Report: Foursum

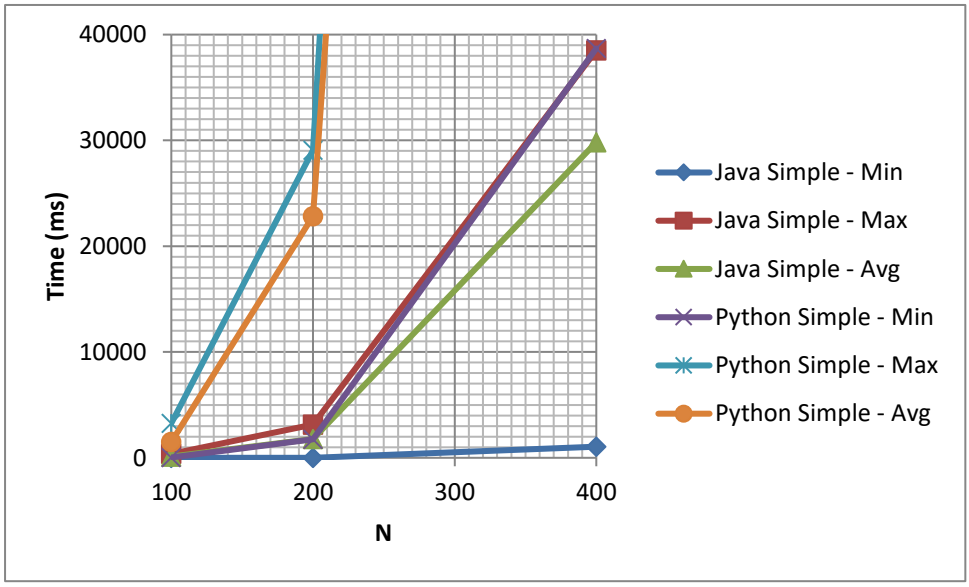
By Sergiy Isakov, Rasmus Lehmann & Andreas Tønder

Exhaustive search - N^4

Our simple search solutions use four for-loops, and validate the indexes to 0. The first index *i* runs from *0* to *N - 3*, next index *j* runs from *i + 1* to *N - 2* , third index *k* runs from *j + 1* to *N – 1* and the last one *l* runs from *k + 1* to *N*. This approach let us avoid comparing same elements. Thus, we can bind the number of array accesses by N^4.

The runtime of simple implementations were tested using all of the given .txt files where N <= 400. The data sets of N <took an incredible amount of time to run with this sub-par implementation, and we deemed that the point was made, using only the other data sets.

		Java Simple			Python Simple		
N	T	Min	Max	Avg	Min	Max	Avg
100	500	0	376	121	31	3.268	1.496
200	40	0	3.138	1.770	1.757	29.075	22.849
400	25	1.050	38.509	29.826	38.693	478.323	397.108



Faster search - N^3

Our faster implementation starts by sorting the array. It then uses two for-loops to loop through the first two variables, as above. The two inner for-loops are replaced by a single while-loop, where two cursors are placed at each end of the array. Based on whether the sum of the four values is above or below the target value of 0, the smallest variable is either incremented, thus increasing the overall value of the sum, or the largest value is decremented, decreasing the overall value of the sum.

The faster implementation was tested using all of the given .txt files. Average times were calculated based on run-times from the four-five files, corresponding to each N-value, and runtimes are stated in ms. With the implementations on the faster solution, runtime is improved to approximately N^3.

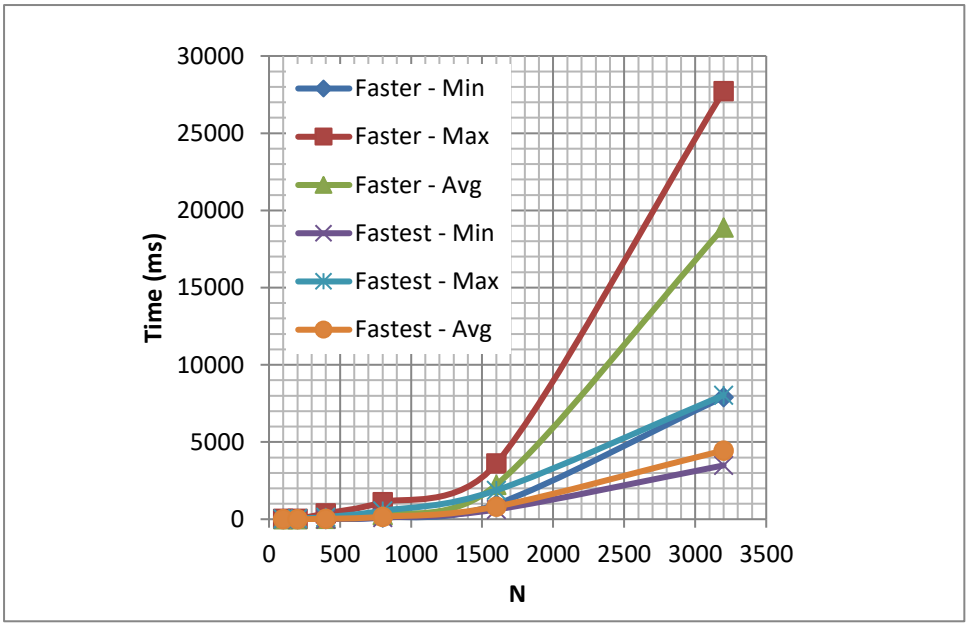
Fastest search – N^2LogN

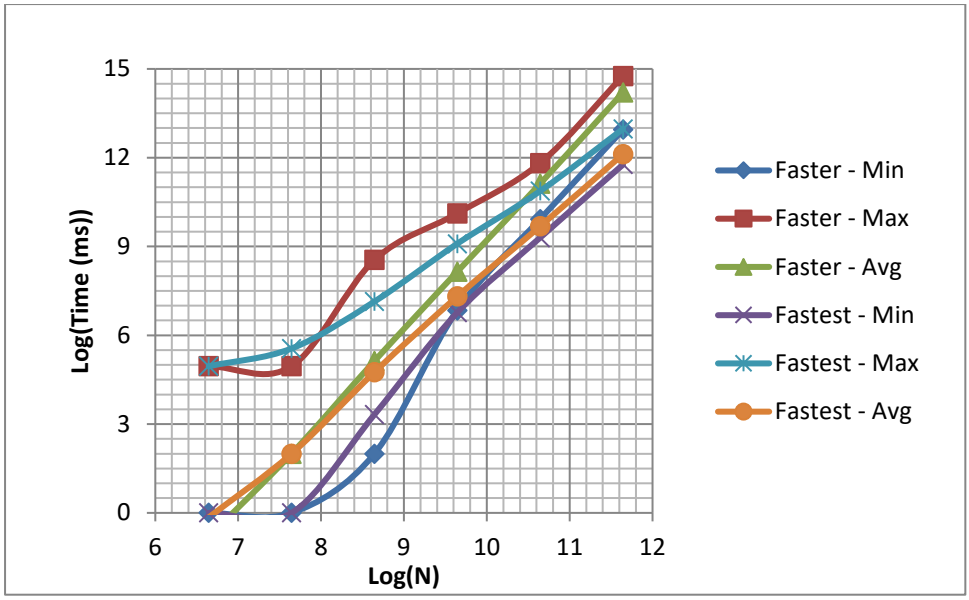
The fastest solution uses an auxiliary array aux[] that stores objects from a class we call "Pairsum". This class stores information about all the possible pairs from the input array vals[], and has the size of N*(N-1)/2. Storing the information is vital for the next step as the array will now be sorted based on sums using an overridden compare method. After having sorted the array it is now time to find two pairs in aux[] with the sum equal to 0 using the same approach as the "faster search". In order to check whether or not these two elements of aux[] represent four distinct elements of input array A[] the method "noCommon()" is called.

Finding all the possible pairs will take $O(N^2)$ time, sorting the auxiliary array with all the possible pairs using, for instance, heap-sort will take $O(N^2logN)$ time. Going through the auxiliary array to find the pairs will also take $O(n^2)$ time. Overall runtime is thereby improved to approximately N^2logN.

Comparison of min, max and avg times for calculation of foursum on data sets of size N, using the faster and fastest algorithms, shown on a regular as well as log-log plot:

		Faster(N^3)			Fastest (N^2LogN)		
N	T	Min	Max	Avg	Min	Max	Avg
100	5000	0	31	0,57	0	31	0,90
200	2800	0	31	4	0	47	4
400	2500	4	374	35	10	141	27
800	1000	115	1.111	284	109	547	159
1600	500	973	3.613	2.229	630	1.876	822
3200	100	7.911	27.725	18.895	3.486	8.038	4.455




15
12
9
6
3
0
6
7
8
9
10
11
12
Log(Time (ms))
Log(N)
Faster - Min
Faster - Max
Faster - Avg
Fastest - Min
Fastest - Max
Fastest - Avg