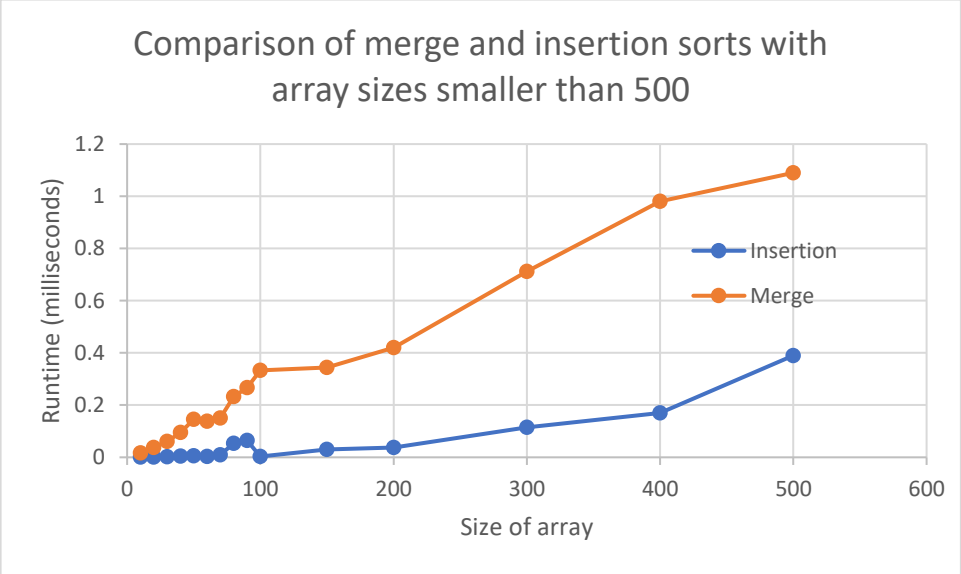
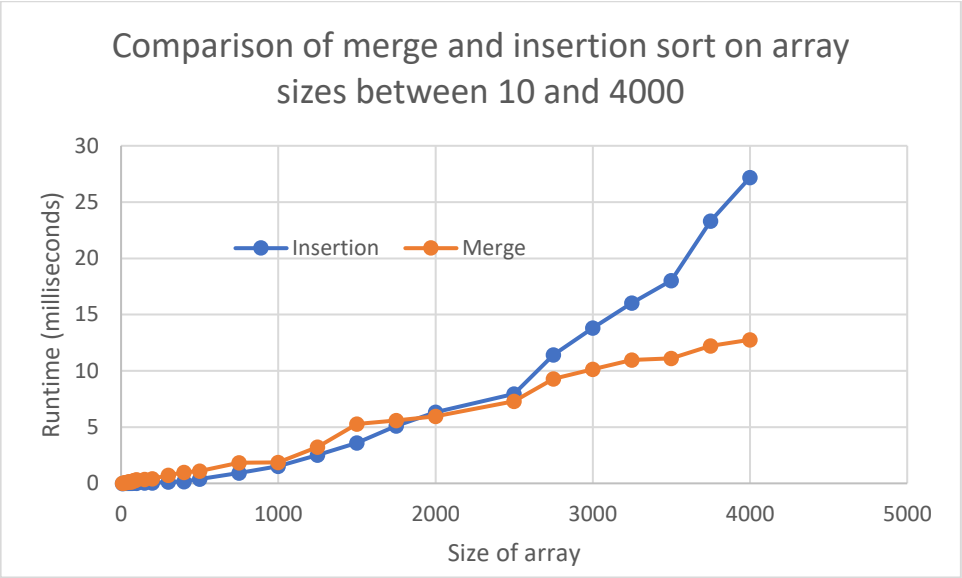
 Hypothesis: I believe that for large values of n, the merge sort will be much faster than the insertion sort. From the theoretical point of view, the expected runtime for merge sort is $O(n * \log(n))$, while for insertion sort it's $O(n^2)$. However, these comparisons have one problem. They hide constants and functions that grow slower than the main one behind the O symbol. So, under certain conditions, the insertion sort with $O(n^2)$ may end up running faster than the merge sort with $O(n * \log(n))$. Another argument that can be used to support insertion sort is that it sorts values in place, unlike merge sort, which requires additional space allocations and recursion that take additional time. My prognosis is that for $n < 100$, the insertion sort will be faster than the merge sort.

 Methods: first, I implemented sorting algorithms insertionSort() and mergeSort() that take reference to a vector as a parameter. Then, to generate a vector with random values I made function getRandomVector() that accepts the size of a desirable vector as a parameter. It uses a uniform distribution between 0 and 1 billion (in C++ implemented as std::uniform_int_distribution) to assign random values. Finally, for benchmarking, I created testSorts() function with two parameters – the size of the vector to test and the number of times the experiment should be repeated. The function itself creates a random vector and a deep copy of it, then sequentially runs insertion sort and merge sort and measures how much time it took to run the algorithm. In the end, it divides the results by the number of repetitions and outputs the average running time to the console. For compilation I used the standard GCC compiler with no optimizations or flags. The code available at this link:

 Results:



The experiment was conducted using array sizes between 10 and 4000. On the small array sizes below 500, the insertion sort ended up sorting elements much faster than the merge sort, which can be clearly seen on the graph below. However, if we look at the full graph, we can see that for sizes of arrays between 1000 and 2500, it is difficult to tell which algorithm is better. Insertion sort begins to lose its effectiveness due to $O(n^2)$ complexity. If we investigate further, it becomes clear that the merge sort becomes much faster than the insertion sort once the size of the array approaches 3000 values.



 Discussion: The conducted experiment gives the opportunity to analyze algorithms with complexities $O(nlogn)$ and $O(n^2)$. While it may not be obvious, an algorithm with $O(n^2)$ can work much faster than $O(nlogn)$ for smaller values of n. Personally, I was surprised that insertion sort can be at least as good as merge sort for sizes of arrays up to 2000-2500. It seems that I underestimated the expensiveness and complexity of such operations as allocation of memory and utilization of recursion.

 Conclusions: For randomly generated arrays, insertion sort produces a faster algorithm for $n < 500$, while data structure B is faster for $n > 2500$. For n between 500 and 2500 the two data structures are indistinguishable.