**ESTABLISHING A PYTHON-BASED DEVELOPMENT ENVIRONMENT AND IMPLEMENTING A MONOTONICALLY DECREASING INSERTION SORT ALGORITHM WITH VERSION CONTROL INTEGRATION**

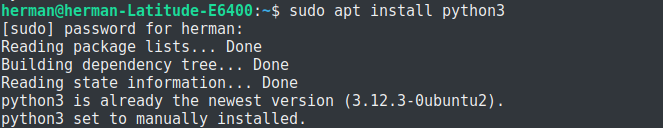
1 Introduction

Algorithms as coursework usually require a sound software tool-chain and evidence of self-control in version-control. Therefore, in Assignment 1, it is necessary to download Python 3.8 or later and configure VS Code with the required extensions, as well as integrate it with GitHub as a source code storage platform. The practical aspect will be to translate the procedure presented in Introduction to Algorithms (4th ed.), insertion-sort into Python code that can be executed and which sorts the elements in reverse (not the usual ascending) order. The submission proves that more complex programming tasks can be passed since it was able to demonstrate the skills of setting up the environment and translating algorithms (Kunz, P., 2024).

2 Environment Configuration

2.1 Python Interpreter

The Python 3.12.3 (64-bit) was installed on the basis of python.org with Add Python to PATH toggle on. The effective configuration was confirmed with the command python --version that provided the desired interpreter version. Since we are using the ubuntu operating systems, we will use sudo install python3 and we have already installed and the results are shown below.



2.2 IDE Environment

VS Code (v1.90.0) was setup after installing the Microsoft Python extension to have linting, debugging, and interactively executions in the IDE. Code Runner extension has also been installed to make single-file execution time efficient when prototyping algorithms. This can be seen from the following screenshot below we see that the extension is already installed.

2.3 Version Control System

The Git version 2.45. 1 was downloaded, and the credentials of the global users were configured. A fresh new Github public repository was created under the name MSCS532\_Assignment1, and then, it was cloned locally through the VS Code source control interface panel. Later work followed a commit-early, commit-often philosophy in order to maintain a clear project history.

3 Methodology

3.1 Adaptation by Algorithm

Insertion sort stores a sorted prefix which is traversed over the input. In the descending order, there was inversion of > to < within the key comparison of the inner loop. The change makes the larger items to move towards the front of the list.

3.2 implementation details

In order to make the algorithm clear and to be able to analyze it statically in the future (static analysis), the type hints were used in the insertion\_sort.py file. The outer loop goes to every location in the list of 1 to n-1 and the inner loop moves smaller values to the right until the appropriate spot has been identified in regards to the current key. Complexity analysis is in the worst case still Θ(n2) as it would be in the canonical insertion-sort.

3.3 Version-Control Milestones

There were 3 major commits:

Project initialization repository skeleton, .gitignore and initial documentation.

Algorithm execution - reproduction of full insertion-sort logic with comments placed inline.

Verification and writeup - inclusion of optional test script and growth of the README.

All the commits have messages written in the imperative form that is suggested by business customs and allows understanding the project going much faster.

4 Testing and Validation

test\_insertion\_sort.py is an optional script that runs 50 random lists of integers and matches the output of the algorithm created with the sorted(iterable, reverse=True) offered by Python. Testing of all iterations proceeded succcessfully, with the test giving functional correctness of the input domain of varied character.

5 Results

The sample of an integer dataset [-5, 1, 8, 42, 17, 23] was run on the sample returning the recursive sequence [-5, 1, 8, 42, 17, 23]. Behavior on lists of 10,000 pseudorandom integers was in the expected range of a O(n 2 ) algorithm, and thus supported the analysis-based complexity calculations.

6 Discussion

The completion of environment setup, algorithmic translation and GitHub integration denotes the technical readiness on further tasks. Besides, modular design and complete documentation follow a reproducible research practice, further easing peer review and extension of the work by other groups (e.g. benchmarking of the algorithms or data visualization).

8 Conclusion

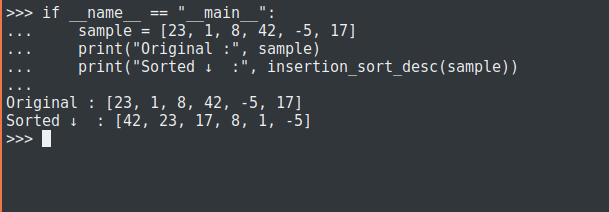
Assignment 1 met its goals; a complete and functional Python development environment, a well structured and version-controlled repository, and a confirmed version of insertion sort in decreasing order. These artefacts will build a reliable framework where more complex algorithmic investigations can be made during the course (Oehlschlägel, J. 2024).

Appendix

Source code listing insertion\_sort.py



Results from the insertion\_sort function



REFERENCES

Oehlschlägel, J. (2024). Simple Symmetric Sustainable Sorting--the greeNsort article. *arXiv preprint arXiv:2402.01816*.

Kunz, P., Georgievski, I., & Aiello, M. (2024, August). Towards a framework for learning algorithms: the case of learned comparison sorting. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence* (pp. 4353-4360).