

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.

## 2. 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.

- a) 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.
- b) Create an instance of the InsertionSort class and pass the list of parsed integers into the InsertionSort constructor's parameter.
- c) 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.
- d) Log the time take to sort, to the console.
- e) Continue to the next iteration.

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

- a) `__init__(data)`:  
Defines the instance variable, data, and sets its value to the value passed in the data parameter.
- b) `sort()`:  
Implements the pseudocode for the Insertion Sort algorithm which 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.

- a) `__init__()`:  
Defines the instance variable data, and sets its type as a list[int]

- b) `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.
- c) `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.
- d) `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.

### 3. Testing

To test the Insertion Sort algorithm six test cases were used to attempt 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	[]	[]

### 4. 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

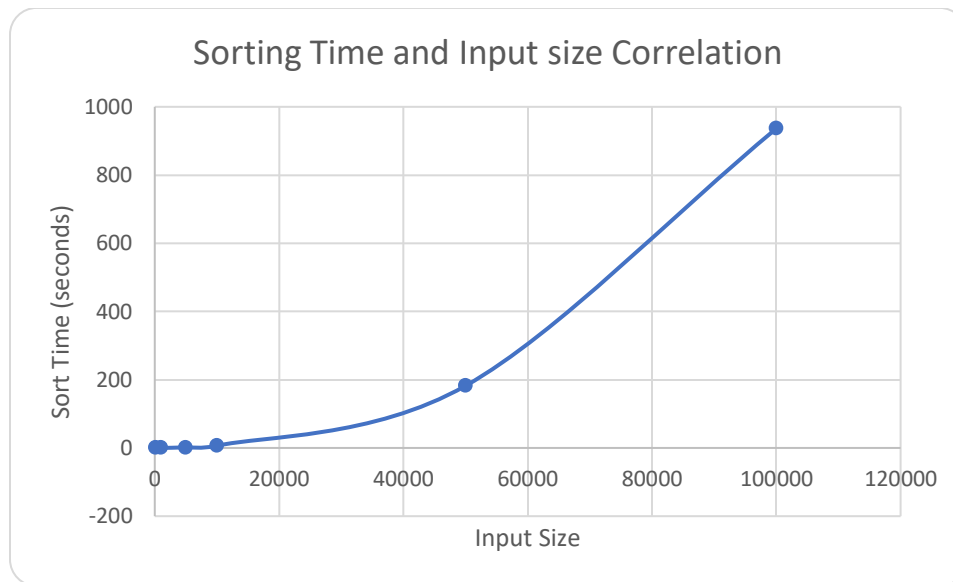
input_50000.txt	182.156554
input_100000.txt	936.355143
input_500000.txt	Could not sort in a reasonable time on my personal computer.

## 5. 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.

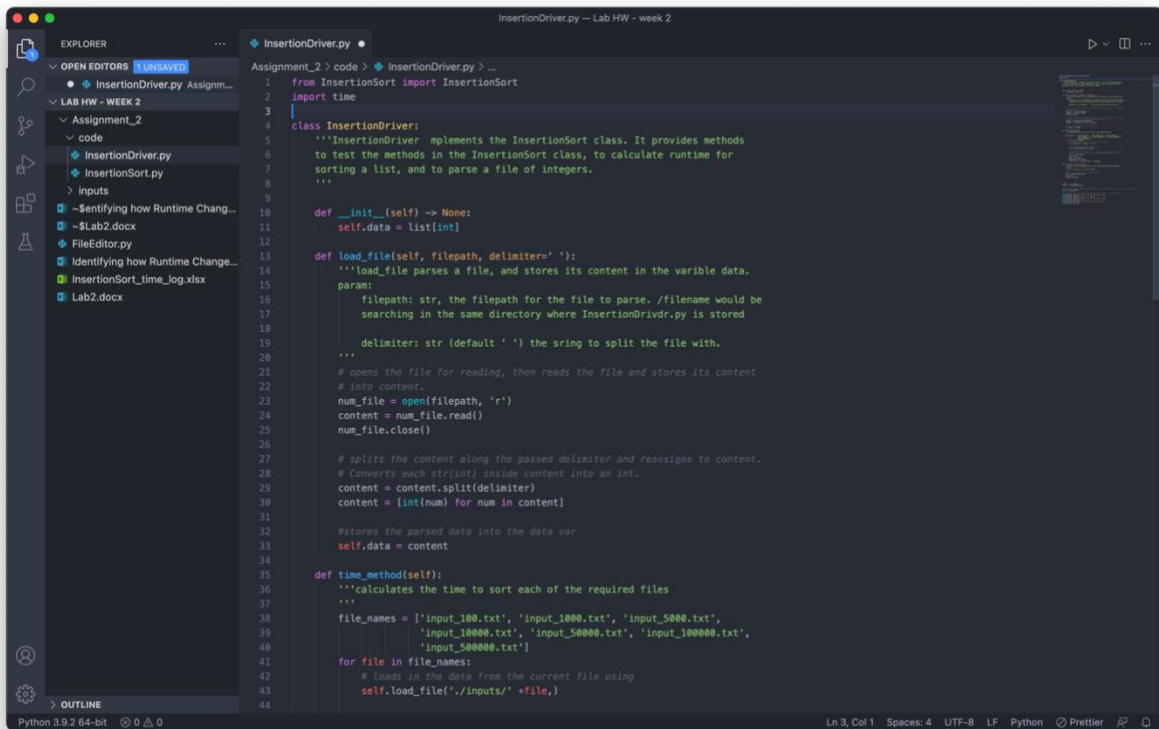
There are a few situations where Insertion Sort could be a practical sorting algorithm. For instance, if the data being sorted is almost entirely sorted and only a few comparisons would need to be made, the time complexity would draw closer to the best case  $O(n)$  time. However, when this situation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

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

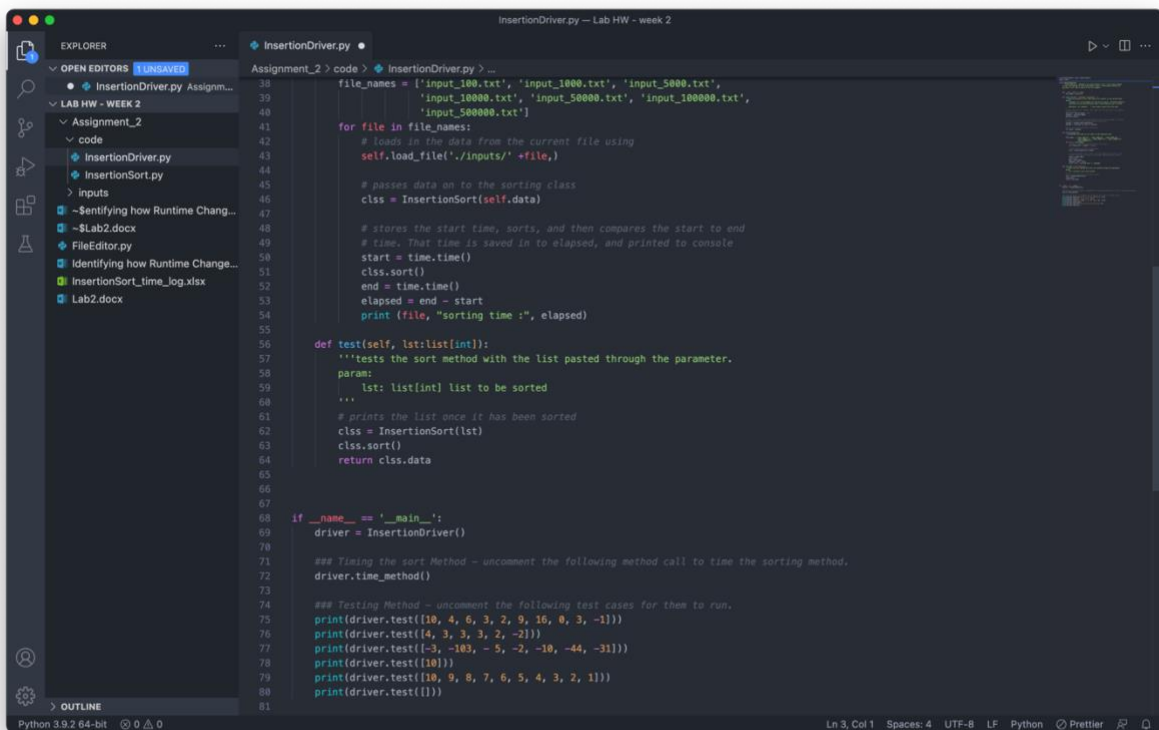


## 6. Screen Shots of Code and Output

The first two screen shots are of the InsertionDriver class:

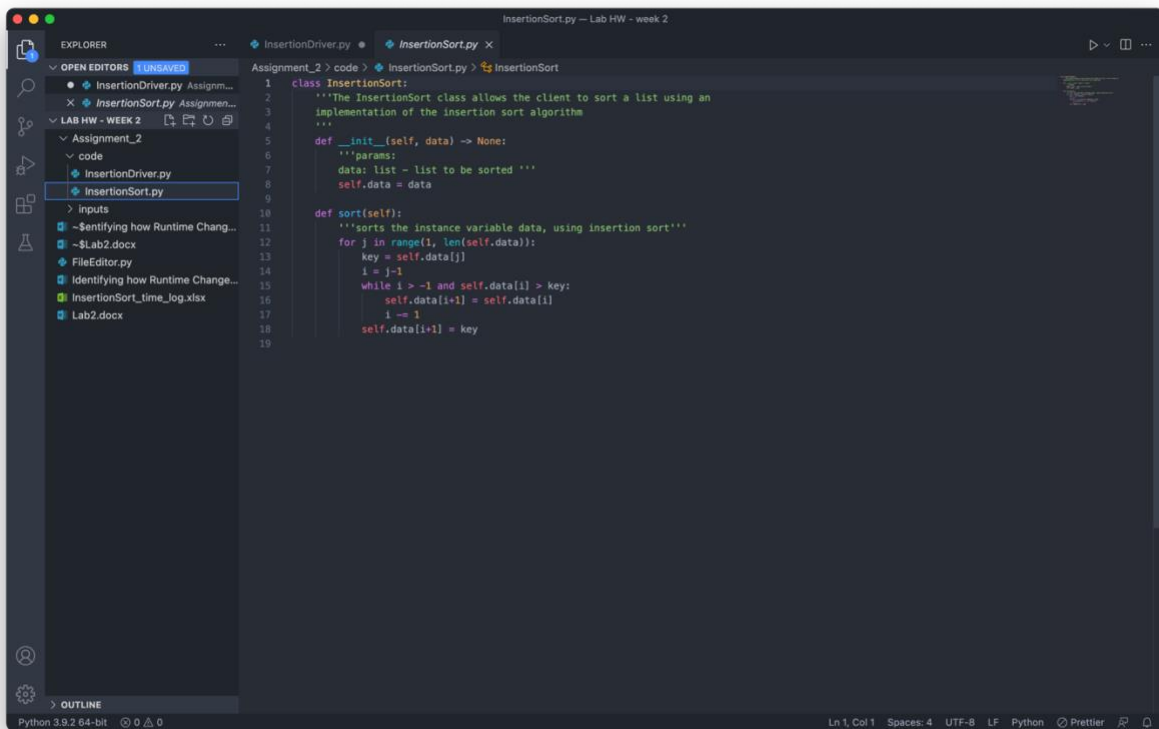


```
1 from InsertionSort import InsertionSort
2 import time
3
4 class InsertionDriver:
5     """InsertionDriver implements the InsertionSort class. It provides methods
6     to test the methods in the InsertionSort class, to calculate runtime for
7     sorting a list, and to parse a file of integers.
8     """
9
10    def __init__(self) -> None:
11        self.data = list[int]
12
13    def load_file(self, filepath, delimiter=' '):
14        """load_file parses a file, and stores its content in the variable data.
15        param:
16            filepath: str, the filepath for the file to parse. /filename would be
17            searching in the same directory where InsertionDriver.py is stored
18
19            delimiter: str (default ' ') the string to split the file with.
20        """
21        # opens the file for reading, then reads the file and stores its content
22        # into content.
23        num_file = open(filepath, 'r')
24        content = num_file.read()
25        num_file.close()
26
27        # splits the content along the passed delimiter and reassigns to content.
28        # Converts each str(int) inside content into an int.
29        content = content.split(delimiter)
30        content = [int(num) for num in content]
31
32        # stores the parsed data into the data var
33        self.data = content
34
35    def time_method(self):
36        """calculates the time to sort each of the required files
37        """
38        file_names = ['input_100.txt', 'input_1000.txt', 'input_5000.txt',
39                     'input_10000.txt', 'input_50000.txt', 'input_100000.txt',
40                     'input_500000.txt']
41        for file in file_names:
42            # loads in the data from the current file using
43            self.load_file('./inputs/' + file)
```



```
44            # passes data on to the sorting class
45            cls = InsertionSort(self.data)
46
47            # stores the start time, sorts, and then compares the start to end
48            # time. That time is saved in to elapsed, and printed to console
49            start = time.time()
50            cls.sort()
51            end = time.time()
52            elapsed = end - start
53            print(file, "sorting time is", elapsed)
54
55    def test(self, lst: list[int]):
56        """tests the sort method with the list passed through the parameter.
57        param:
58            lst: list[int] list to be sorted
59        """
60        # prints the list once it has been sorted
61        cls = InsertionSort(lst)
62        cls.sort()
63        return cls.data
64
65    if __name__ == '__main__':
66        driver = InsertionDriver()
67
68        ## Timing the sort Method - uncomment the following method call to time the sorting method.
69        driver.time_method()
70
71        ## Testing Method - uncomment the following test cases for them to run.
72        print(driver.test([10, 4, 6, 3, 2, 9, 16, 0, 3, -1]))
73        print(driver.test([4, 3, 3, 3, 2, -2]))
74        print(driver.test([-3, -103, -5, -2, -10, -44, -31]))
75        print(driver.test([10]))
76        print(driver.test([10, 9, 8, 7, 6, 5, 4, 3, 2, 1]))
77        print(driver.test([]))
```

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:

