Assignment 1: Three-Sum & Four-Sum Problem

Christian Vilen

2022-09-20

1 Introduction

The Three-Sum and Four-Sum problem is a classic computer science challenge in the area of algorithm design and implementation. The objective of the Three-Sum challenge is to determine if three distinctive elements exist within a data structure that sum up to zero when added together. This is extended to one more distinctive element in the Four-Sum Challenge. The Three-Sum problem satisfies the following mathematical properties:

$$i, j, k\,(i \neq j,\ i \neq k,\ j \neq k),\ x_i + x_j + x_k = 0$$

This academic paper aims to report on three different implementations that solve the Three-Sum and Four-Sum problem. Accordingly, delving and comparing algorithms with cubic, quadratic and hash map solutions of complexity. The comparison should highlight their respective performance characteristics with various tables and figures. Thus, the different solutions will be analysed and compared in terms of their algorithmic intricacies

2 Implementation

The implementation of the three different solutions to the Three-Sum and Four-Sum problem can be found in the code repository here: Git Repository.

2.1 Hardware and Software specification

All of the experiments have been executed on the following equipment:

- Computer: MacBook Pro, 14-inch 2021.

- Hardware: Apple M1 Pro, 16gb ram.
- OS Version: Mac OS X Ventura 13.5.1
- Java: 17.0.5 2022-10-18 LTS
- JUnit: 4.13.2
- Python: 3.11.5
- Gradle: 7.6
- Kotlin: 1.7.10

2.2 Correctness Tests

The implementation of the four different Three-Sum and Four-Sum algorithms has been tested with various JUnit tests to verify their correctness. This includes multiple variable testcases of 0, 1 and more triplets summing to null or not.
It is also important to note, that all of the tests and experiments have been executed while the computer has been connected to a wall power outlet. This is in order to reduce the frequency of power availability from the battery which can lead to varying benchmark results.

2.3 Input data

The data generated for the tests of the different solutions of the Three-Sum and Four-Sum problem has used the following mathematical expression:

$$n_i = 30 * 1.41^i$$

This is in order to produce an approximate increasing factor of $\sqrt{2}$ for every element in the input data. Hereby, providing a sufficient incremental scale and range of data.

2.3.1 Table example

An example of a varying range of data and the average time for the respective cubic Three-Sum solution can be seen in Table 1.

n	Average (s)	Standard deviation (s)
30	0.048482	0.001032
42	0.048283	0.000840
59	0.049192	0.001383
84	0.049817	0.001124
118	0.055000	0.006590
167	0.057875	0.006375
235	0.054528	0.000754
332	0.055737	0.000740
468	0.060067	0.001440
660	0.067821	0.003220
931	0.092099	0.004582
1313	0.140196	0.000329
1852	0.272598	0.001393
2611	0.633316	0.002264
3682	1.623077	0.003325
5192	4.358222	0.014700
7321	11.905279	0.014488

Table 1: Running time of cubic solution to Three-Sum

3 Experiments

3.1 Three-Sum experiments

The four solutions to the Three-Sum problem have an increasing average execution time as the input size increases. This is the expected behaviour for each of the respective solutions which can be seen in Figure 1.
The Cubic algorithm has the highest execution time compared to the three other algorithms in Figure 1.
The HashMap and New HashMap algorithms have a similar trend in terms of execution time in relation to the input values.
The Quadratic algorithm performs best across all of the other algorithms also making it handle a bigger input of values.

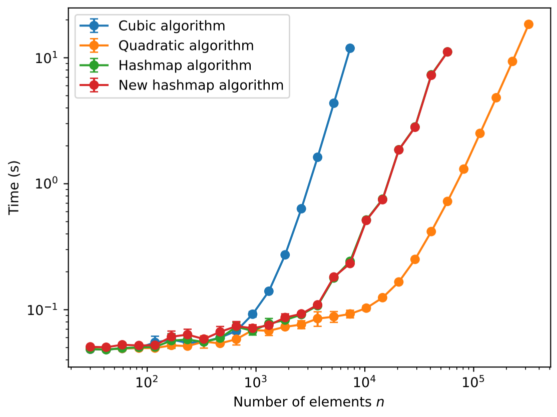

Figure 1: Three-Sum running time visualised

The first ten initial test values are quite similar for all of the four solutions. This can be seen in Figure 2. Thus, highlighting the standard deviation being relatively low for the first ten input values of among all algorithms. Although, afterwards the first ten values the four algorithms take completely different shapes as figure 1 showed.

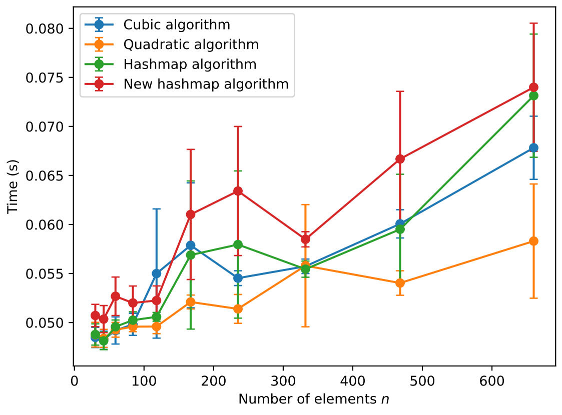

Figure 2: Start of Three-Sum visualised

The standard deviation for each solution to the Three-Sum Problem is notably also increasing as the number of elements increases which can be seen in Table 2. This can be due to the longer time it takes for the solutions to run the algorithm to its fullest, hence, increasing the amount of outside variables introducing noise to the benchmarks.

	Quadratic	HashMap	New HashMap	Cubic
n	SD (seconds)	SD (seconds)	SD (seconds)	SD (seconds)
30	0.00126	0.001085	0.001136	0.001032
42	0.00092	0.00069	0.001963	0.00084
59	0.000753	0.000852	0.001726	0.001383
84	0.000508	0.000446	0.001488	0.001124
118	0.000727	0.007554	0.006633	0.00659
167	0.000706	0.007506	0.006581	0.006375
235	0.001471	0.000824	0.000775	0.000754
332	0.00623	0.005606	0.006894	0.00074
468	0.00125	0.006281	0.006533	0.00144
660	0.005825	0.004998	0.004717	0.00322
931	0.001386	0.007723	0.005378	0.004582
1313	0.005599	0.006977	0.006003	0.000329
1852	0.001416	0.001412	0.001273	0.001393
2611	0.005506	0.001033	0.00199	0.002264
3682	0.011227	0.002575	0.00269	0.003325
5192	0.008667	0.003497	0.003158	0.0147
7321	0.006417	0.005337	0.002877	0.014488
10323	0.0004608	0.0003674	0.0002755	-
14556	0.0003242	0.0004816	0.0001871	-
20525	0.0003217	0.0006928	0.0007886	-
28940	0.0004615	0.0116311	0.0024508	-
40805	0.0006638	0.0021751	0.0048378	-
57536	0.007881	-	-	-
81126	0.002676	-	-	-
114387	0.002589	-	-	-
161286	0.012718	-	-	-
227414	0.018796	-	-	-
320654	0.008395	-	-	-

Table 2: The standard deviation denoted SD in seconds of all Three-Sum solutions.

3.2 Four-Sum experiments

The three solutions to the Four-Sum problem follow the same trends as for the Three-Sum problem. In which, the cubic solution has the highest execution time compared to the other two algorithms in Figure 3. The HashMap solution trends in the middle of the three algorithms. The Quadratic algorithm performs the best of them all in the Four-Sum problem and likewise handles the largest input as seen in Figure 3.

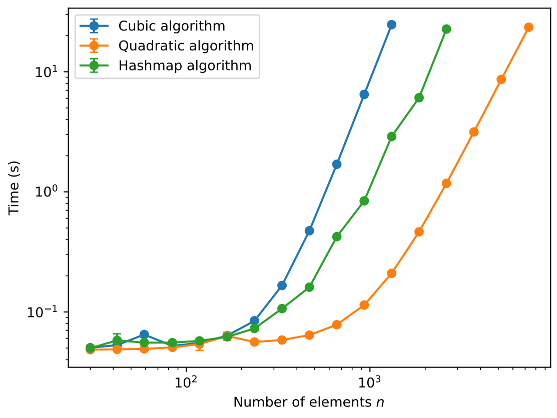

Figure 3: Four-Sum running time visualised

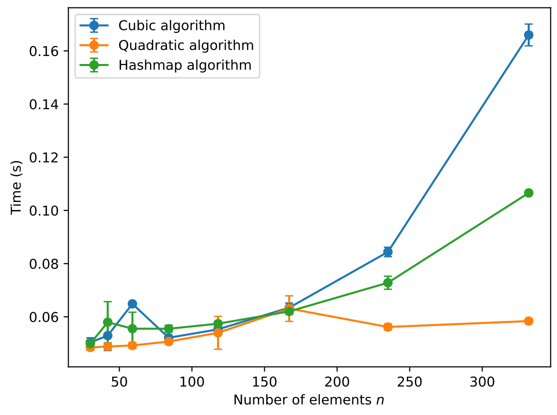

Figure 4: Four-Sum running time visualised

The first six points of the three algorithms to the Four-Sum problem have on average the same computation time as seen in Figure 4. Hereafter, they take different trends according to their algorithmic problem-solving techniques. This suggests that the respective algorithms can solve the same challenge of approximately 180 elements at the same time.
Notably, the cubic algorithm has a relatively high start-up time compared to the other algorithms as seen in the third point in Figure 4 where it jumps on the scale.

	Quadratic	HashMap	Cubic
n	SD (seconds)	SD (seconds)	SD (seconds)
30	0.001076	0.000599	0.001758
42	0.000563	0.007741	0.00561
59	0.001092	0.006202	0.000734
84	0.000967	0.001354	0.000872
118	0.006186	0.00085	0.001002
167	0.004805	0.001098	0.001755
235	0.001132	0.00246	0.001765
332	0.000682	0.000787	0.004095
468	0.000965	0.001022	0.004256
660	0.000704	0.00177	0.001136
931	0.000962	0.004809	0.027755
1313	0.000678	0.00264	0.061727
1852	0.000952	0.065454	-
2611	0.003646	0.233811	-
3682	0.008666	-	-
5192	0.044877	-	-
7321	0.010864	-	-

Table 3: The standard deviation denoted SD in seconds of all Four-Sum solutions.

The cubic algorithm has a low standard deviation, especially for a lower number of elements (for example n = 30). However, a few outliers with large standard deviations exist, particularly for n = 931 and 1313. These outliers show that there are somewhere the execution duration varies greatly. When compared to the cubic algorithm, the Hashmap algorithm has somewhat higher standard deviations. Across a range of n values, the quadratic algorithm keeps standard deviations low and constant. This indicates that the quadratic algorithm's execution times are often consistent and predictable.