Identifying how Insertion Sort Runtime

Changes as Input Size Grows

Lab #2

By

Michael Harden

CS 303 Algorithms and Data Structures

September 7, 2021

1. **Problem Specification**

The goal of this lab was to implement the Insertion Sort Algorithm, and with it sort a list of random numbers, stored in multiple input files of varying length. Then, compare the times taken to sort each file, and finally, identify how the time required to sort each input grows, as input size increases.

1. **Program Design**

This program requires two classes, InsertionSort and InsertionDriver. The InsertionSort class contains the required methods to sort a list with the insertion sort algorithm. The InsertionDriver class contains methods to; parse a text file of integers, with any delimiter; calculate the runtime of sorting all required files; and test the correctness of the insertion sort algorithm implementation.

To implement the Insertion Sort Algorithm, the pseudocode in the provided lab2 document file, was used.

To calculate the run times for each required file, the following procedure was followed.

1. Parse the content of the file into a list of integers, starting with the smallest file and for each iteration move towards the largest file.
2. Create an instance of the InsertionSort class and pass the list of parsed integers into the InsertionSort constructor’s parameter.
3. Store the current time just before sorting, then call the sort method on the instance of InsertionSort. Once the list has been completely sorted, take the difference of the time directly after sorting and the time just before sorting.
4. Log the time take to sort, to the console.
5. Continue to the next iteration.

The following constructor and method is defined within the InsertionSort class.

1. \_\_init\_\_(data):

Defines the instance variable, data, and sets its value to the value passed in the data parameter.

1. sort():

Implements the pseudocode for the Insertion Sort algorithm whcoh is provided in the assignment lab2 document, and sorts list of integers stored in the instance variable, data.

The following constructor and methods are defined within the InsertionDriver class.

1. \_\_init\_\_():

Defines the instance variable data, and sets its type as a list[int]

1. load\_file(filepath, delimiter=’ ‘):

Finds the file on the path passed into the parameter and parses its context into a list storing a type of int. The file is parsed using the delimiter passed into the delimiter parameter.

1. time\_method():

Calculates the time to sort the content of each required file and prints that those times to the console. The process used is described above.

1. test(lst:list[int]):

Sorts the lists passed in the parameter lst and returns the sorted list.

To parse the files, the open() and read() built in methods are used.

1. **Testing**

To test the Insertion Sort algorithm six test cases were used to attempts and break the algorithm each in different ways.

|  |  |  |
| --- | --- | --- |
| **Test Number** | **Input** | **Expected Output** |
| **#1** | [10, 4, 6, 3, 2, 9, 16, 0, 3, -1] | [ -1, 0, 2, 3, 3, 4, 6, 9, 10, 16] |
| **#2** | [4, 3, 3, 3, 2, -2] | [-2, 2, 3, 3, 3, 4] |
| **#3** | [-3, -103, - 5, -2,  -10, -44, -31] | [-103, -44, -31,  -10, -5, -3, -2] |
| **#4** | [10] | [10] |
| **#5** | [10, 9, 8, 7, 6, 5, 4, 3, 2, 1] | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] |
| **#6** | [] | [] |

1. **Timing**

The times to sort the provided files are as follows:

|  |  |
| --- | --- |
| **File name** | **Sorting Time** (seconds) |
| input\_100.txt | 0.0064421 |
| input\_1000.txt | 0.084224425 |
| input\_5000.txt | 1.81907511 |
| input\_10000.txt | 6.46251702 |
| input\_50000.txt | 182.156554 |
| input\_100000.txt | 936.355143 |
| input\_500000.txt | Could not sort in a reasonable time on my personal computer. |

1. **Analysis and Conclusions**

In majority of conditions Insertion sort would not be the ideal sorting algorithm. With a worst-case time complexity of O(n^2), sorting large amounts of data would take an extremely long time. Furthermore, as the data size grows linearly, the time taken to sort the data would grow quadratically, resulting in a poor performance.

There are a few situations where Insertion Sort could be a practical sorting algorithm. For instance, if the data being sorted is almost entirely sorted and only a few comparisons would need to be made, the time complexity would draw closer to the best case O(n) time. However, when this situation cannot be assumed, Insertion Sort’s time complexity is not idea.

The following graph displays how the time grows in relation to a linear input size growth (data collected with provided text files):

1. **Screen Shots of Code and Output**

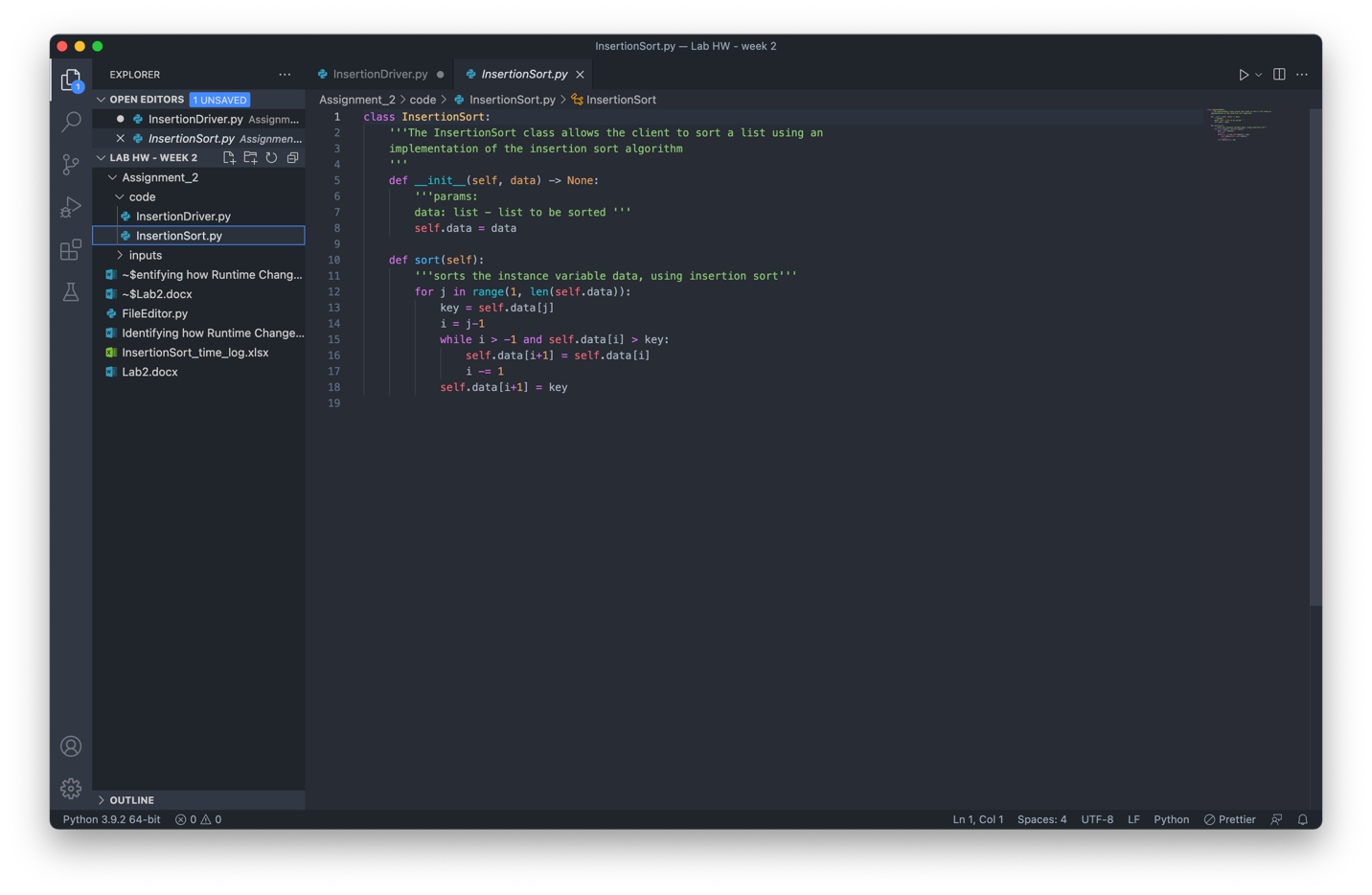
The first two screen shots are of the InsertionDriver class:

A picture containing text, monitor, screenshot, silver

Description automatically generatedA picture containing text, monitor, screenshot, black

Description automatically generated

The third screen shot is of the InsertionSort class:



The final screen shot is of the output when timing the Insertion Sort algorithm with the InsertionSort driver class: Text

Description automatically generated