

Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre
Estd: 1991 A.D.



LAB-1
Algorithms and Complexity
[Code No: COMP 314]

Submitted by:
Nigam Niraula (32)

Submitted to:
Dr. Rajani Chulyadyo
Department of Computer Science and Engineering

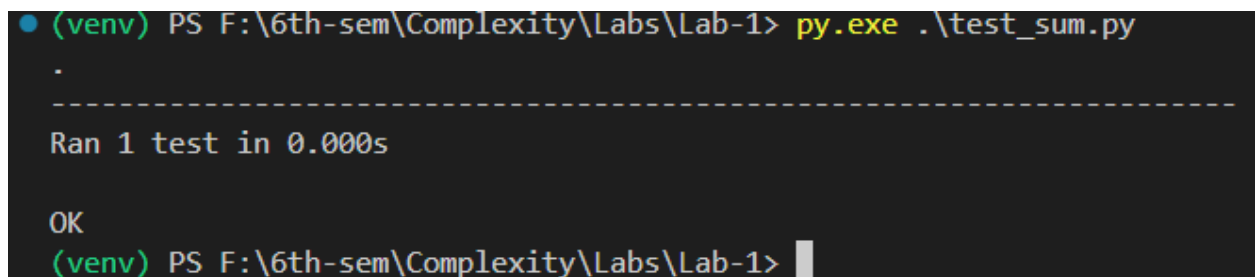
Submission Date: 04/24/2024

LAB DESCRIPTION:

In the first lab of this course, we learnt about testing algorithms using “unittest” library provided in Python. Firstly, a simple sum function was tested to demonstrate the functioning of the library. This task was undertaken concurrently with the assessment of two traditional sorting algorithms: Selection Sort and Insertion Sort. Firstly, the algorithm for the two sorts was written and test cases for the sorting algorithms were written for checking the algorithm on random array, sorted array, reverse sorted and empty array. Once the test cases were validated, I computed the time taken for both sorting algorithms to sort the same array, also for insertion sort best and worst cases were analyzed. To compare the sorting algorithms, a graphical approach was taken.

This lab provided a practical experience with sorting algorithms, giving us valuable insights into their creation, testing, and most importantly, their analysis to find the most effective solution. By actually using and testing the algorithms, we developed a thorough understanding of how they are developed and improved. Additionally, evaluating their performance systematically helped us better grasp the nuances of algorithm efficiency, enabling us to choose the appropriate algorithm for a specific task with confidence. We also had a valuable hands-on experience, highlighting the importance of testing in the creation and assessment of sorting algorithms. By using Python's 'unittest' library, we carefully examined the accuracy and strength of our sorting algorithms in various situations. This thorough testing process guaranteed that our algorithms were capable of handling different types of inputs, such as random, sorted, reverse-sorted, and even empty arrays. By subjecting our algorithms to such extensive testing, we not only confirmed their effectiveness but also strengthened our trust in their dependability and flexibility. This focus on testing emphasized the significance of thorough validation.

Output:

A terminal window with a dark background and light-colored text. The prompt is '(venv) PS F:\6th-sem\Complexity\Labs\Lab-1>'. The command 'py.exe .\test_sum.py' has been entered. The output shows a single dot '.' on the first line, followed by a dashed line '-----' on the second line. The third line shows 'Ran 1 test in 0.000s'. The fourth line shows 'OK'. The prompt is repeated at the bottom: '(venv) PS F:\6th-sem\Complexity\Labs\Lab-1>'.

```
(venv) PS F:\6th-sem\Complexity\Labs\Lab-1> py.exe .\test_sum.py
.
-----
Ran 1 test in 0.000s

OK
(venv) PS F:\6th-sem\Complexity\Labs\Lab-1>
```

Fig 1: Testing sum function with various test cases

```
(venv) PS F:\6th-sem\Complexity\Labs\Lab-1> py.exe .\sorting_algorithms\Insertion_Sort\test_insertion_sort.py
.
-----Ran 1 test in 0.000s
OK
```

Fig 2: Testing Insertion Sort with various test cases

```
(venv) PS F:\6th-sem\Complexity\Labs\Lab-1> py.exe .\sorting_algorithms\Selection_sort\test_selection_sort.py
.
-----
Ran 1 test in 0.000s
OK
```

Fig 3: Testing Selection Sort with various test cases

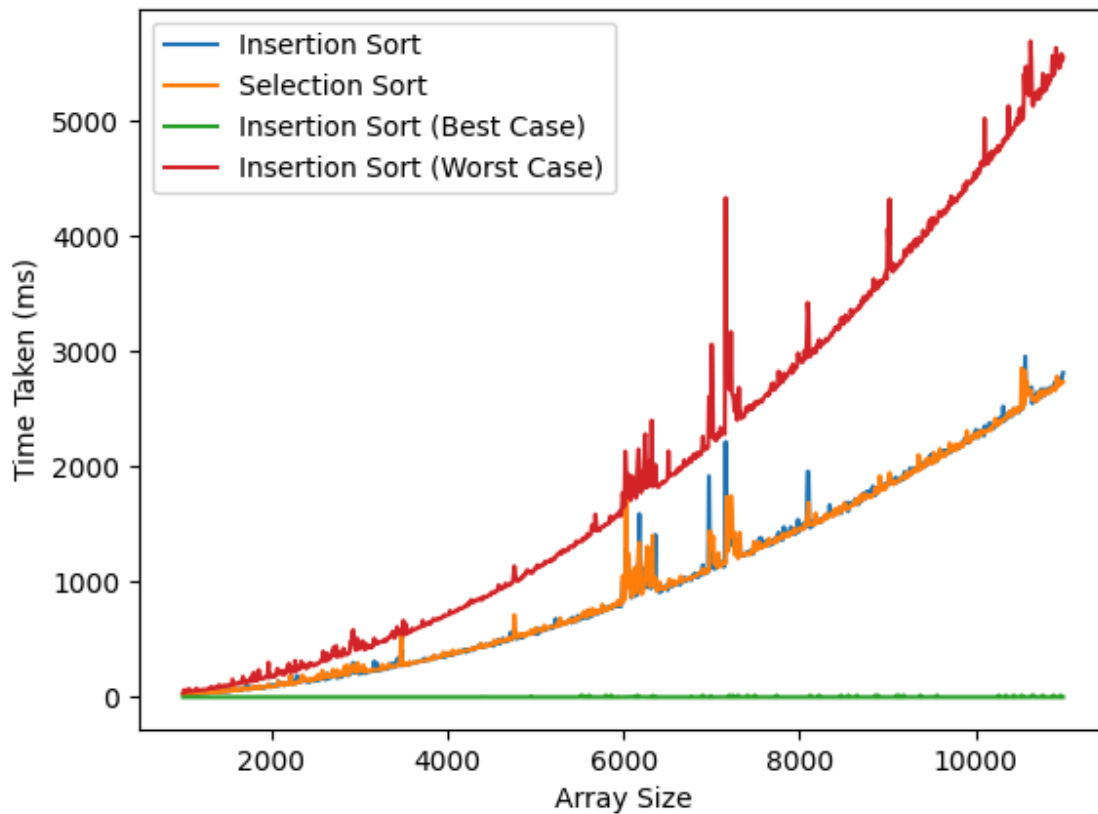


Fig 4: Comparison between Selection sort and Insertion Sort

OBSERVATION:

The findings from this lab highlight the crucial role of testing in algorithm development, showing how important it is in wider software engineering settings. Through careful examination of smaller sections of code with thorough testing, we establish a strong base for larger programs to ensure their reliability and effectiveness. This method of modular testing not only makes debugging more efficient but also boosts confidence in the system's overall integrity.

Additionally, the visual depiction of how well algorithms perform provided clear and valuable insights into the time complexities associated with sorting algorithms. The graph effectively highlighted how Insertion Sort's efficiency varies depending on different situations, with its best-case scenario of $O(n)$ standing out against the worst-case scenario of $O(n^2)$. This emphasized the significant role that the distribution of input data plays in determining algorithm performance. Moreover, when comparing Selection Sort and Insertion Sort, it became evident that both algorithms generally have similar time complexities, typically behaving with $O(n^2)$ efficiency.

SOURCE CODE:

The source code can be found at:

[Complexity-Labs/Lab-1 at master · ninix07/Complexity-Labs \(github.com\)](https://github.com/ninix07/Complexity-Labs)