Chandler Dykes and Sebastian Alger

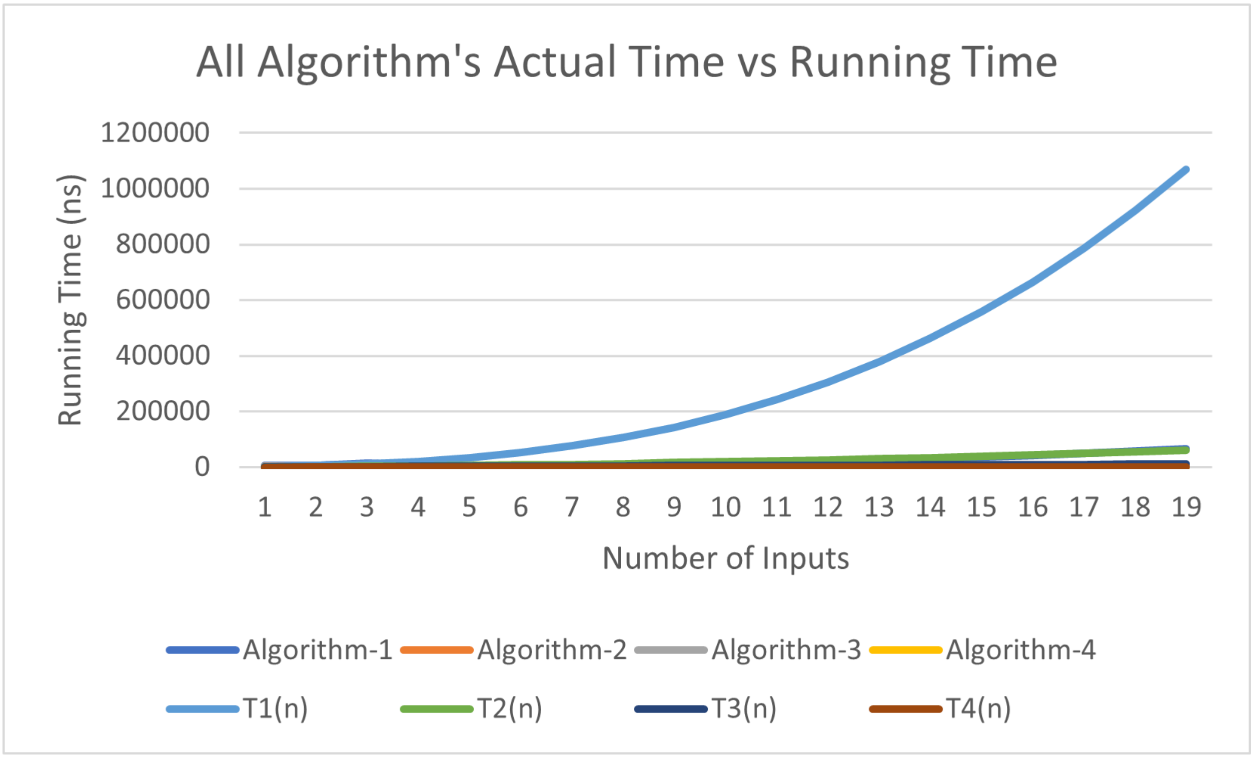
Programming Assignment

Intro to Algorithms 3270

11/18/2022

For this assignment, we used different algorithms of varying complexities to find the **Maximum Sum Contiguous Subvector** problem.

Here is a graph detailing all of the algorithm's actual running times, versus their expected running times:



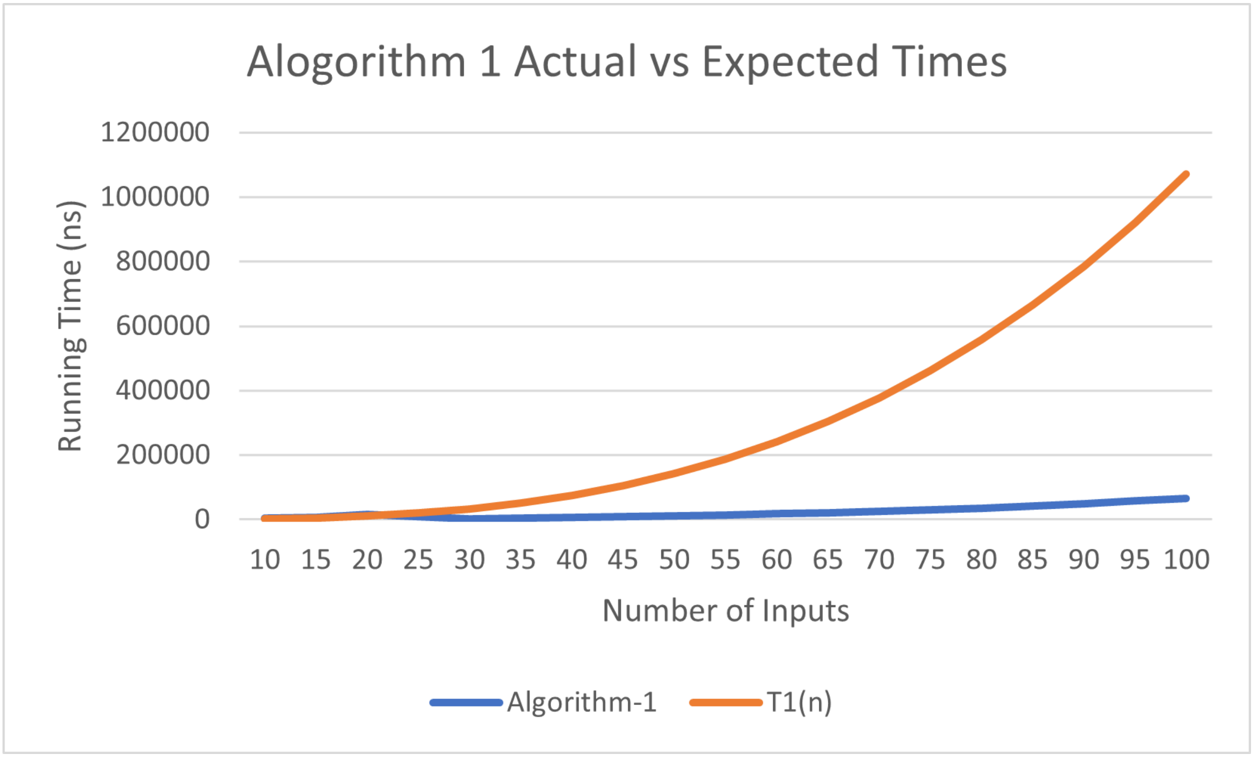
Now, we will talk about each specific program's expected running time vs actual running time and try to explain any irregularities if any appear.

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	6	(n(n+1)(n+2)/6)
7	6	(n(n+1)/2)
8	2	1

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

$T_1(n) = ((7n^3 + 45n^2 + 44n + 36) / 6) = O(n^3)$

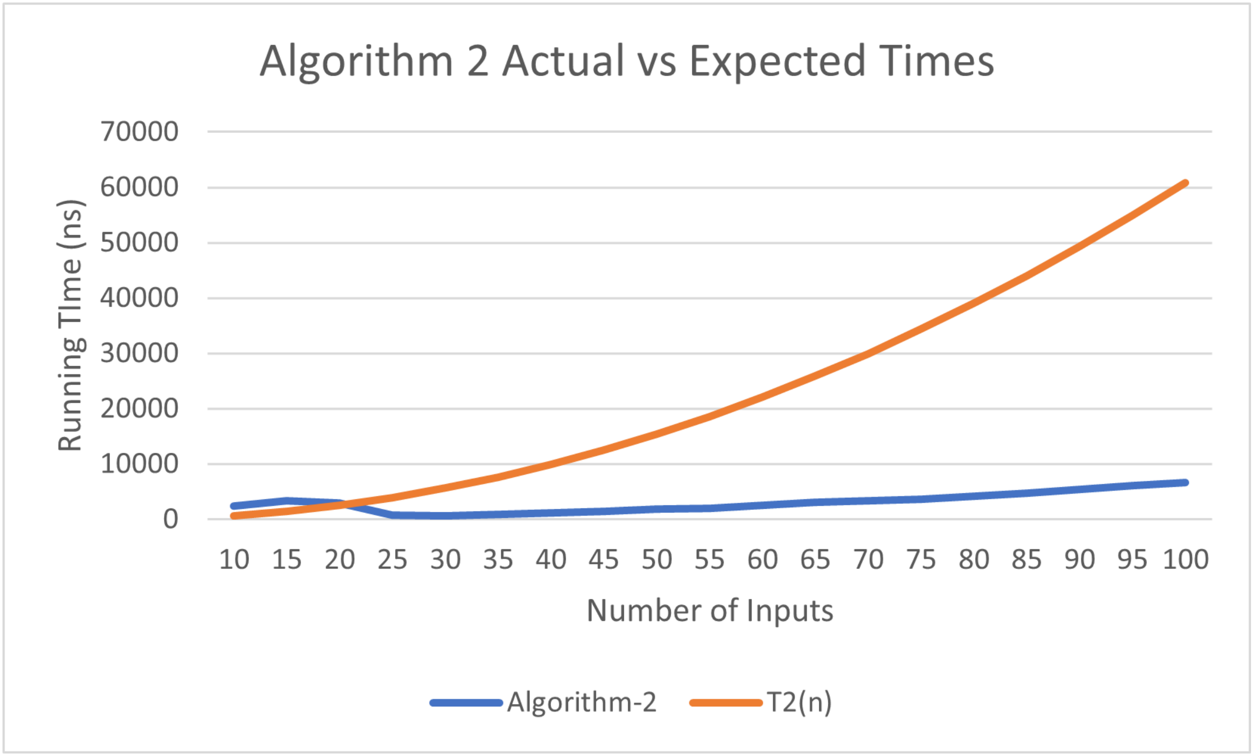


Conclusion: We can see that apart from the very beginning, the algorithm runs as expected. Its expected time complexity is O(n^3) and it stays less than that upper bound.

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	7	(n(n+1)/2)
7	2	1

Multiply col.1 with col.2, add across rows and simplify
$T_2(n)$ =7n^2 + 9n + 5 = O(n^2)



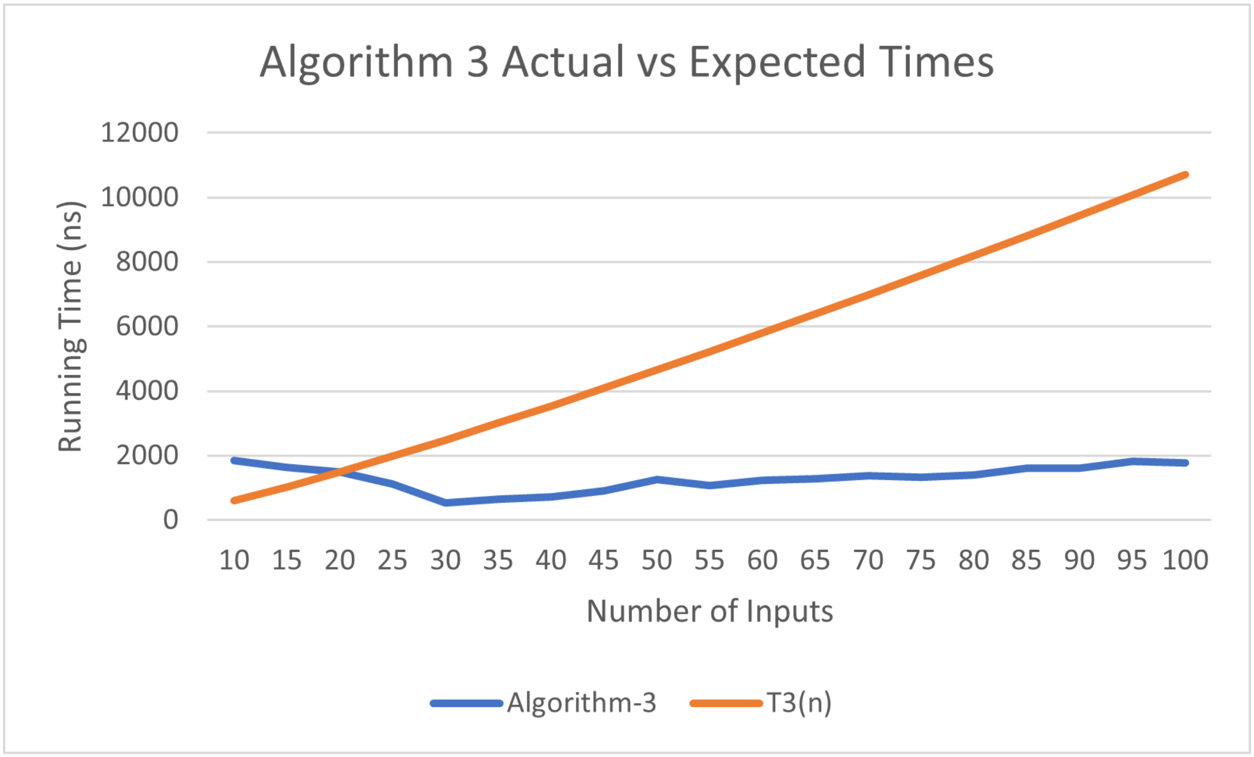
Conclusion: Same as the first algorithm, this one goes above the upper bound at the beginning, but then goes back down and stays beneath the upper bound of O(n^2).

Algorithm 3:

Step	Cost of each execution	Total # of times executed in any single recursive call
1	4	1
2	4	1
Steps executed when the input is a base case: 1-2		
First recurrence relation: $T(n=1 \text{ or } n=0) = 5 \text{ or } 16$		
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	5	(cost excluding the recursive call) 1
14	6	(cost excluding the recursive call) 1
15	17	1
Steps executed when input is NOT a base case:1-15		
Second recurrence relation: $T(n>1) = 14n + 43$		
Simplified second recurrence relation (ignore the constant term): $T(n>1) = 14n$		

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

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



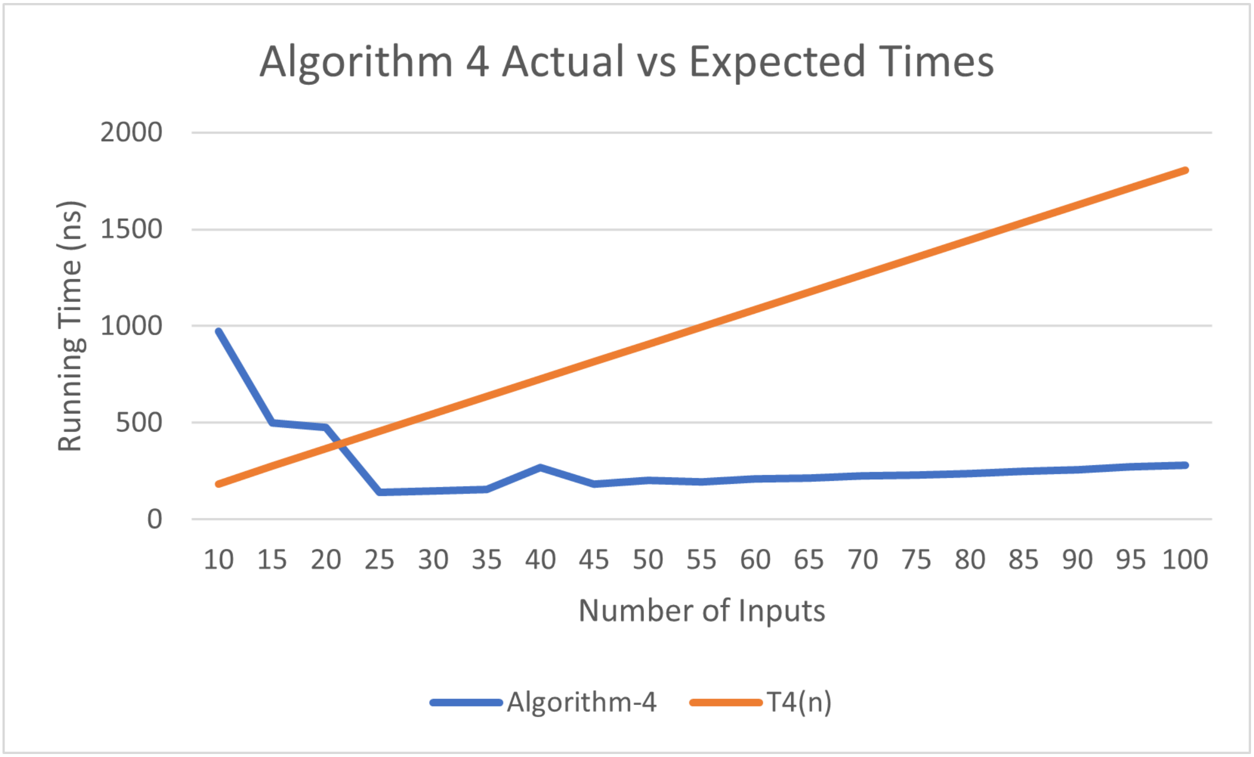
Conclusion: Once again, the program goes above the upper limit of O(nlogn) during the beginning, but then goes back beneath the curve and runs as expected.

Algorithm 4:

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

$T_4(n) = 1 + 1 + (n+1) + 10n + 7n + 2 = 18n + 5 = O(n)$



Conclusion: This algorithm drastically showed the weird behavior mentioned where it goes above the expected upper limit (O(n) in this case). Just like all of the other algorithms, however, it drops back down to expected levels.

Overall: All of the algorithms went through a phase in the beginning where they went above their expected running times. We could only imagine that this is due to some weird facet of Java or how the program was being run. However, all of the algorithms dropped below their respective upper bounds provided by the T(n) calculations and behaved as expected once they got past the beginning few inputs.