

Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre



Lab Report I

[Code No: COMP 314]

Submitted by:

Prashant Manandhar

Roll No:30

Group: Computer Engineering

Level: III year/II sem

Submitted to:

Dr. Rajani Chulyadyo

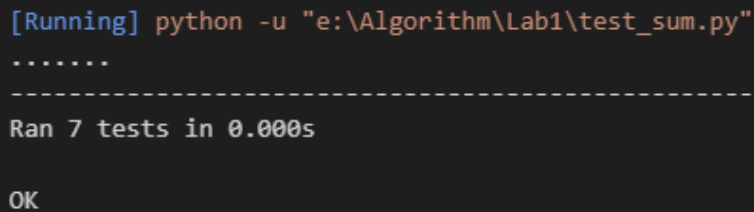
Department of Computer Science and Engineering

Submission Date:24/04/2024

Task I - Test For Sum

Here, we first make the function to add all the elements of the array that are passed into the function. After that, we have to make the test function for the sum of the array containing only positive numbers, empty data, only negative numbers, mixed numbers of positive and negative, only one element, large numbers, and float numbers.

Output:



```
[Running] python -u "e:\Algorithm\Lab1\test_sum.py"
.....
-----
Ran 7 tests in 0.000s

OK
```

Task II - Insertion Sort and Selection Sort

Insertion Sort: In Insertion Sort, the algorithm builds the sorted list one element at a time by repeatedly taking an unsorted element and inserting it into its correct position in the sorted part of the list. It iterates through the unsorted portion, comparing each element to the elements in the sorted portion and shifting elements as necessary until the entire list is sorted. The time complexity of the insertion sort is $O(n^2)$ in the worst-case scenario and average-case scenario and $O(n)$ in the best-case scenario.

Selection Sort: Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted part. This process is repeated for the remaining unsorted portion until the entire list is sorted. The time complexity of the insertion sort is $O(n^2)$ in the worst-case scenario, average-case scenario, and best-case scenario.

Task III - Some test cases to test sorting Algorithm

We have generated various tests for the sorting algorithm. the array containing only negative elements, only positive elements, in descending order, in ascending

order, duplicate elements, single data, and empty are tested in those two algorithms.

Output:

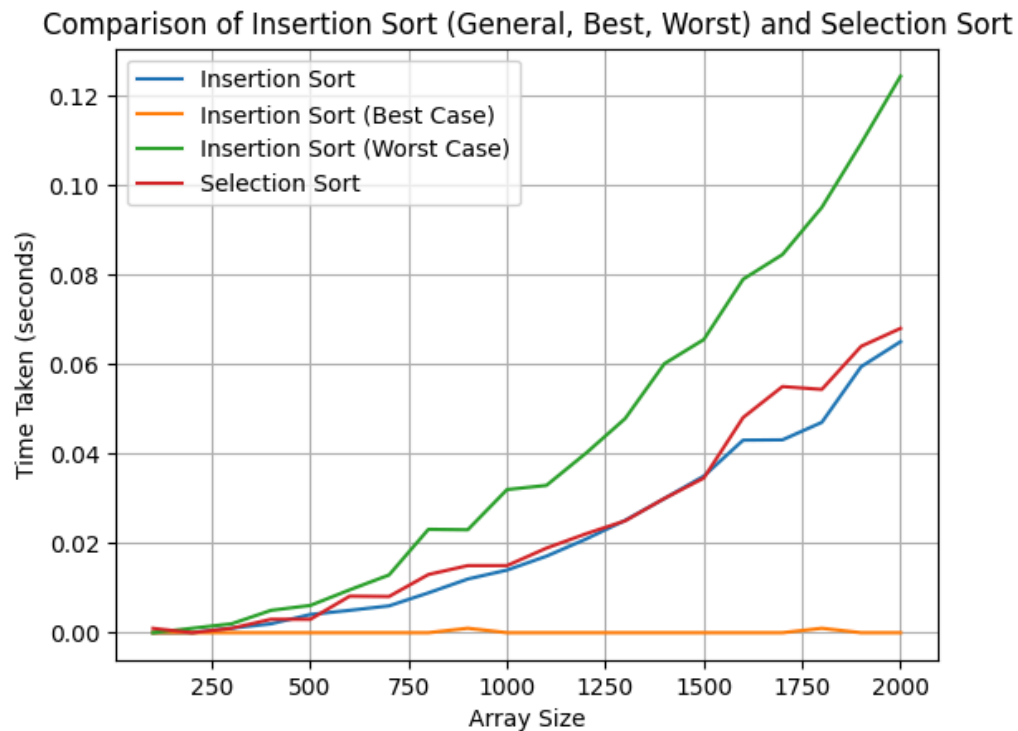
```
[Running] python -u "e:\Algorithm\Lab1\test_sorting.py"
.....
-----
Ran 8 tests in 0.000s
OK
```

Task IV: Generate some random inputs for your program and apply both insertion sort and insertion sort algorithms to sort the generated sequence of data. Record the execution times of both algorithms for inputs of different sizes. Plot an input-size vs execution-time graph.

The program generated a random number using the random.randint which generates the random number from range -1000 and 1000. About 2000 random numbers were generated. The generated random were sorted in ascending order and descending order for best case and worst case checking of insertion sort. Then the generated random numbers are applied to both the insertion sort and selection sort algorithms to sort the data. The execution times of both algorithms were recorded for inputs of different sizes, and an input-size vs execution-time graph was plotted.

The trend of execution time for insertion sort (worst case and average case) and selection sort as the input size increases is generally proportional to the square of the input size for the worst-case scenario. This is because both algorithms have a worst-case time complexity of $O(n^2)$, where n is the input size. The figure below provides a visual representation of how the execution time of the insertion sort and selection sort increases as the input size grows. In theory, the time complexity of the best case for insertion sort is $O(n)$ but due to the hardware and software limitations, the time taken by the best case of insertion sort cannot be captured effectively. the graph shows that the largest time is taken by the insertion sort

(worst case) followed by the selection sort, insertion sort (average case), and then insertion sort (best case).



BreakPoints:	Insertion Sort (General Case):	Insertion Sort (Best Case):	Insertion Sort (Worst Case):	Selection Sort:
100	0.0	0.0	0.0	0.0009715557098388672
200	0.0	0.0	0.0009999275207519531	0.0
300	0.0010113716125488281	0.0	0.002000570297241211	0.0009975433349609375
400	0.001995563507080078	0.0	0.0049991607666015625	0.0029981136322021484
500	0.004072666168212891	0.0	0.0060961246490478516	0.0030002593994140625
600	0.0050008296966552734	0.0	0.009583473205566406	0.008168220520019531
700	0.005999088287353516	0.0	0.012902975082397461	0.008094310760498047
800	0.00890660285949707	0.0	0.023090804561767578	0.013004541397094727
900	0.011997461318969727	0.0010020732879638672	0.023004770278930664	0.014997005462646484
1000	0.013999700546264648	0.0	0.03199481964111328	0.015006303787231445
1100	0.017096757888793945	0.0	0.03292131423950195	0.01889944076538886
1200	0.020003825759887695	0.0	0.04008078575134277	0.02200329605102539
1300	0.025101661682128906	0.0	0.04784440994262695	0.024999085694885254
1400	0.029993534088134766	0.0	0.0601345718383789	0.03000688528564453
1500	0.03499937057405117	0.0	0.06551432609558105	0.034566402435302734
1600	0.043036460876464844	0.0	0.07899594306945801	0.040094987869262695
1700	0.04309844970703125	0.0	0.08451366424560547	0.05500960350036621
1800	0.04700970649719238	0.0009984970092773438	0.09504938125610352	0.05436968803405762
1900	0.05945563316345215	0.0	0.10936737060546875	0.06400561332702637
2000	0.06503438949584961	0.0	0.124359130859375	0.0680079460144043

GitHub Link:

The code of the Lab 1 can be found in the following GitHub Link:

<https://github.com/Eemayas/Algorithm-and-Complexity-COMP-314-Labs/tree/main/Lab1>