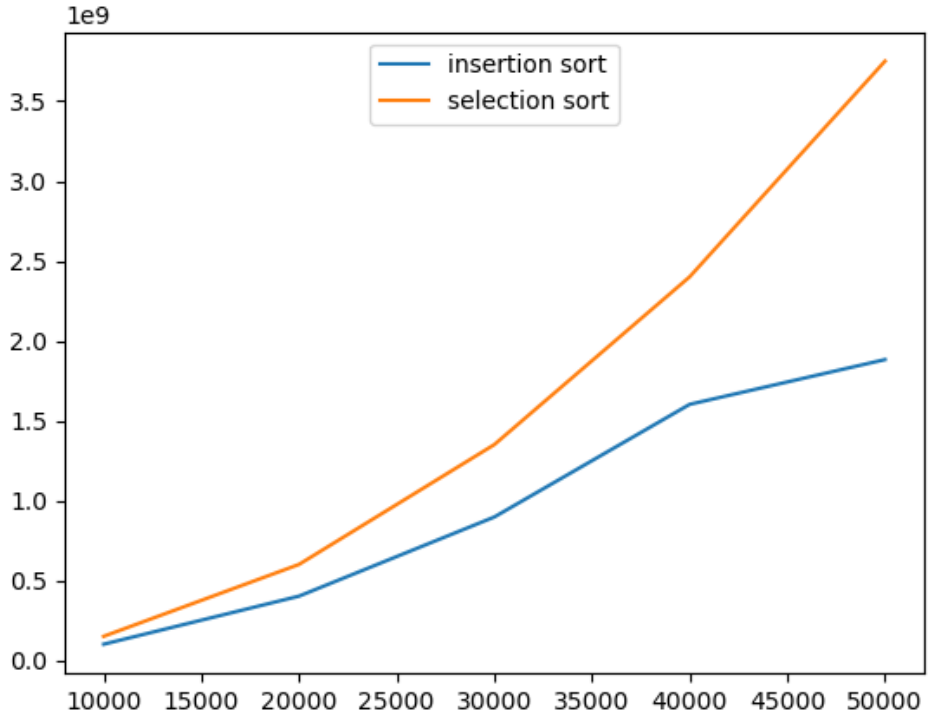

For random data, the insertion sort takes fewer steps than selection sort because:
For selection sort, it always needs to do a full loop-inside-a-loop no matter how the data is presented because when iterating each element in the data, the function will always go through ALL the elements after it to see if there is a minimum one. It means that the performance will remain the same for the same data size.
For insertion sort, it only has the worst performance when the data is pre-sorted in decreasing order in which it has to go through every former element when iterating. But if the random data generated is not in decreasing order, it will perform better because it doesn't need to go through many former elements while iterating. In fact, random-generated data cannot really be in decreasing or nearly-decreasing order if lucky, so the insertion sort rarely performs in the worst case and thus take fewer steps than selection sort generally. Also, the difference will be enlarged when the data size gets bigger because the randomness will increase and the data is less likely to get more decreasingly-ordered, so the insertion sort will perform better than selection sort then.