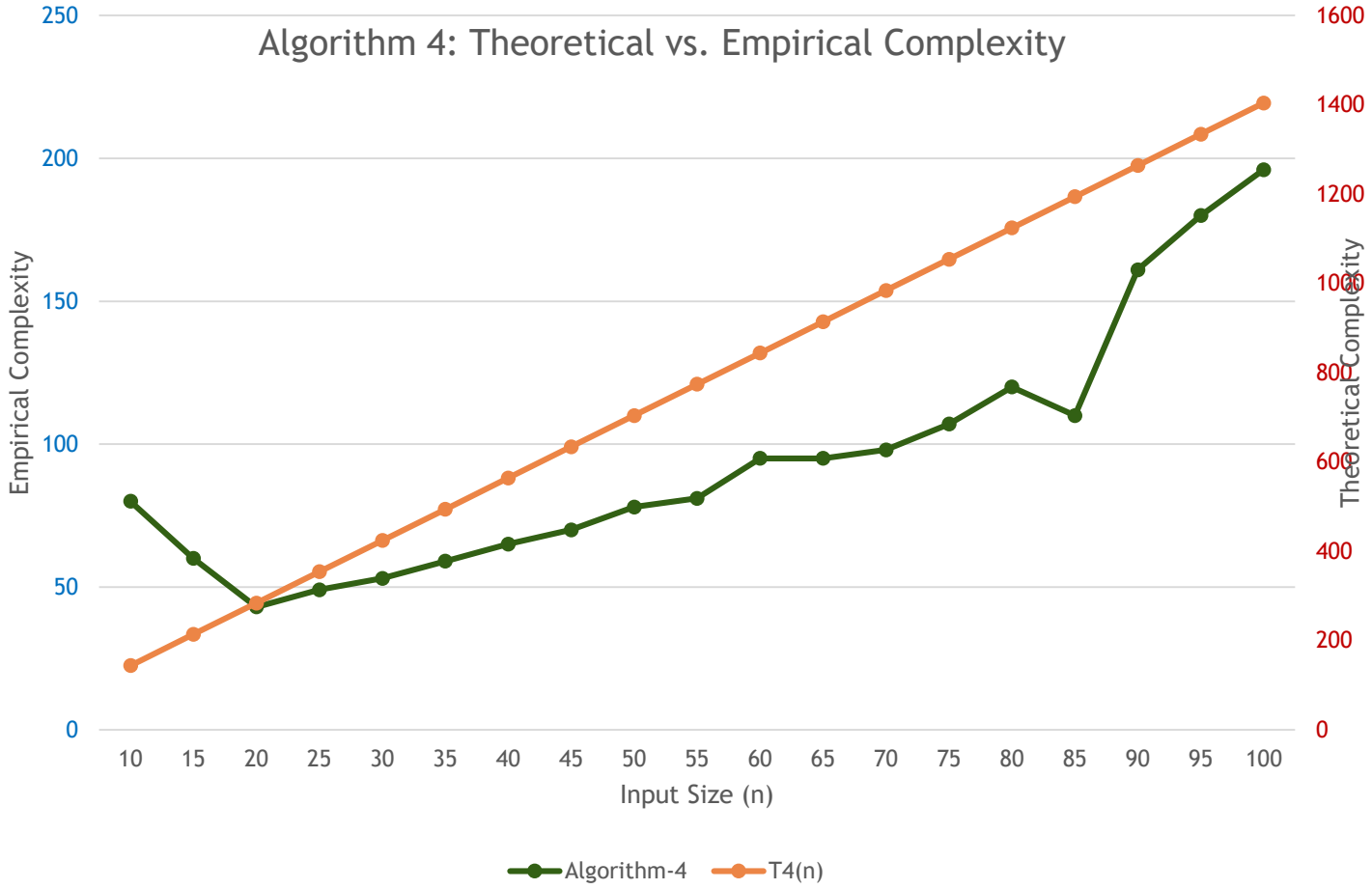
Algorithm 2: T(n) = $6n^2 + 8n + 4$

Algorithm 2 has quadratic complexity, which is an improvement over Algorithm 1's cubic complexity, but still not great for large data sets.



Algorithm 3: T(n) = $12nlgn + 12n + 11$

The divide-and-conquer Algorithm 3 has nlgn complexity, which is much better than polynomial time Algorithms 1 and 2, but slightly worse than linear time.



Algorithm 4 is a linear time algorithm, making it the best out of all 4 algorithms in terms of scalability. It is the best order that can be obtained to solve the MSCS because all terms must be considered at least once.

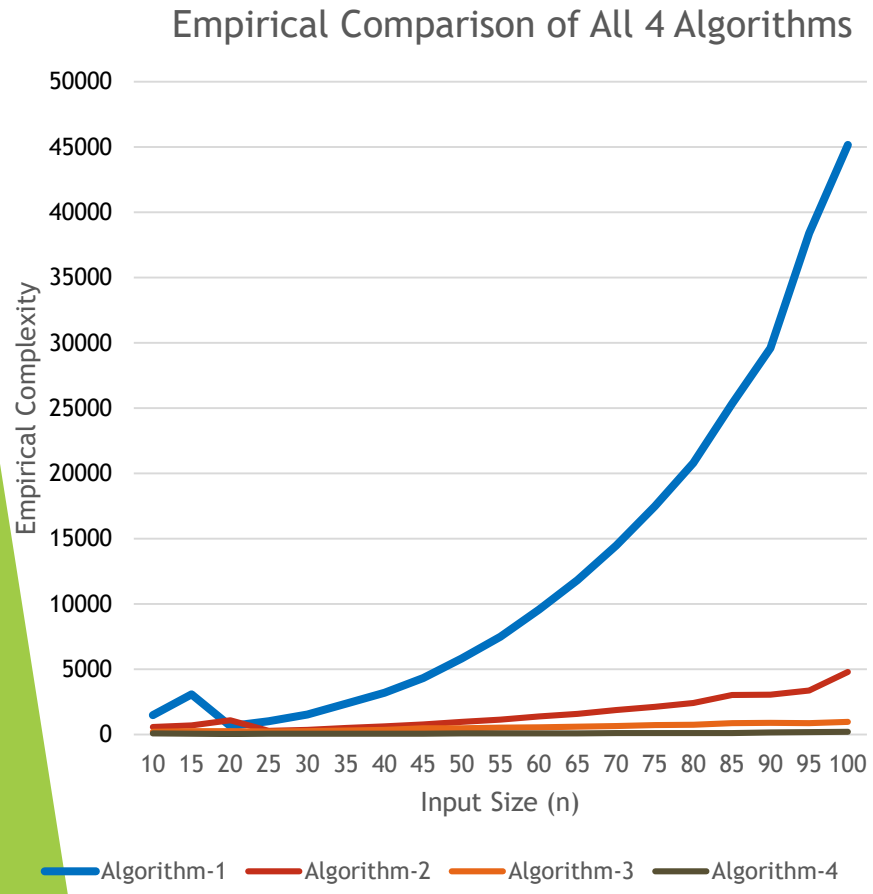
Algorithm 4: T(n) = $14n + 4$

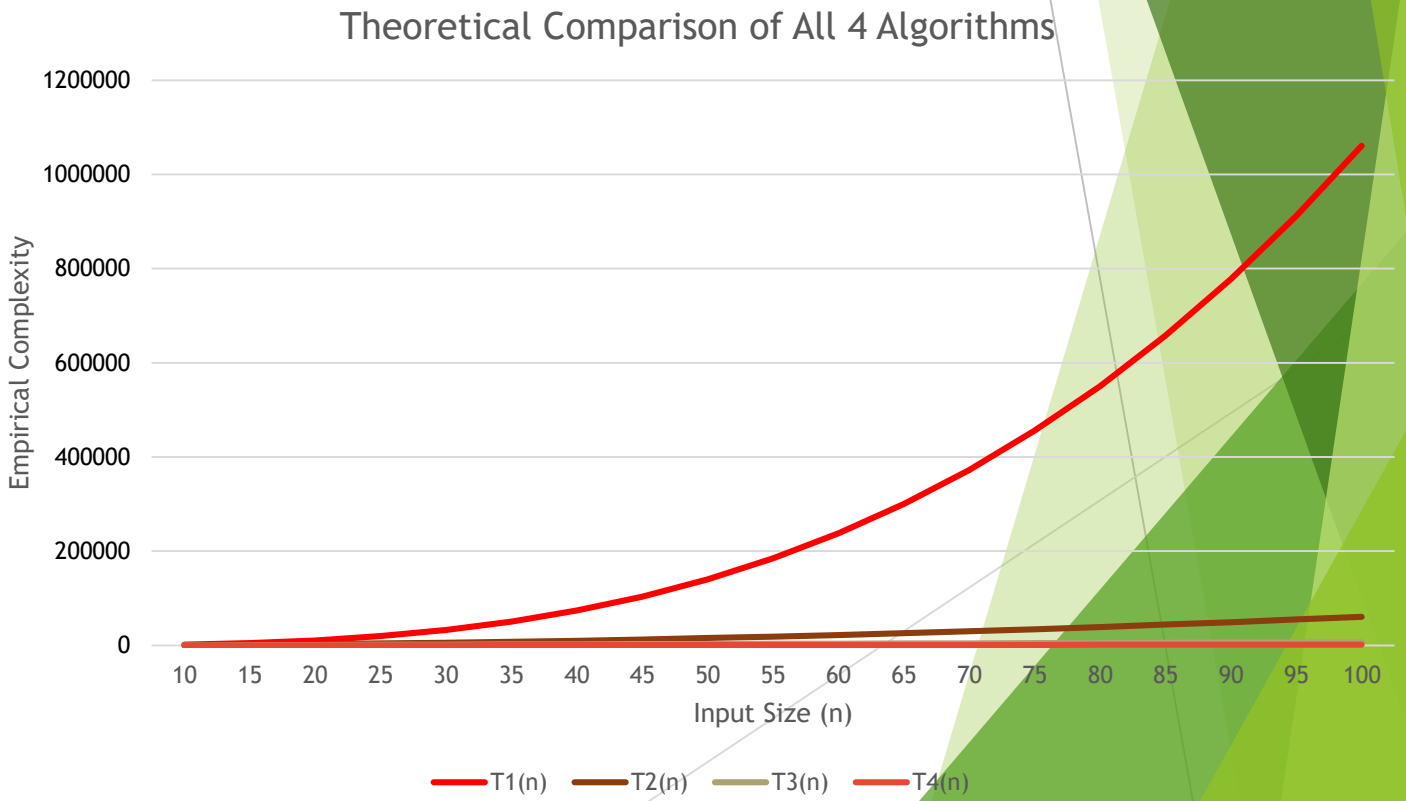
Combined Graphs

The combined graphs (split into theoretical and empirical comparisons for scaling purposes) show the vast differences between these four algorithms in terms of complexity.

Algorithm 1: O(n^3)
Algorithm 2: O(n^2)
Algorithm 3: O(nlgn)
Algorithm 4: O(n)

Algorithm 1 makes all other algorithms seem efficient by comparison. On the other hand, Algorithm 4 is so efficient, that it can barely be discerned from the others when all graphed on a single chart.





What was to be expected?

- As you can see from the charts, all algorithms performed empirically closely to their predicted theoretical values.
- The more complex the algorithm, the more closely the theoretical line traced with the empirical line.
- The one graph to point out with meaningful differences is that of Algorithm 4
  - This graph starts high and has some variation from the linear structure in between.
    - This is due to the fact that it's linear time complexity with a data set of n = 100 on a fast computer is very sensitive to the processes surrounding it, as it is insignificant when compared to the total operation of the processor running the operating system.
    - Modern processors also increase their clock speeds dynamically based on calculations being preformed (ex. Go from 1.2Ghz to 4.0Ghz depending on utilization), and these on the fly adjustments would explain why the empirical data starts higher then dips.