

Lab 6: Basic Comparison Sorts

A comparison sort is a type of sorting algorithm that compares elements in a list (array, file, etc) using a comparison operation that determines which of two elements should occur first in the final sorted list. The operator is almost always a **total order**:

1. $a \leq a$ for all a in the set
2. if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity)
3. if $a \leq b$ and $b \leq a$ then $a=b$
4. for all a and b , either $a \leq b$ or $b \leq a$ // any two items can be compared (makes it a total order)

In situations where three does not strictly hold then, it is possible that a and b are in some way different and both $a \leq b$ and $b \leq a$; in this case either may come first in the sorted list. In a **stable sort**, the input order determines the sorted order in this case.

The following link shows visualization of some common sorting algorithms:

<https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html>

Your goal for this lab is to implement simple versions of Insertion Sort - **insertion_sort(alist)**, and Selection Sort - **selection_sort(alist)**, that will sort an array of integers and **count the number of comparisons**. Each function takes as input a list of integers, sorts the list counting the comparisons at the same time, **and returns the number of comparisons**. After the function completes, the “**alist**” should be sorted.

The worst-case runtime complexity is $\Theta(n^2)$ for selection and insertion sort. Why? Write out the summation that represents the number of comparisons.

Note: There is a fundamental limit on how fast (on average) a comparison sort can be, namely $\Omega(n \log n)$.

Fill out and submit via GitHub, a table similar to the table below as well as answers to the questions below.

Lab 6

Selection Sort		
List Size	Comparisons	Time (seconds)
1,000 (observed)	499500	0.08872747421264648
2,000 (observed)	1999000	0.3720388412475586
4,000 (observed)	7998000	1.4451372623443604
8,000 (observed)	31996000	5.973026514053345
16,000 (observed)	127992000	24.529415369033813
32,000 (observed)	511984000	97.54634666442871
100,000 (estimated)	4999500000	887
500,000 (estimated)	124999750000	3240
1,000,000 (estimated)	499999500000	88700
10,000,000 (estimated)	4999999500000	887000

Insertion Sort		
List Size	Comparisons	Time (seconds)
1,000 (observed)	247986	0.07679533958435059
2,000 (observed)	1018717	0.34607529640197754
4,000 (observed)	3995264	1.3723299503326416
8,000 (observed)	16112194	5.777550935745239
16,000 (observed)	64667449	22.22055149078369
32,000 (observed)	257507119	101.42142844200134
100,000 (estimated)	247986000	767.9
500,000 (estimated)	991944000	3071
1,000,000 (estimated)	2479860000	76790
10,000,000 (estimated)	24798600000	767900

1. Which sort do you think is better? Why?

Both sorts are around the same efficiency but insertion is slightly faster.

2. Which sort is better when sorting a list that is already sorted (or mostly sorted)? Why? Insertion should be better at sorting an already sorted list because it doesn't have to iterate through the entire list like selection sort has to.

3. You probably found that insertion sort had about half as many comparisons as selection sort. Why? Why are the times for insertion sort **not** half what they are for selection sort? (For part of the answer, think about what insertion sort has to do more of compared to selection sort.)

Insertion sort only must iterate until it finds a lower value while selection sort has to iterate through the entire list to look for the minimum. The times are not half because insertion sort must switch elements more than selection sort does.