Comparing two algorithms from different growth classes

We have two algorithms (ThreeSumA and ThreeSumB) that count the number of triples in a file of *N* integers that sums to 0 (ignoring integer overflow).

1. Estimate the Big O for each of these algorithms by looking at their code
2. Add simple java code to time the running time of both algorithms
3. Use the input files provided to run timing tests on both algorithms for each file, noting the results
4. Graph the results

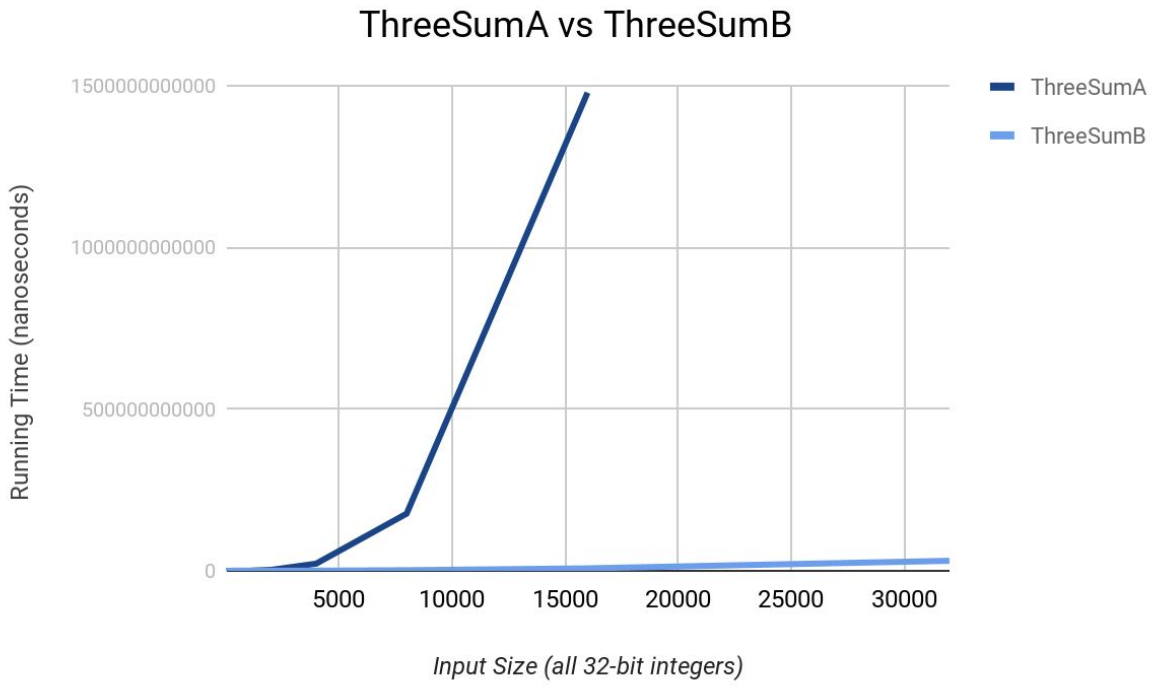
1.

ThreeSumA: I estimate this program is approximately O(n^3) due to the three-level nested for loop in the count() method

ThreeSumB: I estimate this program is approximately O(n^2 * logn) due to the two-level nested for loop in the count() method which contains a BinarySearch() call, which is approximately O(logn), as well as a call to Arrays.sort(), which uses the quicksort algorithm behind the scenes, which is O(n logn).
Overall complexity: O(n logn) + O(n^2) * O(logn)
== O((n^2 + n) * logn)
== O(n^2 * logn)

2 & 3.

Input Size	ThreeSumA running time	ThreeSumB running time
8	5979	26759373
1,000	305091238	36524437
2,000	2784938942	72150029
4,000	22110883116	469190023
8,000	176715436368	1837424852
16,000	1478384197112	7603106323
32,000	DNF (est. time ~4 hours)	30927793277



As we can see from the above graph, the complexity of ThreeSumA is much less favourable than that of ThreeSumB. So much so, that I had to forego testing the running time for the 32Kints.txt test file, as the estimated running time would have been ~4 hours. For comparison, ThreeSumB took 30 seconds to perform the test for 32K ints.