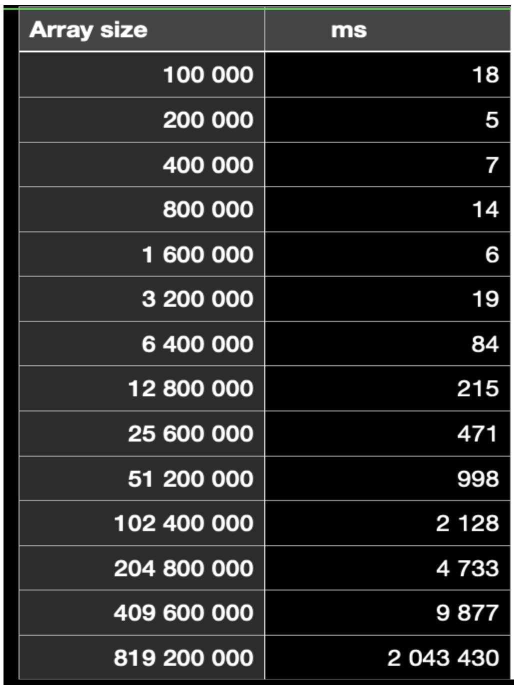
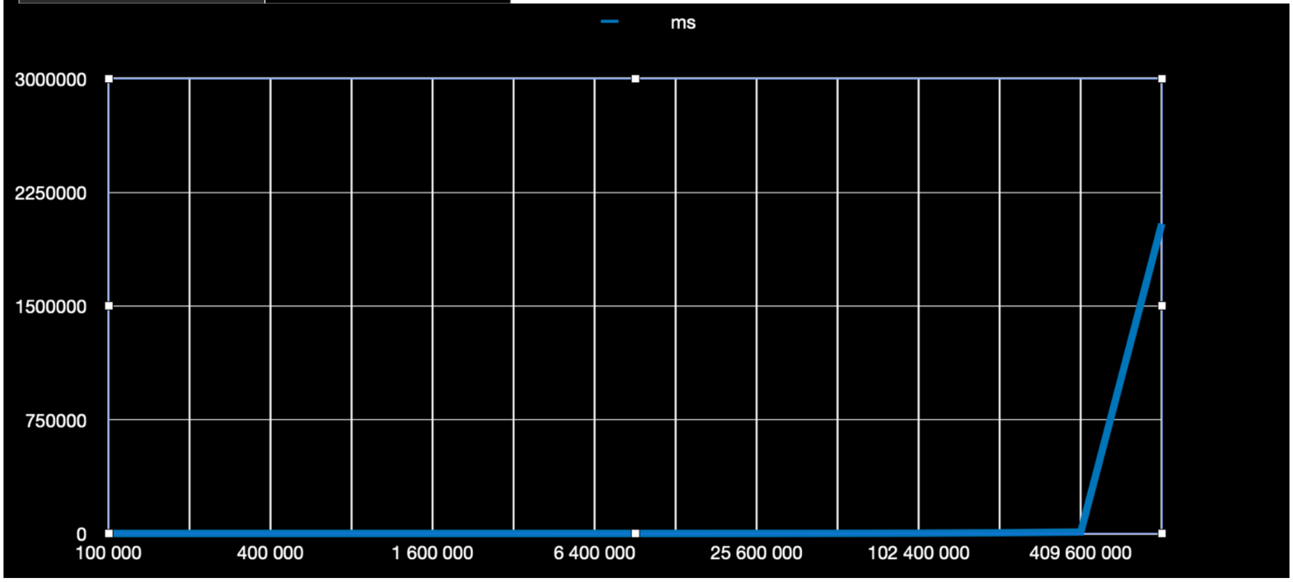
Test done on Macbook pro: OS: Mac OS X 12.2.1 - x86_64 - processors: 8 - JVM heap size: 4 311 744 512



Array size	ms
100 000	18
200 000	5
400 000	7
800 000	14
1 600 000	6
3 200 000	19
6 400 000	84
12 800 000	215
25 600 000	471
51 200 000	998
102 400 000	2 128
204 800 000	4 733
409 600 000	9 877
819 200 000	2 043 430



At First the time complexity seems to be O(n) because when the number of arrays is doubled the time is also doubled. But something happens after 409 600 000 of array size the time skyrockets to 200 times bigger and I have kept the program running for overnight and it is still working with the bigger numbers. The time complexity of the code should be O(n) because it is just using two not nested loops. And it is until the big numbers. When looking at the graph Complexity of O(n) should be more linear, mine is linear until the big numbers. At first, I thought that it the bigger numbers would need more memory because they are bigger so that would explain it, but after quick research I found that that can't be the reason. don't know if there something wrong with the code or did I just misunderstand this time complexity. But in the excel tutorial video the teachers time complexity wasn't like mine. Reason for this might also be just that my laptop is slow.

Int range of the values could also be limited by for example if you have very old system which runs on 16-bits OS, there would be much less integers because java int is 32-bit.

To get more number than java int you can use java long or biginteger or you can use Strings if you don’t need the numbers for math.