Tea Elming
1DV018
Sept 2024

Question 7 – Comparing Sum3 Algotrithms

Pre-Information

To allow for effective testing of the two versions of 3Sum, various programs were written. Initially, the tests were designed to satisfy the requirements included in the MyMoodle instructions[1] which mainly focused on looking at growth and whether the functions behaved as hypothesised prior to testing. However, additional instructions were added on Slack[2] which required a different form of comparisons, which is why the approach was altered throughout the process of testing. As the initial testing results could be considered quite interesting from a discussion point of view, these will be included as well.

Introduction

The algorithm for 3Sum is a classic problem which aims to find all unique triplets in an array that sum to 0. The two implementations have both been designed to take the summed value as an argument, which was an active choice to allow for more flexibility in the usage. However, for the purpose of the testing, the sum was always 0 for the final 3Sum.

The brute force method, called Sum3_Brute in this project, iterates through all values of the array to find unique combinations, meaning it is quite resource heavy. Due to the lack of sorting, it performs redundant checks, and due to it having an upper execution time boundary of ON^3 it means that it is not very efficient for larger data sets.

The improved version, in this case called Sum3_Imp, utilises the two-point method, which is a variation of the 2Sum problem that has been implemented to improve the 3Sum. It sorts the array and splits it so that each iteration compares an element to the sum of two elements of the second part of the array. The two elements of the second part of the array are dictated by two pointers, which start of with the first and last element, and then move depending on if the sum is the inverse of the first element outside the split array, too low, or too high. This brings the upper limit of the time required down to ON^2 rather than ON^3, and the initial sorting of the array means that no unnecessary duplicates are checked.

Measurements and Analysis

General Observations

[1] **Uppgift 7:** Testa dina implementationer av 3sum med olika storlekar på vektorer/listor och mät hur lång tid de tar. Använd dessa mätvärden för att empiriskt avgöra hur de växer. Stämmer dina resultat överens med vad du förväntade dig? Varför/varför inte?

[2] https://coursepress.slack.com/archives/C03TXFZ8G0Z/p1726142521406199?thread_ts=1726139923.166139&cid=C03TXFZ8G0Z

Tea Elming
1DV018
Sept 2024

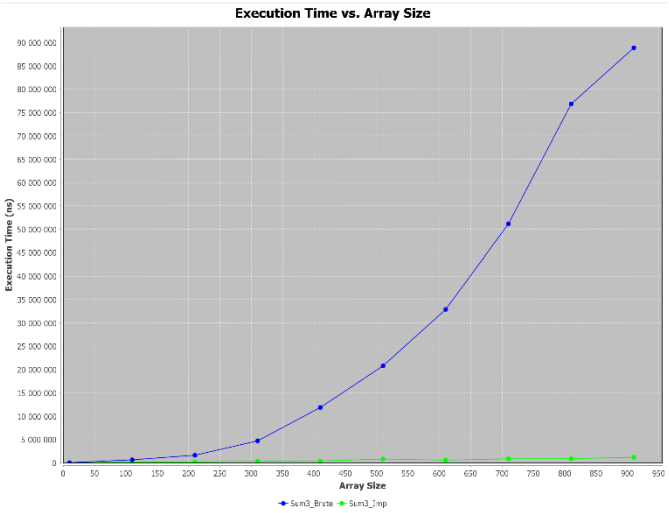


Figure 1 - Comparison of 3Sum implementations (50 runs)

int startSize = 10; *// smallest array size*

int endSize = 1000; *// largest array size*

int increment = 100; *// Increment step*

int runTimes = 50; *// Multiple runs per size*

int targetSum = 0; *// Sum to find*

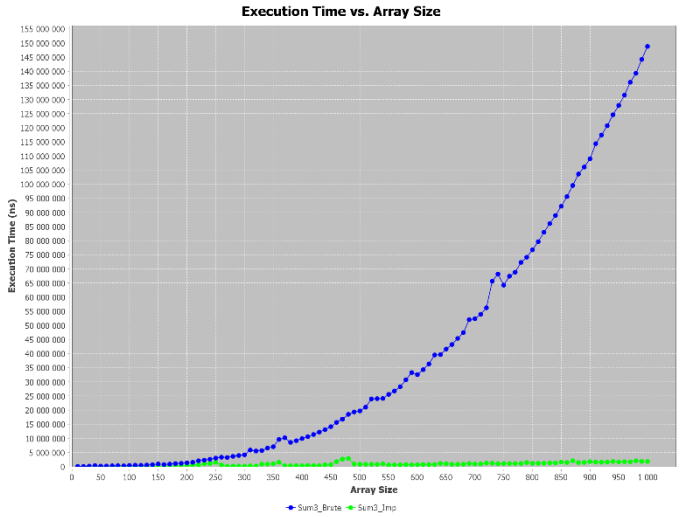


Figure 2 - Comparison of 3Sum implementations (inc. step size 10)

int startSize = 10; *// smallest array size*

int endSize = 1000; *// largest array size*

int increment = 10; *// Increment step*

int runTimes = 5; *// Multiple runs per size*

int targetSum = 0; *// sum to find*

Above are two graphs displaying the 3Sum_Brute and 3Sum_Imp algorithms, and how they grow based on array size. The difference between the two graphs is the number of runs per size, and therefore the accuracy of the measurements, and the increment steps. A smaller increment means a more accurate graph which follows the exponential growth better, whereas the larger increment provides fewer points to plot, but takes less resources.

The two-point implementation of the Sum3_Imp makes the graph appear almost linear in comparison to the Sum3_Brute, but this is purely due to the difference in magnitude when plotting them. Below, there are graphs which focus on the two implementations of the two algorithms separately, and there it is easier to see that the Sum3_Imp is close to quadratic.

Logarithmic Linear Regression

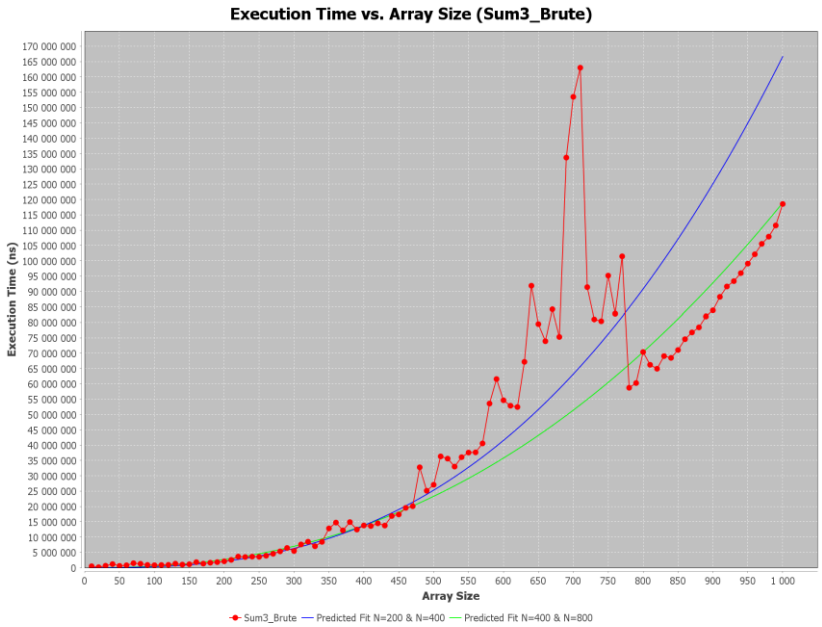
In an attempt to manually find the time estimates for both implementations of the 3Sum-algorithms, the logarithmic linear regression, also known as the log-log regression formula, was used. The

formula is as follows: $log_2\big(time(x)\big) = b \times log_2(x) + c$, where $b = \frac{\Delta \log_2(time(x))}{\Delta \log_2(x)}$ and $c = \log_2\big(time(x)\big) - b \times \log_2(x)$ and *x* representing the size of the array. To be able to apply this to the two 3Sum-algortihms, the comparisons used above were re-written to focus on one at a time. Instead of running through all the array sizes and plotting straight away, it is done in steps where the value pairs of the array sizes and their corresponding times are found, *b* and *c* values are found based on the equations above, and the fitted line is then graphed so that $time(x) = \frac{b \times log_2(x) + c}{log_2(x)}$. The formula can also be re-written as a power law where $ax^b = 2^c \times x^b$.

Logarithmic Linear Regression – Sum3_Brute

The Sum3_Brute has, as mentioned previously, an upper boundary of $O(N^3)$, meaning that we would expect the value of *b* in $ax^b = 2^c \times x^b$ to be less than or equal to 3.

1st Run – 5 Run Times



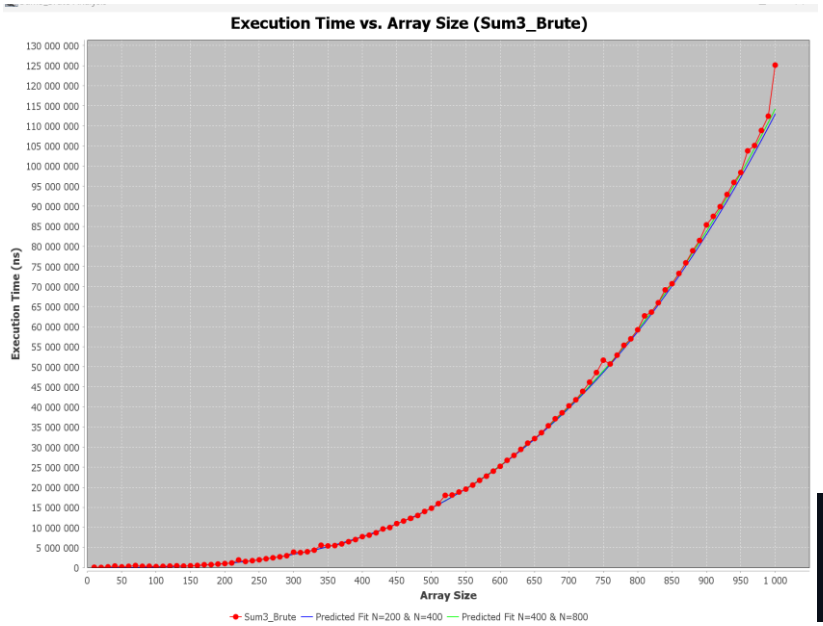

```
For N=200 and N=400:
The b-value is: 2.719530927517672
The c-value is: 0.20976770199615302

For N=400 and N=800:
The b-value is: 2.351447853541835
The c-value is: 3.3914248593334975
```

As can be seen from the first graph of the brute run, it was not very successful. Many test runs were executed and depending on where the outliers were concentrated, the calculations of *b* and *c* were far from consistent. Therefore, the choice was made to calculate two different lines of best fit: one based on the two points of 200 and 400, with the other being based on the two points of 400 and 800. As can be seen from the small image representing the *b* and *c*-values, both had *b*-values below 3, which was positive. However, there were too many outliers between approximately 600 and 800 to get an accurate representation. Therefore, the test was run and recorded again.

Tea Elming
1DV018
Sept 2024

2nd Run – 15 Run Times




```
For N=200 and N=400:
The b-value is: 2.9285145945194175
The c-value is: -2.434372106865599

For N=400 and N=800:
The b-value is: 2.940591438044329
The c-value is: -2.538762605521349
```

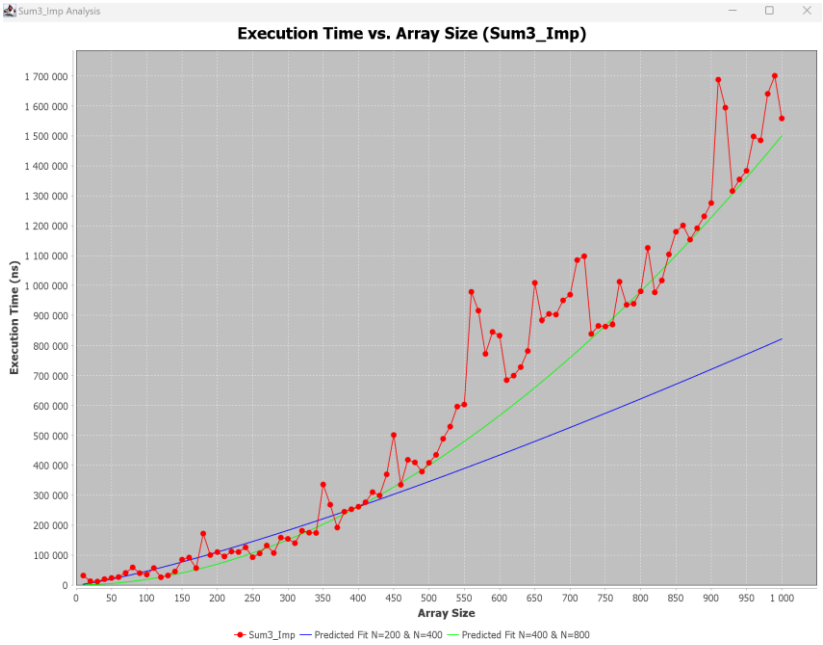
The second attempt altered the amount of run times so that the averages were calculated based on 15 runs rather than 5, as they were in the first run. It may have been that, or other environmental factors that occur in a less-than-ideal testing environment such as a laptop with many other things running at the same time, that produced this quite accurate execution time estimate. As can be seen from the small list of *b* and *c*-values, the *b*-value was relatively close to 3, which aligns with what we would expect. In addition to that, the calculations for the two different combinations of *N*-values yielded similar results for both the *b* and *c*-values, meaning it was a fairly accurate estimate in the first place.

Logarithmic Linear Regression – Sum3_Imp

Following the same logic as when we expected Sum3_Brute to have an upper boundary of $O(N^3)$, we expect Sum3_Imp to have and upper boundary of $O(N^2)$. We therefore expect our *b*-value in the equation $ax^b = 2^c \times x^b$ to be less than or equal to 2.

Tea Elming
1DV018
Sept 2024

1st Run – 100 Run Times

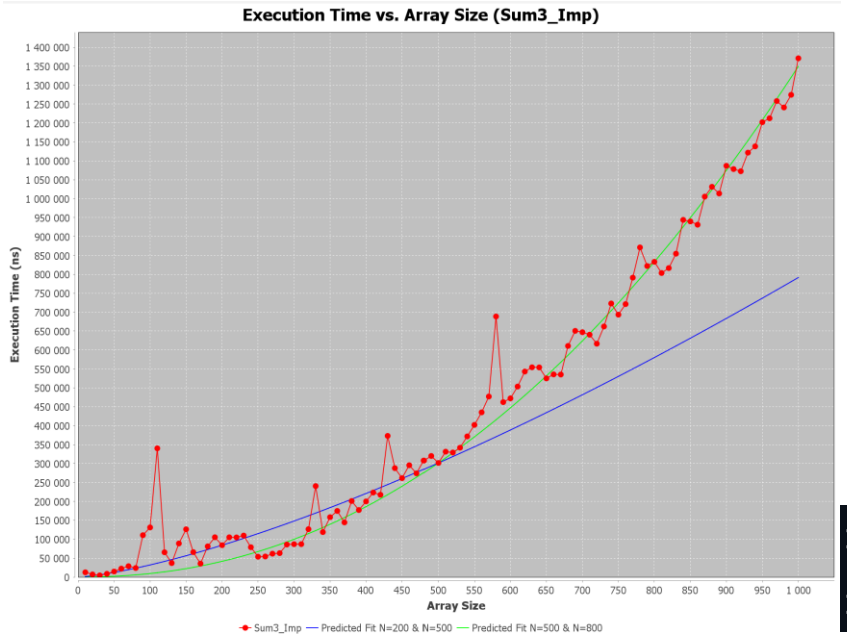



```
For N=200 and N=400:
The b-value is: 1.2502059041363356
The c-value is: 7.19019890697537

For N=400 and N=800:
The b-value is: 1.9062620401006534
The c-value is: 1.519344015280513
```

Learning from the experience of the incosistencies that occurred when measuring too few runs, and thereby creating an unreliable average, all sizes were initally tested 100 times. However, despite the more rigourous testing, our results were not ideal when using the first two points: N=200 and N=400. The line calculated based on N=400 and N=800 was relatively successful, especially with a *b*-value of roughly 1.9, however there was a high number of outliers which caused inconsistencies. The failure of the calculation based on N=200 and N=400 is obvious past an N of 500, as it is significantly below the plotted points. The second set of N-values, represented by the green line, were not only higher values, but also further apart (400 apart vs. 200 apart for the first set of points). Therefore, the logical next step was to test with N-values that were the same distance apart.

2nd Run – Changed N-points, 100 Run Times



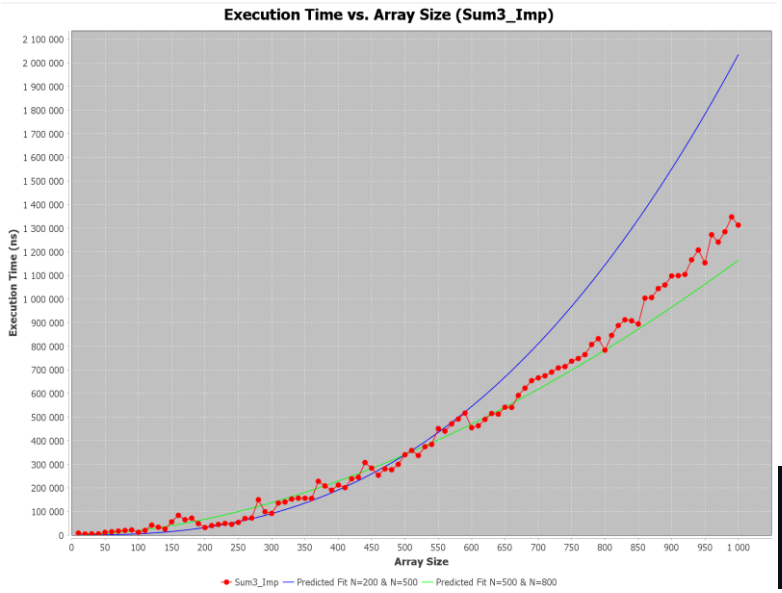

```
For N=200 and N=500:
The b-value is: 1.3926174955375554
The c-value is: 5.715742076096085

For N=500 and N=800:
The b-value is: 2.1623755128525947
The c-value is: -1.1857440516965987
```

Tea Elming
1DV018
Sept 2024

The second test was run using the points N=200, N=500, and N=800, making them more evenly spaced and the same distance apart. This yielded a slightly more successful outcome, with the blue line having a gradient of 1.39 rather than 1.25, but still not an accurate representation. On the other hand, the green line had a *b*-value of 2.16, which is above what we would expect from our formula. In a last attempt to resolve this, the next run would double the amount of run times for each execution.

3rd Run – 200 Run Times

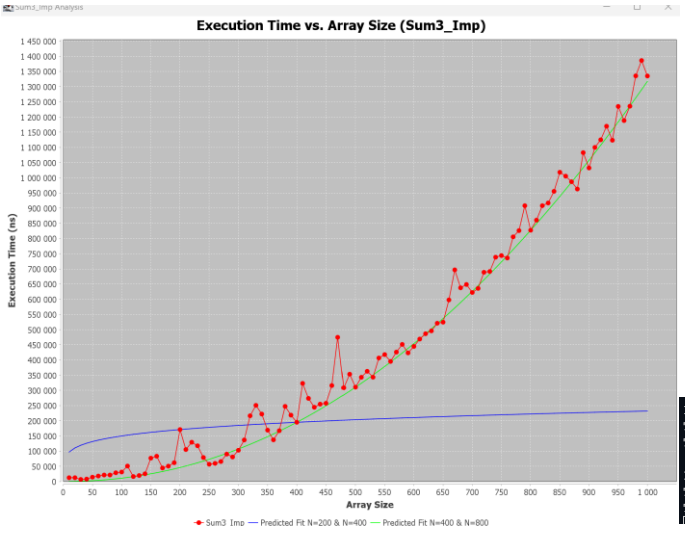



```
For N=200 and N=500:
The b-value is: 2.57944782500494
The c-value is: -4.750062322977421

For N=500 and N=800:
The b-value is: 1.7734984097247202
The c-value is: 2.4759062787745716
```

The blue line based on the points N=200 and N=500 overestimated the amount of time it would take significantly, with the exponent reaching 2.58. however, the green line calculated based on N=500 and N=800 is, although a slight underestimate, relatively close to the plotted points. With that said, neither of the calculations are ideal.

It is interesting that the 3Sum_Imp proved that much more challenging to estimate, and it appears as if the smaller the array, the less reliable it is. One run, following the set up of the first run, yielded this graph:




```
For N=200 and N=400:
The b-value is: 0.18964017097482594
The c-value is: 15.932856079649113

For N=400 and N=800:
The b-value is: 2.085683101490634
The c-value is: -0.4562663413685648
```

Tea Elming
1DV018
Sept 2024

The blue line based on the earlier plots is clearly very far off the mark, whereas the green line based on the later plots was almost perfect. It goes to show that it is beneficial to test several different points if wanting to achieve an accurate execution time estimate.