Computing Quick Sort using Python

Homework #5

By

Mario Pendleton

CS 303 [Algorithms and Data Structures](https://uab.instructure.com/courses/1507655)

September 27, 2019

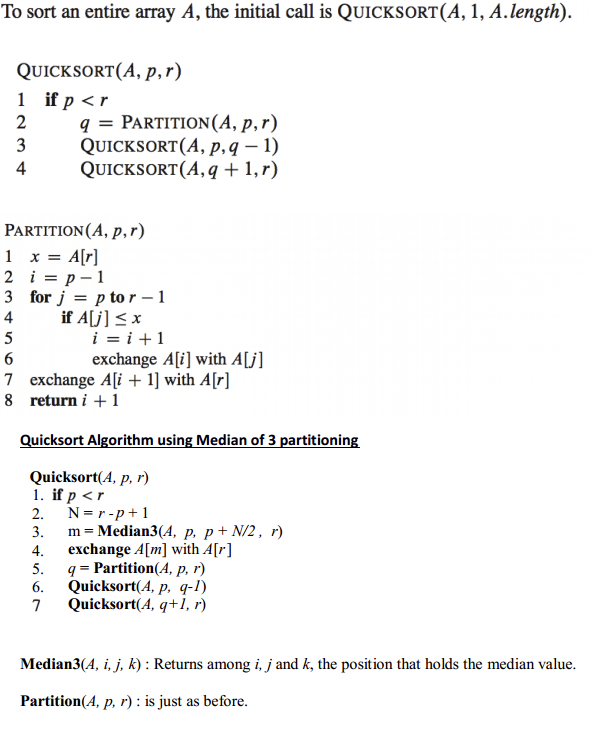
### Problem Specification

Implement a method that will sort a given array using the quick sort algorithm. Write a driver program to test the quick sort algorithms implemented. Read the input file “input\_100.txt” for the input numbers and store them in an array. Sort this array using quick sort. Test the program for the different size input files and compare the performance of quick sort to that of merge, max heap, and insertion sort. Record the runtime quick sort on various sized arrays by using the provided files.  Comment on how the execution time of quick sort varies with size of the input array. Use a table or plot to summarize the results and document your observations and explanations in the report.

1. Implement a method to sort a given array using the quick sort algorithm.
2. Implement quick sort using median of 3 partitioning
3. Write a driver program to test the heap sort algorithm for the arrays of varying lengths provided  in Canvas. Use input\_100.txt file to test your code initially.
4. Compare the performance of insertion sort, merge sort, heap sort, and quick sort.

### Program Design

This program requires an array of data that will be sorted using the quick sort method using median of three. The method was designed in python after the pseudo below.



The following steps were required to develop this program:

1. Re-write the recursive quick sort pseudo code to a python methods QuickSort(x,l,r); partition(x,l,r); QSort(x,l,r) and median3(x,i,j,k).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case | Input Values | Expected Output | Actual Output | Insertion Sort Time | Merge Sort Time | Heap Sort Time | Quick Sort |
| (a) | ["apple", "cherry", "mango", "banana", "dragon fruit"] | ['apple', 'banana', 'cherry', 'dragon fruit', 'mango'] | ['apple', 'banana', 'cherry', 'dragon fruit', 'mango'] | 0.000656 | 0.000695 | 0.0006519 | 0.000534 |
| (b) | [123,"apple",5,6,"green"] | not supported between instances of 'int' and 'str' | not supported between instances of 'int' and 'str' | N/A | N/A | N/A | N/A |
| (c) | ["123","apple","5","53","5a","6","green"] | ['123', '5', '53', '5a', '6', 'apple', 'green'] | ['123', '5', '53', '5a', '6', 'apple', 'green'] | 0.0006379 | 0.000664 | 0.000627 | 0.000558 |
| (d) | [10.1,10.9,9.3,7.4,6.49,2.0,1.999,0.01,5.999] | [0.01, 1.999, 2.0, 5.999, 6.49, 7.4, 9.3, 10.1, 10.9] | [0.01, 1.999, 2.0, 5.999, 6.49, 7.4, 9.3, 10.1, 10.9] | 0.000670 | 0.000684 | 0.0006879 | 0.000568 |
| (e) | sampleList(n) n = 5 | Random array of 8 integers  [8, 7, 3, 5, 8, 3, 2, 5] | Random array of 8 integers sorted  [2, 3, 3, 5, 5, 7, 8, 8] | 0.000648 | 0.000653 | 0.0006719 | 0.000520 |
| (f) | load(path) path = input\_100.txt  [4, 50, 34, 40, 22, 54, 94, 3, 94, 38, 8, 95, 0, 36, 54, 54, 81, 30, 24, 98, 12, 25, 43, 0, 52, 52, 88, 22, 83, 70, 96, 57, 89, 53, 13, 64, 74, 18, 37, 86, 73, 76, 15, 1, 93, 69, 77, 81, 29, 78, 14, 45, 67, 1, 0, 41, 60, 63, 74, 16, 75, 75, 36, 49, 68, 5, 67, 29, 15, 84, 47, 77, 40, 80, 24, 61, 25, 7, 85, 83, 81, 47, 10, 39, 22, 72, 87, 64, 92, 27, 50, 69, 12, 54, 23, 85, 38, 75, 73, 94] | [0, 0, 0, 1, 1, 3, 4, 5, 7, 8, 10, 12, 12, 13, 14, 15, 15, 16, 18, 22, 22, 22, 23, 24, 24, 25, 25, 27, 29, 29, 30, 34, 36, 36, 37, 38, 38, 39, 40, 40, 41, 43, 45, 47, 47, 49, 50, 50, 52, 52, 53, 54, 54, 54, 54, 57, 60, 61, 63, 64, 64, 67, 67, 68, 69, 69, 70, 72, 73, 73, 74, 74, 75, 75, 75, 76, 77, 77, 78, 80, 81, 81, 81, 83, 83, 84, 85, 85, 86, 87, 88, 89, 92, 93, 94, 94, 94, 95, 96, 98] | [0, 0, 0, 1, 1, 3, 4, 5, 7, 8, 10, 12, 12, 13, 14, 15, 15, 16, 18, 22, 22, 22, 23, 24, 24, 25, 25, 27, 29, 29, 30, 34, 36, 36, 37, 38, 38, 39, 40, 40, 41, 43, 45, 47, 47, 49, 50, 50, 52, 52, 53, 54, 54, 54, 54, 57, 60, 61, 63, 64, 64, 67, 67, 68, 69, 69, 70, 72, 73, 73, 74, 74, 75, 75, 75, 76, 77, 77, 78, 80, 81, 81, 81, 83, 83, 84, 85, 85, 86, 87, 88, 89, 92, 93, 94, 94, 94, 95, 96, 98] | 0.001495 | 0.001070 | 0.001194 | 0.001139 |
| (f) | load(path) path = input\_1000.txt | (Data) Insertion sort was successful. No errors found.  True | (Data) Insertion sort was successful. No errors found.  True | 0.099101 | 0.009914 | 0.009672 | 0.008156 |
| (f) | load(path) path = input\_5000.txt | (Data) Insertion sort was successful. No errors found.  True | (Data) Insertion sort was successful. No errors found.  True | 2.29641 | 0.132098 | 0.055474 | 0.035658 |
| (f) | load(path) path = input\_10000.txt | (Data) Insertion sort was successful. No errors found.  True | (Data) Insertion sort was successful. No errors found.  True | 9.41182 | 0.476336 | 0.11231 | 0.090622 |
| (f) | load(path) path = input\_50000.txt | (Data) Insertion sort was successful. No errors found.  True | (Data) Insertion sort was successful. No errors found.  True | 235.287068 | 12.253934 | 0.63467 | 0.45541 |
| (f) | load(path) path = input\_100000.txt | (Data) Insertion sort was successful. No errors found.  True | (Data) Insertion sort was successful. No errors found.  True | 904.434601 | 50.99572 | 1.356058 | 0.912404 |
| (f) | load(path) path = input\_500000.txt | (Data) Insertion sort was successful. No errors found.  True | (Data) Insertion sort was successful. No errors found.  True | 22824.3855 | 3732.64041 | 7.977586 | 5.230294 |
| (g) | [] | [] | [] | 0.000651 | 0.000641 | 0.000646 | 0.000500 |
| (h) | [13] | [13] | [13] | 0.000657 | 0.000615 | 0.0006519 | 0.000497 |
| (i) | [“red”] | [‘red’] | [‘red’] | 0.000670 | 0.000628 | 0.000642 | 0.000496 |
| (j) | [-245,245,-1,1,0,-45,45,2,-2,3,-3] | [-245, -45, -3, -2, -1, 0, 1, 2, 3, 45, 245] | [-245, -45, -3, -2, -1, 0, 1, 2, 3, 45, 245] | 0.000681 | 0.000662 | 0.000674 | 0.000540 |

1. Use the a method to read the following txt files and covert them into arrays to be sorted

* input\_100.txt
* input\_1000.txt
* input\_5000.txt
* input\_10000.txt
* input\_50000.txt
* input\_100000.txt
* input\_500000.txt

The following methods were defined within the lab5.py:

1. QuickSort(x,l,r)

A method that calls for a partition then splits a given array in two and calls them to be sorted using the QSort(x,l,r) method.

1. partition(x,l,r)

A method places a pivot element in it proper place and moves elements greater to the right and elements low to the left.

1. QSort(x,l,r)

A recursive method used to quick sort elements of an array using median of three.

1. median3(x,i,j,k).

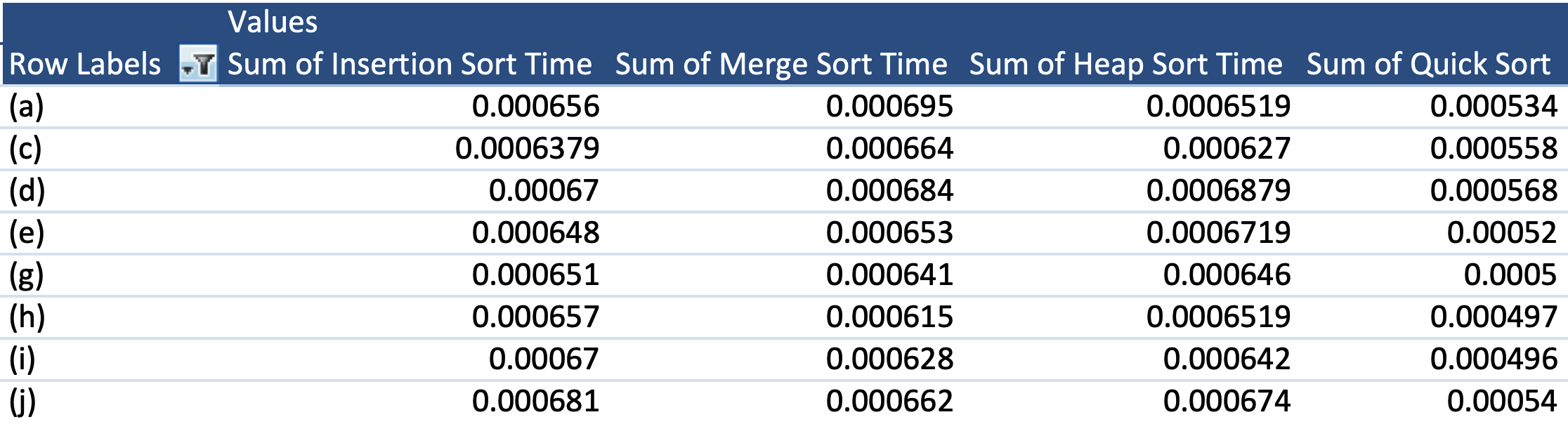
A method used to find the median of three elements of an array and returns the index.

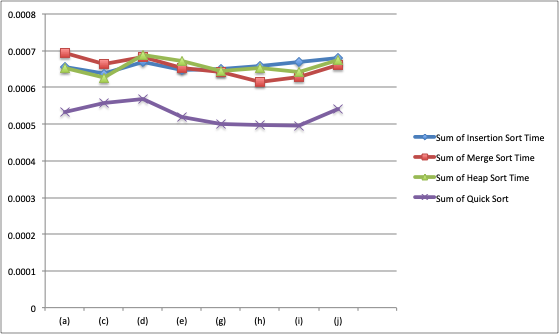
### Testing Plan

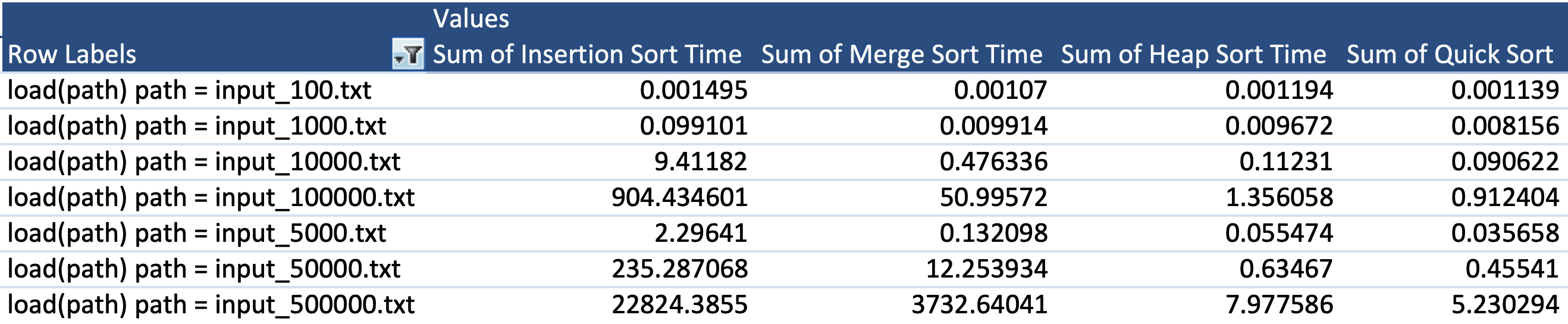
Sample string inputs were selected to see the program could sort (a) none integer values, (b) string and integer values, (c) string integers with strings, (d) floats, (e) random array of integers of 2^n in size, (f) values loaded from the txt files, (g) empty array, (h) single integer, (i) single string, and (j) list of positive and negative integers.

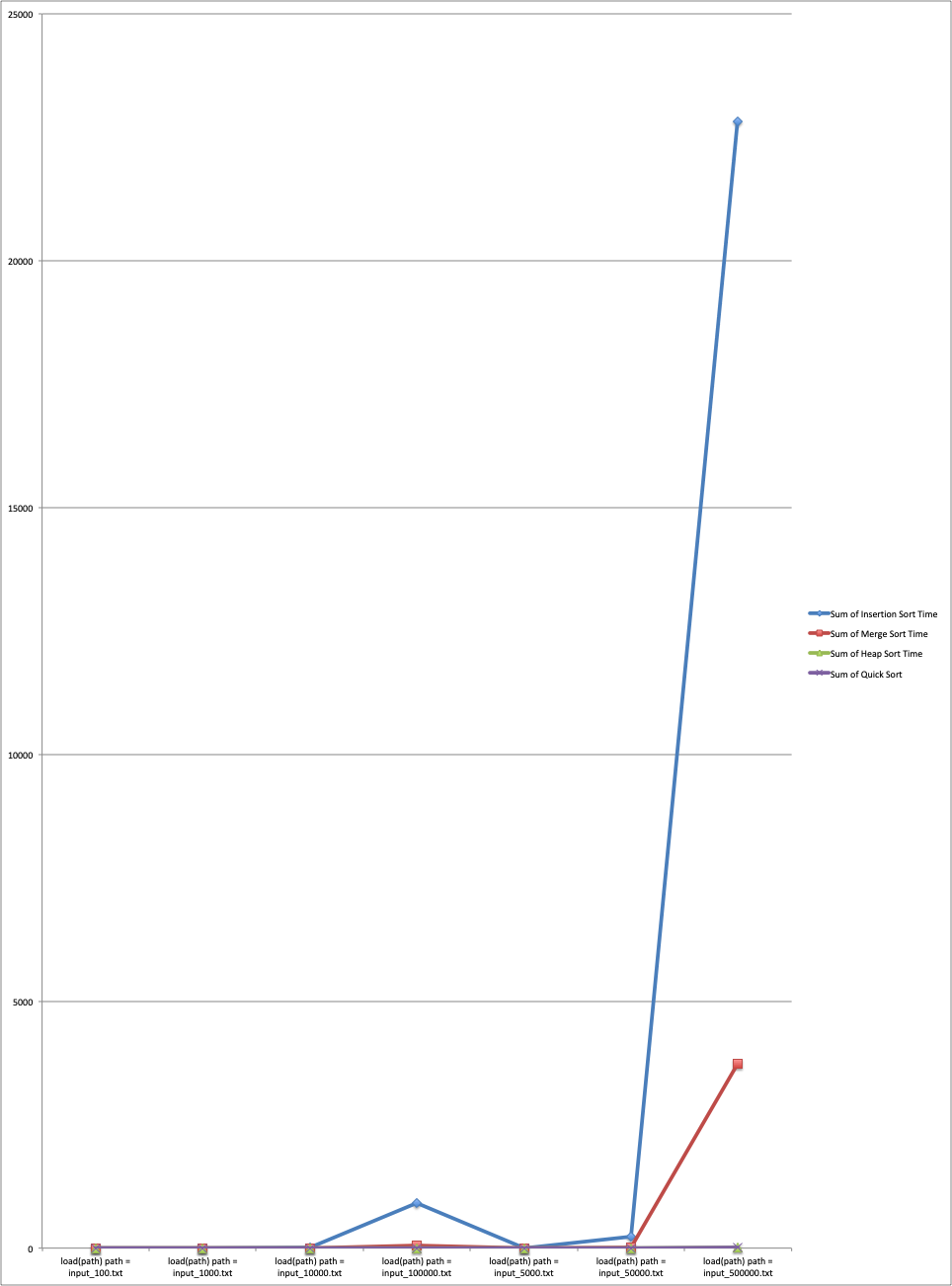
### Test Cases

The test cases are shown in the table below using a MacBook Pro 16GB, 8 core 2.3 GHz Intel Core i9:









### Analysis and Conclusions

Quick sort was successfully computed for all listed test cases. Test case (b) failed as expected. Testing confirmed that the algorithm was able to properly sort the known test cases. The quick sort algorithm a using median of 3 shows faster than insertion, merge and heap in cases where data sizes are small. Quick sort proves to be much faster than insertion sort and merge sort in cases where data sizes are larger. Heap sort however, proves to be slower than quick sort but able to keep up its massive speed.

### References

Textbook, python.org, and examples provided in the assignment.

**Screen Shot**

