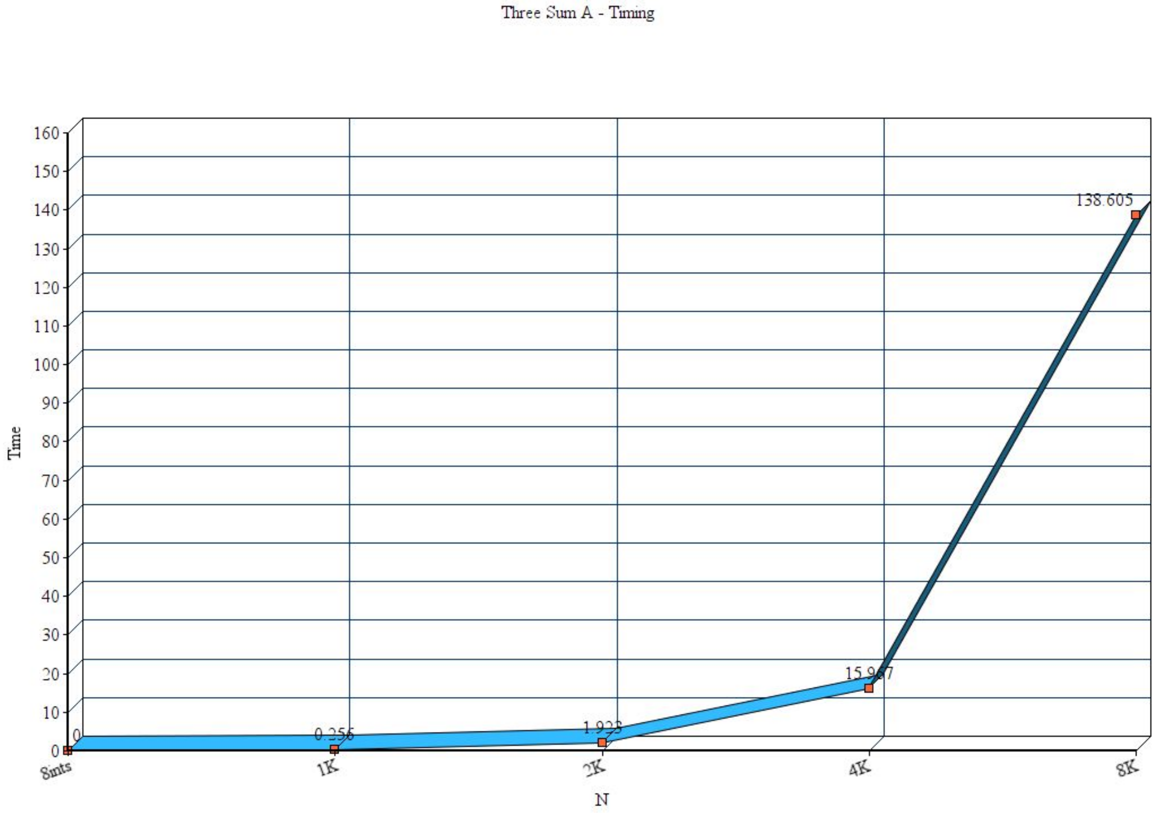
# Practical 1 - Empirical Analysis

## ThreeSumA - Timing experiments

| Algorithm | Input | Time | Number of Triples |
|---|---|---|---|
| **ThreeSumA** | 8ints.txt | 0.0 | 4 |
| | 1Kints.txt | 0.256 | 70 |
| | 2Kints.txt | 1.923 | 528 |
| | 4Kints.txt | 15.967 | 4039 |
| | 8Kints.txt | 138.605 | 32074 |

## ThreeSumA Timing Graph

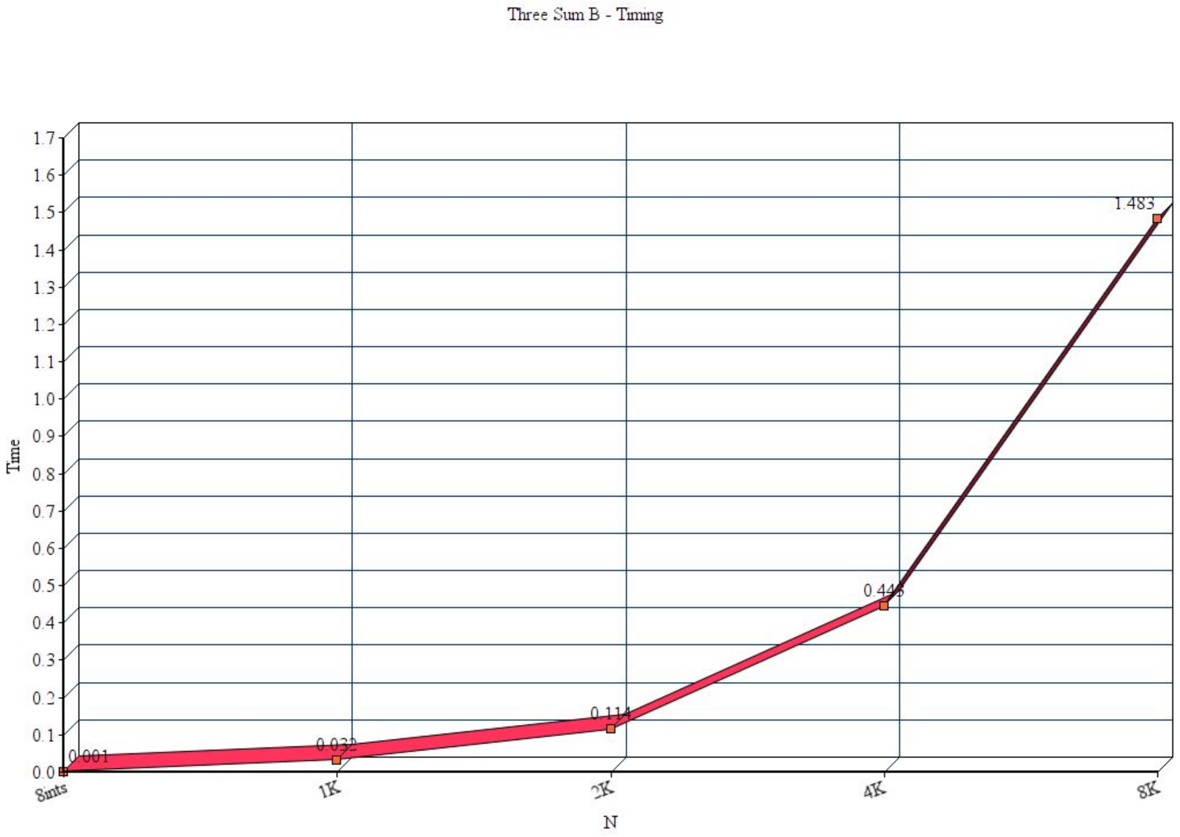


## ThreeSumB - Timing Experiments

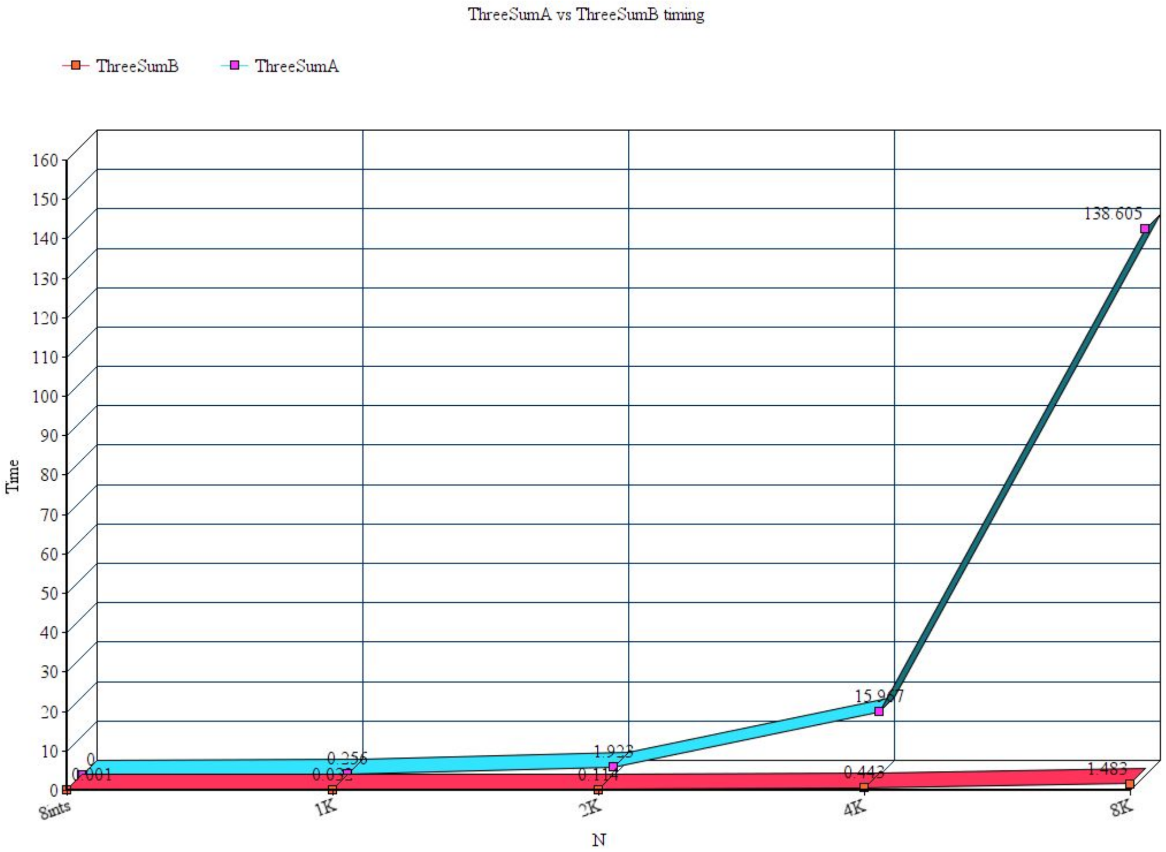
| Algorithm | Input | Time | Number of Triples |
|---|---|---|---|
| **ThreeSumA** | 8ints.txt | 0.001 | 4 |
| | 1Kints.txt | 0.032 | 70 |
| | 2Kints.txt | 0.114 | 528 |

| | | | |
|---|---|---|---|
| | 4Kints.txt | 0.443 | 4039 |
| | 8Kints.txt | 1.483 | 32074 |

### ThreeSumB Timing Graph



### ThreeSumA versus ThreeSum B - timing graph



**Questions:**

**Which algorithm performs better (i.e. take less time in relation to the size of the input)?**
ThreeSumB performs better, with the exception of the 8ints input.

**Why do you think this is the case?**
In ThreeSumA, the algorithm uses three nested for loops in order to find the triplets.
As we can see in the above timing graph, while the solution could work for smaller inputs (see 8ints), it is not maintainable for larger inputs.

In ThreeSumB, the triplets are found using a binary search. It is interesting to see that this algorithm takes longer for the smallest N (8 ints). This is because the array needs to be sorted in order to be able to perform a binary search. For that reason, it makes sense to use the algorithm ThreeSumB for larger inputs, where the sorting is justified. The binary search allows for far quicker output and it is scalable, as opposed to the ThreeSumA algorithm.