Writeup for Project 1

Part 1: Explanation of the Program

This Program mainly compares the performance of two different sorting methods, insertion sort, and selection sort, given data of different orders. Insertion sort evaluates each value in the list starting at the second position. It compares the element with its left neighbor, shifting right any greater values it encounters and ultimately produce a list with increasing order. Selection Sort, on the other hand, evaluates a certain cell and then determines which value should go to that cell. Starting at cell 0, the function finds the smallest element in the list and keep a record of its position. It then swaps the position of the value in cell 0 and that smallest element. After that, it starts to evaluate cell 1 and keep sorting until the whole list is in increasing order.

According to the requirement, three different types of lists should be tested, lists in increasing order, random order and decreasing order. Each type of these lists should be in length of 1000, 2500, 5000, 7500 and 10000. And a single list would be sorted by both insertion sort and selection sort. For the convenience of operation, create 6 2-dimensional lists, each contains 5 lists of the same order but with different lengths. Use a for loop to enable the functions to iterate through all the elements. In order to compare the result of the two sorting methods, the lists sorted by the two methods should be identical. Use list comprehension to create a list called inc_ins_list (which stands for increasing insertion list). It consists of 5 lists containing integers from 0 to 1000, 2500, 5000, 7500, and 10000. Use deepcopy to create an identical list that would be sorted by the selection method. In order to create lists in random order, import the shuffle function from the package random. Use a for loop to iterate through the lists in inc_ins_list. Use deepcopy to copy each list and use the shuffle function to randomize the order of elements in it. This produces a list containing lists in random order which would be sorted by insertion sort. Use deepcopy to attain an identical list for selection sort. In order to acquire lists of decreasing order, use list comprehension. Set step size to -1. This would create a list of lists with decreasing order.

Then define the insertion sort function. Use a for loop to iterate through all the elements starting from position 1 in a target list, each time record the current position k and the current element v. For each element v, use a while loop to change its position. If v is smaller than its left neighbor, swap their position. Now v is at position k-1 and the element originally at k-1 is at position k. Continue to compare the element at position k-2 to v. Keep changing until the element on the left of v is smaller than v. As the for loop starts from position 1, after the first iteration, elements at positions 0 and 1 are in increasing order. After the second iteration, elements at positions 0, 1, 2 are in increasing order. Thus, after nth iteration, the first n+1 element in the list are in increasing order. Eventually, the whole list is sorted.

Define the selection sort function. Use a for loop to iterate through all the cells in a list. Record the current cell number n and the current element v. For each cell, use a while loop to find the least element in the unsorted part of the list. Use variable cur to record the current least element. At the beginning, cur equals v. The while loop starts from cell n+1, compare cur to the element within that cell. If cur is larger than that element, change the value of cur to that element and record the position P of the cell. P is the position of the current least element. Go

to cell n+2 and repeat the process. After iterating through all the remaining elements, the value and position of the least element are recorded. Swap the position of v and the least element. Now the first n cell in sorted, and the for loop would continue to select the element for cell n+1. After each cell is evaluated in the for loop. The list is in increasing order.
Then define two functions called insertion_time and selection_time. They record the starting time, run the insertion or selection function, and then record the ending time of the sorting process. They print the time difference between the starting and ending time, which is the processing time of the function.

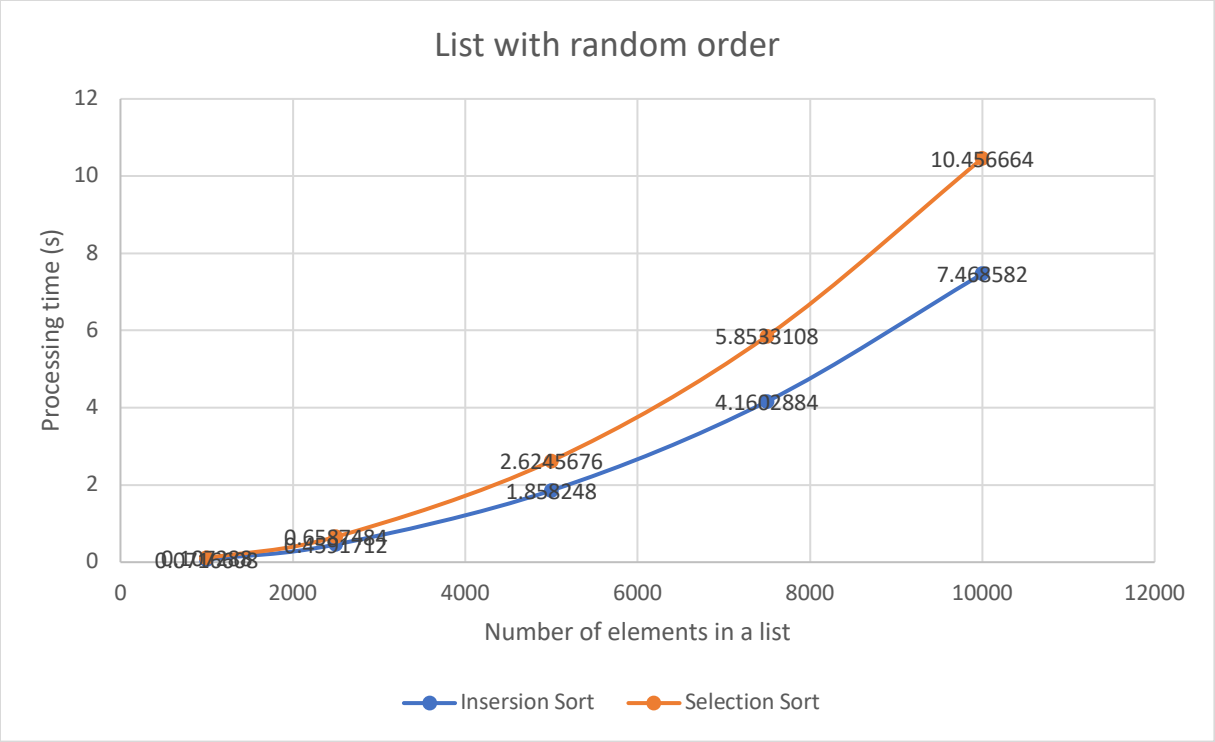
In the main function, use 6 for loops to iterate through the 6 lists created above, each containing 5 test lists of different lengths. Use insertion_time function and selection_time function to record the processing time of each list using different methods.

Part 2: Output analysis
(First Column: number of elements. Last Column: average processing time(s))

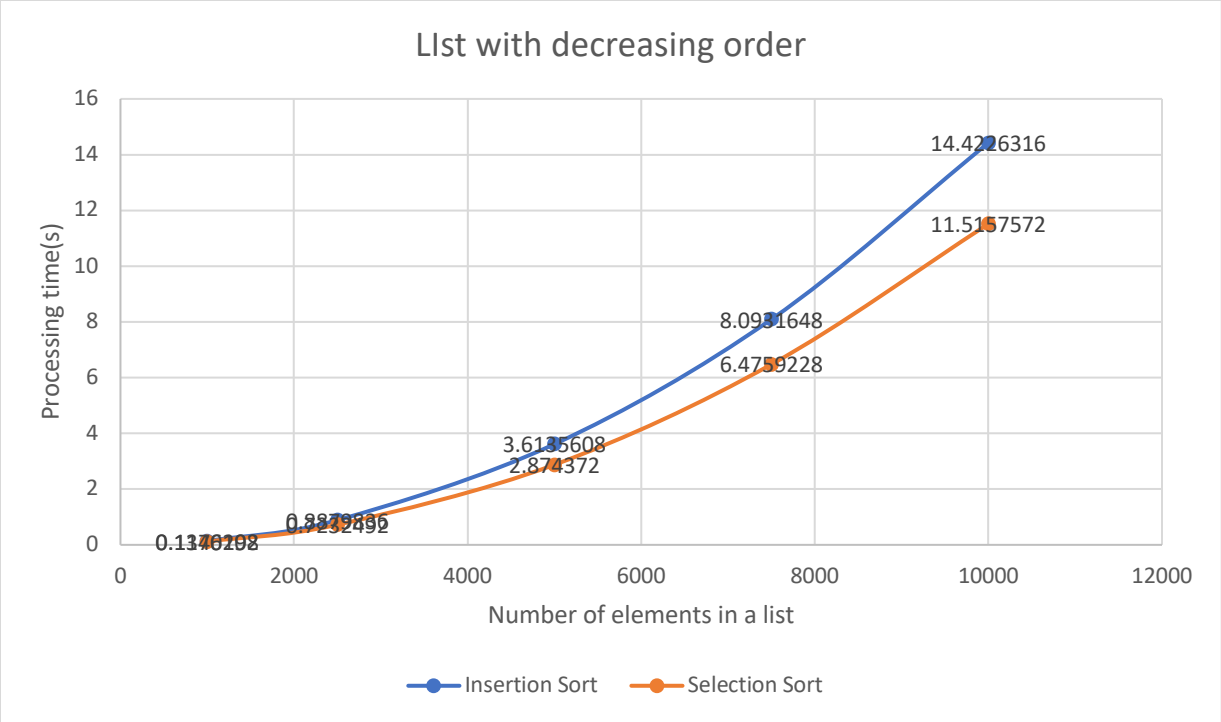
#elements						
increasing order						insertion average
1000	0.000232	0.000344	0.000306	0.000232	0.000238	0.0002704
2500	0.000722	0.000726	0.00065	0.000724	0.000668	0.000698
5000	0.001306	0.00136	0.001252	0.001304	0.001374	0.0013192
7500	0.001886	0.002072	0.001884	0.001886	0.00204	0.0019536
10000	0.002624	0.002998	0.002866	0.002754	0.002684	0.0027852
						selection average
1000	0.10888	0.108338	0.105372	0.107298	0.11483	0.1089436
2500	0.673382	0.661454	0.651318	0.66688	0.71003	0.6726128
5000	2.589436	2.538556	2.51532	2.546806	2.775512	2.593126
7500	5.81908	5.703502	5.642644	5.733812	6.223614	5.8245304
10000	10.33742	10.10884	10.35058	10.173382	11.13032	10.4201072
random order						insertion average
1000	0.074554	0.069588	0.074294	0.068394	0.071474	0.0716608
2500	0.443596	0.449834	0.4822	0.45279	0.467436	0.4591712
5000	1.823284	1.827534	1.997952	1.797674	1.844796	1.858248
7500	4.145654	4.096368	4.329782	4.118808	4.11083	4.1602884
10000	7.359952	7.313222	7.926504	7.492338	7.250894	7.468582
						selection average
1000	0.106004	0.102286	0.111468	0.104314	0.112368	0.107288
2500	0.648472	0.644596	0.656744	0.645028	0.698902	0.6587484
5000	2.582422	2.544526	2.633362	2.566592	2.795936	2.6245676
7500	5.827382	5.739526	5.658234	5.787046	6.254366	5.8533108
10000	10.52865	10.22338	10.10019	10.297384	11.13372	10.456664
decreasing order						Insertion average
1000	0.14156	0.130564	0.130246	0.130026	0.137658	0.1340108
2500	0.881358	0.866368	0.929858	0.880648	0.881686	0.8879836
5000	3.581976	3.559038	3.718732	3.588686	3.619372	3.6135608
7500	8.101828	8.054432	8.106292	8.07603	8.127242	8.0931648
10000	14.4258	14.36621	14.45113	14.395434	14.47459	14.4226316
						selection average
1000	0.11779	0.119916	0.115864	0.115308	0.119268	0.1176292
2500	0.724682	0.71591	0.708596	0.71103	0.756028	0.7232492
5000	2.89307	2.840496	2.82257	2.828572	2.987152	2.874372
7500	6.61846	6.390352	6.320634	6.348436	6.701732	6.4759228
10000	11.70793	11.37314	11.26889	11.316422	11.9124	11.5157572

The above table is the processing time for different lists using different sorting methods. The following graphs compares the time curves for two methods. Each graph represents the time curve for two methods to process lists of same order but different length.

The above graph shows two functions sorting lists with increasing order. It is clear in the graph that insertion sort is much faster than selection sort. Another difference is that the time curve for insertion sort is linear but the time curve for selection sort is a parabola. These differences can be explained by looking at the distinction between the two methods. For insertion sort, it iterates through each element and uses a while loop to keep moving that element to its left as long as it is smaller than its left neighbor. When sorting a list with increasing order, every element is always greater than the element on its left, so the condition for the while loop never occurs. The function would skip the operations inside the while loop and simply iterate through each element. So, there are totally n-1 iterations for a list of length n. The number of iterations grows linearly with the length of the list, which results in a linear time curve. While selection sort uses a different method, it has two layers of iterations. It iterates through each cell and for each cell it iterates through all the cells on its right to find the least element in the remaining unsorted list. So, the number of total iterations is n(n-1)/2 which is proportional to the square of the list length. And this number does not depend on the order of the sorted list. So, the result time curve is approximately a parabola and the sorting time is much larger than insertion sort especially at higher length because there are many more iterations occurring.



When sorting a list in random order, both time curves are parabolas. Note that the processing time for insertion sort is still shorter than the processing time for selection sort. As explained above, selection sort has two layers of iterations and this is a similar case, for a list of length n, the number of iterations is n(n-1)/2 so the time curve is a parabola. For insertion sort, the situation is different from the last graph. Here it also has two layers of iterations. The function iterates through each element and uses a while loop to keep moving it to the left until its left neighbor is smaller than itself. But the second iteration does not have to go through all the cells on the left of that element. For example, there is a list [0,1,3,2], and the while loop is moving the element 2 to the right position. It compares 2 to its left neighbor which is 3. As 3>2, it exchanges the position of 3 and 2 and now the list is [0,1,2,3]. It then compares the value of 1 to 2, 1 < 2 so the while loop ends. Although 2 is in position 3, the while loop only iterates once. Thus, an element at position n would be compared to at most n elements. The number of iterations in a while loop for that element is smaller or equal to n. The total iteration of the function is smaller than n(n-1). Also, as the position number n of an element increases, this element tends to be compared to more elements in a while loop. For example, an element at position 100 tends to be compared to more elements than an element at position 30. Thus, the number of iterations in a while loop is proportional to the position of an element in the list. So, the time curve of insertion sort is proportional to the number of elements squared. As the total iteration for insertion sort is smaller than n(n-1), the processing time is shorter than selection sort.



When sorting a list with decreasing order, both time curves are parabolas. However, insertion sort is slower this time. When sorting data in decreasing order, insertion sort iterates through every element in the list. For each element, it will have to compare n times if the element is in position n as the list is in decreasing order. So, the processing time of insertion sort increased greatly compared to the last two graphs as the total iteration is now n(n-1)/2. The number of iterations of selection sort is still n(n-1)/2. Although these two functions have the same number of iterations, the insertion sort function requires more rewrites to the value in a certain cell. Every time a comparison is made, insertion sort would swap two values and thus require more time for the computer to process. However, on the other hand, selection sort requires fewer rewrites in the memory than insertion sort thus requires less processing time even though they have the same number of iterations. This is the reason why selection sort performs faster when sorting lists of decreasing order. There is a slight increase in time for selection sort to sort data in decreasing order than in any other order. The reason is that when selection sort evaluates data with decreasing order, it will also have to change the memory of position and least element value more often. This also requires more rewrites to the memory of computers. Thus, the processing time is also a little longer comparing to sorting list in other orders.

Part 3 Overall analysis

Insertion sort is a better method for sorting data. It is an effective method that requires fewer iterations – which means making fewer comparisons than selection sort. Only in the worst case where data are presented in decreasing order, insertion sort makes the same number of comparisons as selection sort. Although insertion sort may have the problem of requiring more changes in the memory, it is overall a better way of list sorting.