Foursum Report

Domenico Villani, Peter Zander Havgaard & Thomas Keralla Høpfner-Dahl

March 4, 2019

Exhaustive search

Our program `Simple.java` solve the Four-sum problem using four nested loops. The index variables i, j, k, l run respectively from 0, 1, 2 and 3 to N. Thus, we can bound the number of array accesses by $\sim N^4 \log N$.

Experiments

The following table summarises the empirical performance data on the input files in the `input` directory, created using Weed.java. We ran each input size 5 times, and report the minimum, maximum, and average running time. The tests were run on a machine with an Intel 3570k CPU running Linux. The unix `"time"` command was used to measure the elapsed time in seconds.

	Simple.java		
N	min	max	avg.
30	0.06	0.08	0.064
50	0.07	0.08	0.072
100	0.08	0.09	0.082
200	0.13	0.14	0.138
400	0.86	0.87	0.868
800	3.86	9.19	8.11
1600	107.88	108.32	108.15
3200	178.46	178.92	178.79

Exhaustive Search Python

We have also implemented the same algorithm in python as `simple.py`. The python implementation was run using python2 and only values 30 through 400 were computed in the interest of time. The table below show the recorded running times from this implementation.

	simple.py		
N	min	max	avg.
30	0.05	0.06	0.052
50	0.08	0.08	0.08
100	1.23	1.75	1.472
200	19.46	22.31	20.588
400	218.41	294.35	244.86

Improved Search

Using the binary search-based idea sketeched in [SW, 1.4] for the Three-sum problem, we can improve our running time to $\sim N^3 \log N$. We implemented this algorithm in the [`Faster.java`] document. To do so we created a method based on the same algorithm of the binary search that was able to handle long data types.

The following table reports our maximum, minimum and average running times on the test inputs.

	Faster.java		
N	min	max	avg.
30	0.06	0.07	0.068
50	0.07	0.08	0.078
100	0.07	0.08	0.076
200	0.09	0.1	0.096
400	0.22	0.23	0.22
800	1.38	1.54	1.458
1600	10.84	13.35	11.459
3200	90.5	94.21	91.429

Quadratic Search

We have implemented a faster algorithm called FasterQuad.java which is able to improve our running time to $\sim N^2 \log N$. It is based on a inner class called Sum which represents the sum of two numbers in our array. The inner class also keeps track of the elements which are summed using a method (noCommon) to be sure that there is no repetition in the count. Then the algorithm uses two nested for loops to compare the values.

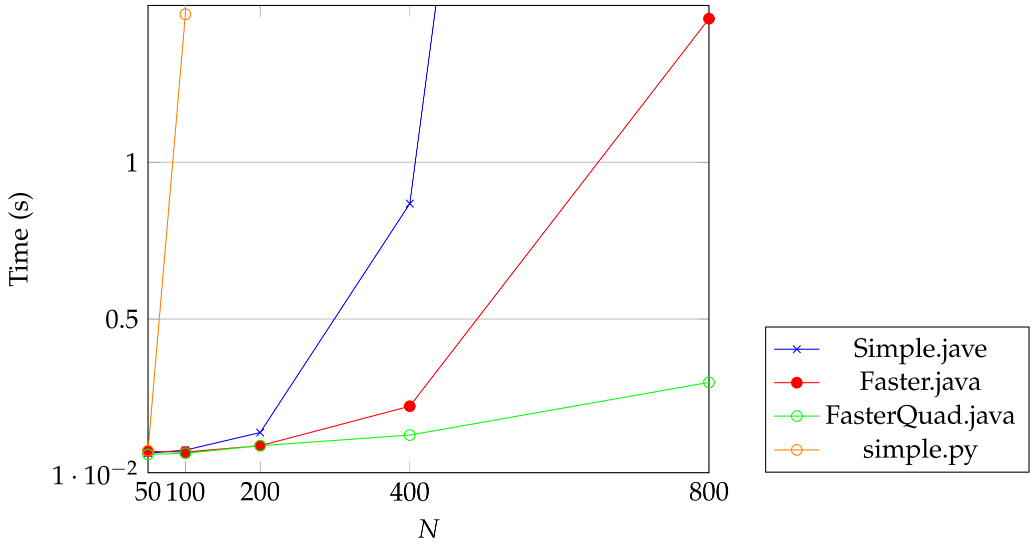
The following table reports our maximum, minimum and average running times on the test inputs.

	FasterQuad.java		
N	min	max	avg.
30	0.06	0.08	0.07
50	0.06	0.08	0.068
100	0.07	0.08	0.072
200	0.09	0.1	0.096
400	0.13	0.13	0.13
800	0.28	0.32	0.298
1600	0.64	0.68	0.666
3200	2.95	3.02	2.983

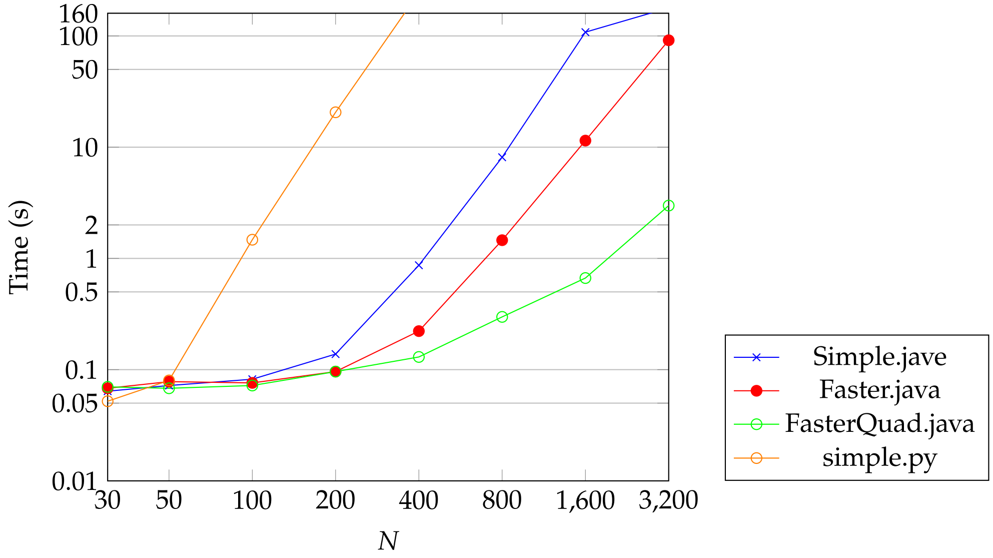
Graphs

Below are two graphs comapring the running times of the three implemented algorithms. The first is a strandard plat and second a log plot.

Average running time of 5 runs



Log table of average running time of 5 runs


Time (s)
N
Simple.jave
Faster.java
FasterQuad.java
simple.py