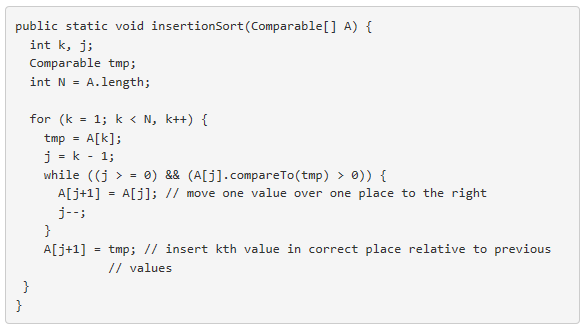
**ITE7107: Laboratory Exercise 004 Answers**

C++ Sorting

Note: Source codes used for this exercise, together with this answer sheet are uploaded in the GitHub repository: <https://github.com/rvillamangca/AMA-ITE7107-DataSturctureAlgorithms>. Program/s are written in C++ or Java.

Question 1: **Given the code below, answer the succeeding questions.**



**a. What is the running time for insertion when:**

**(1) the array is already sorted in ascending order?**

**(2) the array is already sorted in descending order?**

**b. One each iteration of its outer loop, insertion sort finds the correct place to insert the next item, relative to the ones that are already in sorted order. It does this by searching back through those items, one at a time. Would insertion sort be speeded up if instead it used binary search to find the correct place to insert the next item?**

Answer:

1. First of all the code given will not work, as is, because of the following syntax error:



Assuming we corrected all errors (see attached source), we notice that the method is implementing an ascending sort. The following are the complexity analysis:

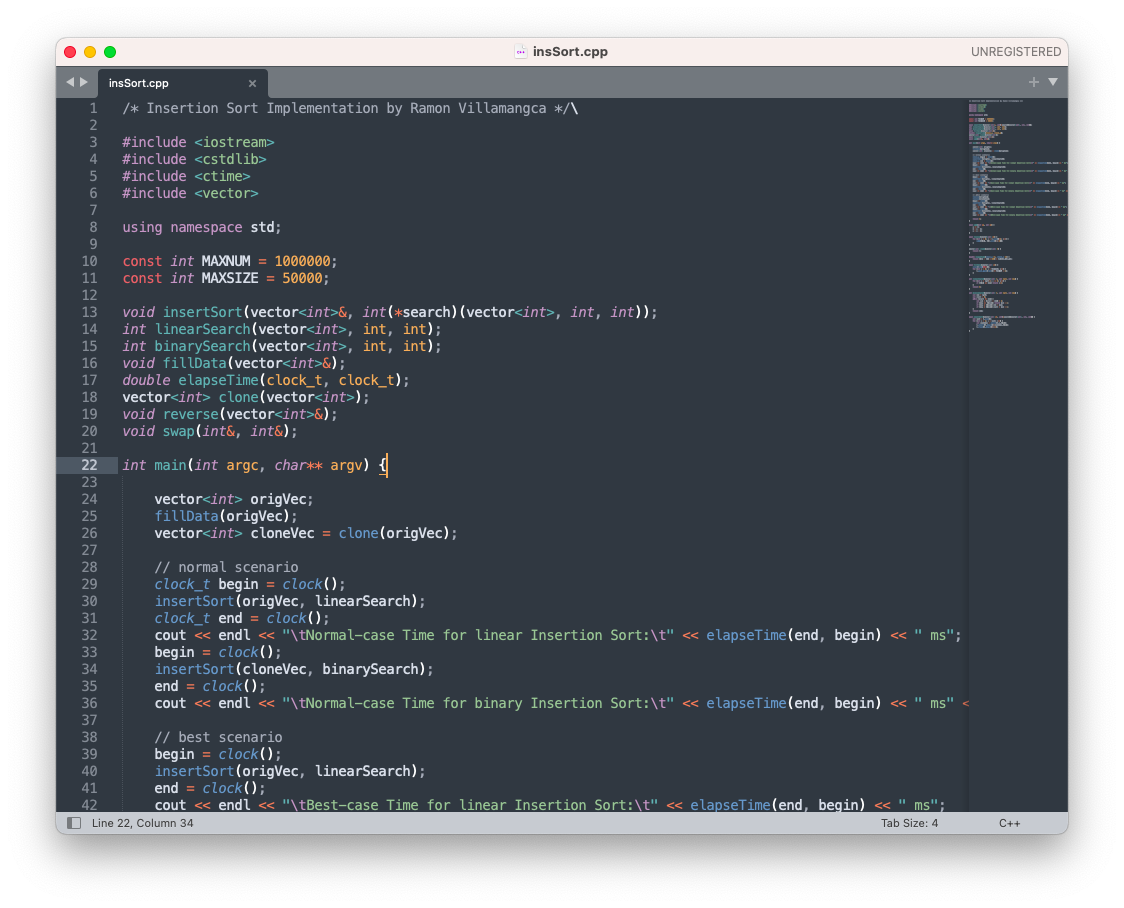
1. If the array to be sorted is already sorted in ascending order, the inner loop will not execute and since the outer loop is linear (O(N)) it follows that in this case the total complexity is in the order O(N) as well.
2. If the array to be sorted in already sorted descending, the inner loop will execute in the maximum number of times i.e. O(N). Therefore in this case, the total complexity is O(N×N) or O(N2).

Of course, this all “theory” and depends upon the implementation of the insertion sort. In the implementation made by this student for question “b”, it can be shown that the order of the array before sorting has little effect in the speed of sorting if the searching algorithm for insertion is not optimized.

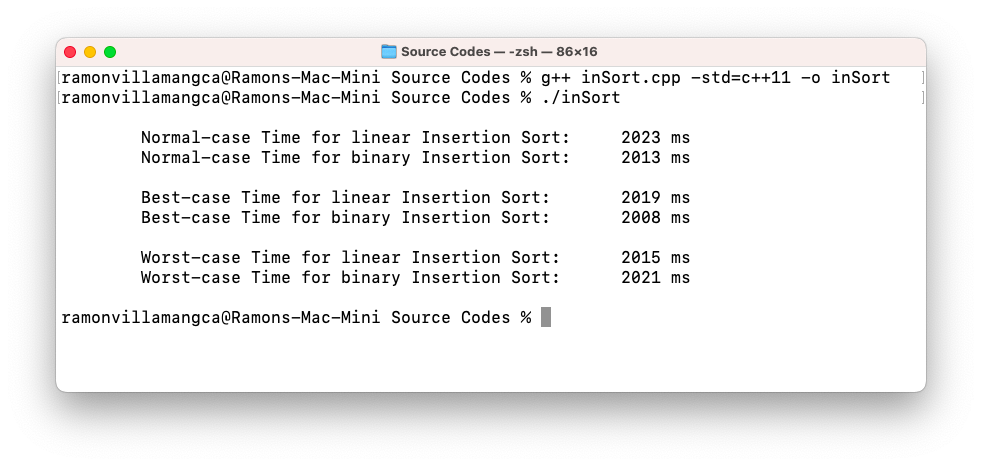
1. Linear search has complexity of O(N) while binary search has O(logN). Hence in theory, insertion sort using linear search has average complexity of O(N2) while insertion sort with binary search could reach a fairly desirable O(N×logN). In worst case, however, where the original array is sorted

descending, the binary search algorithm may still have to traverse all element, hence in that could still be O(N2).

These are all theory, so to settle, since this is a laboratory exercise anyway, a test is made to compare the “Linear Insertion Sort” and “Binary Insertion Sort” in the following C++ program:

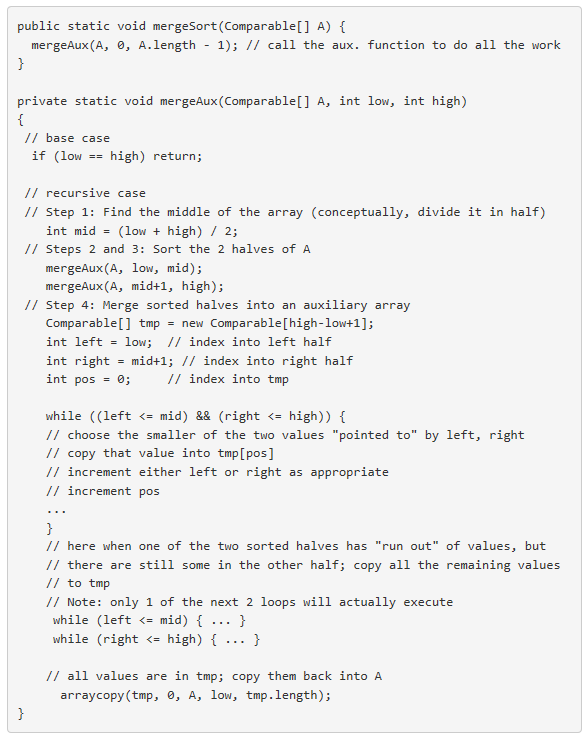


With the following result:



As we can see, binary searching does have a very slight advantage in sorting in the normal (random) and best (ascending) cases. In the worst scenario (descending), using a binary search is even worse.

Question 2: **Fill in the missing code in the mergeSort method.**



Answer:

Below is completed mergeSort program with the added “missing codes, highlighted:



The driver program, simplifies the algorithm using “int” instead of “Comparable”, then tests the merge sort algorithm and compares it with the insertion sort algorithm. As can be seen, the merge sort is way much faster the insertion sort.

