Computing Max and Min sort using Python

Homework #6

By

Mario Pendleton

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

October 5, 2019

### Problem Specification

Implement a method that will sort a given array using the max and min sort algorithm of my own design. Write a driver program to test the max and min sort algorithms implemented. Read the input file “input\_100.txt” for the input numbers and store them in an array. Sort this array using max and min sort. Test the program for the different size input files and compare the performance of max and min sort to that of, quick, 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 max and min 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.

**Problem**

1. Consider an algorithm that sorts an array of n elements by finding the smallest and largest elements and then exchanges those elements with the elements in the first and last positions in the array. Then the size of the array is reduced by two elements after excluding the two elements that are already in the proper positions, and the process is repeated on the remaining part of the array until the entire array is sorted. Write a code to implement the above algorithm. Write a driver program to show the novel sorting algorithm works correctly.

2. Consider an application that logs transactions. The log includes the location where a transaction originated and the time the transaction occurred and this information is stored such that they are ordered by the time of the transaction (see sample input below). You are required to sort this log by location while preserving the order of the time field (see sample output below). Implement an algorithm of your choice to sort this array based on the location while preserving the order of the time field. Test your algorithm with the input file provided in Canvas named “NovelSortInput.txt”. Explain in your report why the sorting algorithm you chose is the best for the job.

### Program Design

This program requires an array of data that will be sorted using the max and min method. The method was designed in python. The speed of this algorithm is O(n) but it is greatly increased by shortening the array from the right. I will demonstrate this in my data below. The pseudo code below should explain my thought process while designing.

mySort (A,p,r)

1. While p <= r:
2. MaxMinSort(A,p,r)
3. p = p+1
4. r = r-1 (Here I reduce the distance i has to travel. In doing so it cuts the speed in half. I highly recommend leaving it for max speed.)

MaxMinSort (A,p,q)

1. i = p
2. max = A[p]
3. max position = A[p]
4. min = A[p]
5. min position = A[p]
6. While i < = r:
7. If A[i] >= max
8. max = A[i]
9. max position = i
10. If A[i] <= min:
11. min = A[i]
12. min position = i
13. i = i+1
14. Swap A[p] with A[min position]
15. Swap A[r] with A[max position]

The following steps were required to develop this program:

1. Write a recursive quick sort pseudo code to a python methods mySort(x,l,r); MaxMinSort(x,l,r); and loadLine(path).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case | Input Values | Expected Output | Actual Output | Insertion Sort Time | Merge Sort Time | Heap Sort Time | Quick Sort | Max/Min Sort | Max/Min Sort with (r = r-1) |
| (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 | 0.000519 | 0.000429 |
| (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 | 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 | 0.000553 | 0.000522 |
| (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 | 0.000557 | 0.0005378 |
| (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 | 0.000536 | 0.000516 |
| (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 | 0.001739 | 0.001138 |
| (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 | 0.1037 | 0.05571 |
| (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 | 2.359206 | 1.26060 |
| (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 | 9.4702 | 4.6509 |
| (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 | 242.9748 | 118.89138 |
| (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 | 942.10293 | 470.74297 |
| (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 | 25124.9800 | 12074.52009 |
| (g) | [] | [] | [] | 0.000651 | 0.000641 | 0.000646 | 0.000500 | 0.000508 | 0.000511 |
| (h) | [13] | [13] | [13] | 0.000657 | 0.000615 | 0.0006519 | 0.000497 | 0.000505 | 0.000505 |
| (i) | [“red”] | [‘red’] | [‘red’] | 0.000670 | 0.000628 | 0.000642 | 0.000496 | 0.000509 | 0.000509 |
| (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 | 0.000544 | 0.000528 |

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
* NovelSortInput.txt

The following methods were defined within the lab5.py:

mySort (x,l,r)

A recursive method designed to break down an array by repeatedly removing first and last element.

MaxMinSort (x,l,r)

A method that finds the maximum and minimum values of an array then swaps the minimum with the first element and the maximum with the last element.

loadLine(path)

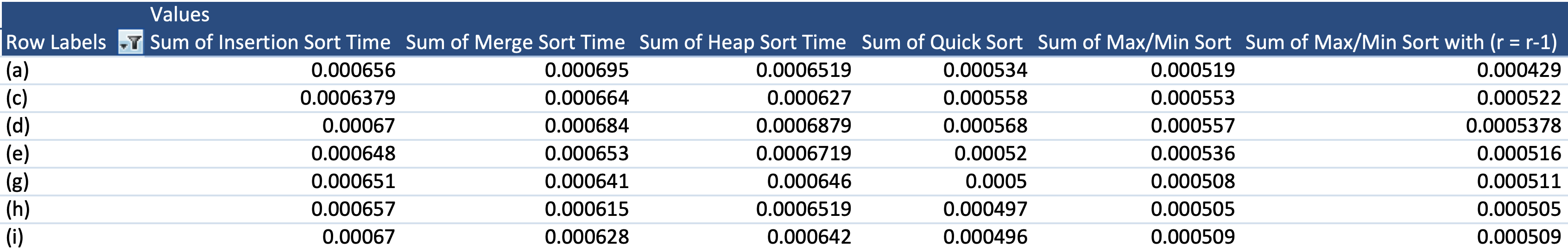
A method that opens a file and reads each line one by one then stores the values of that line into separate arrays.

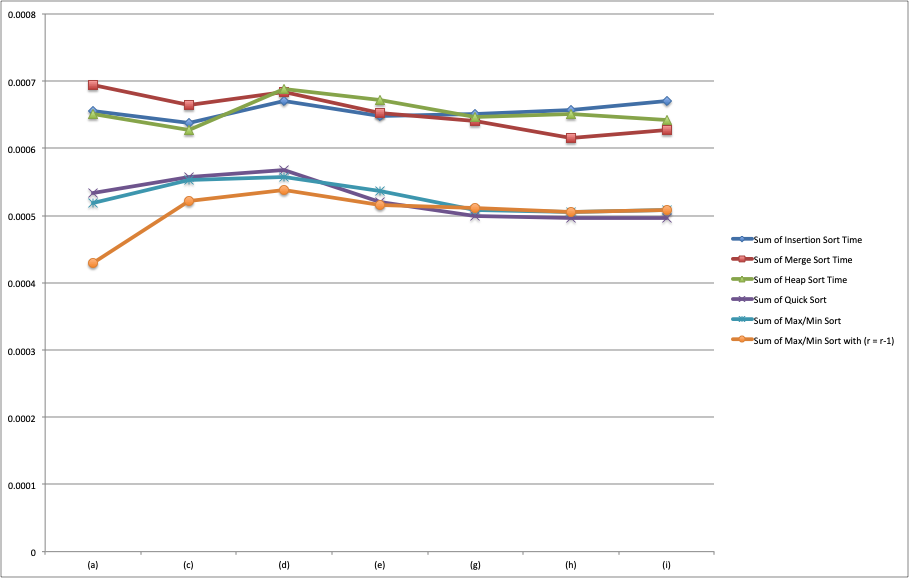
### Testing Plan

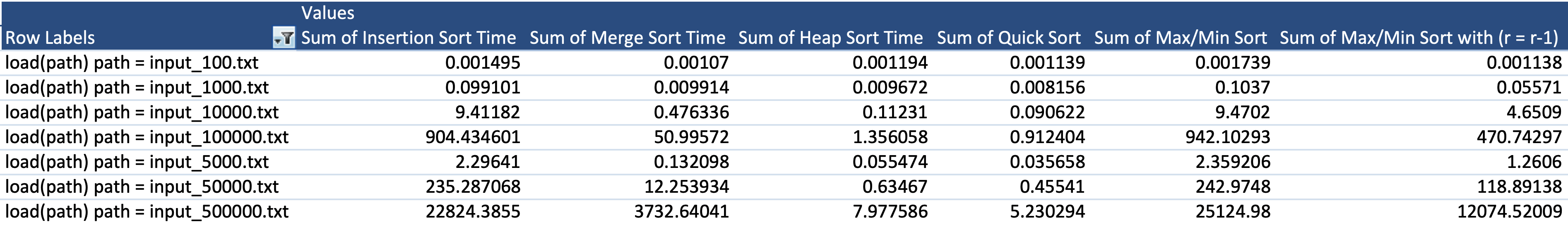
Sample string inputs were selected to see the program could sort (a) none integer values, (b) string and integer values, (c) string integers with strings, (d) floats, (e) random array of integers of 2^n in size, (f) values loaded from the txt files, (g) empty array, (h) single integer, (i) single string,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

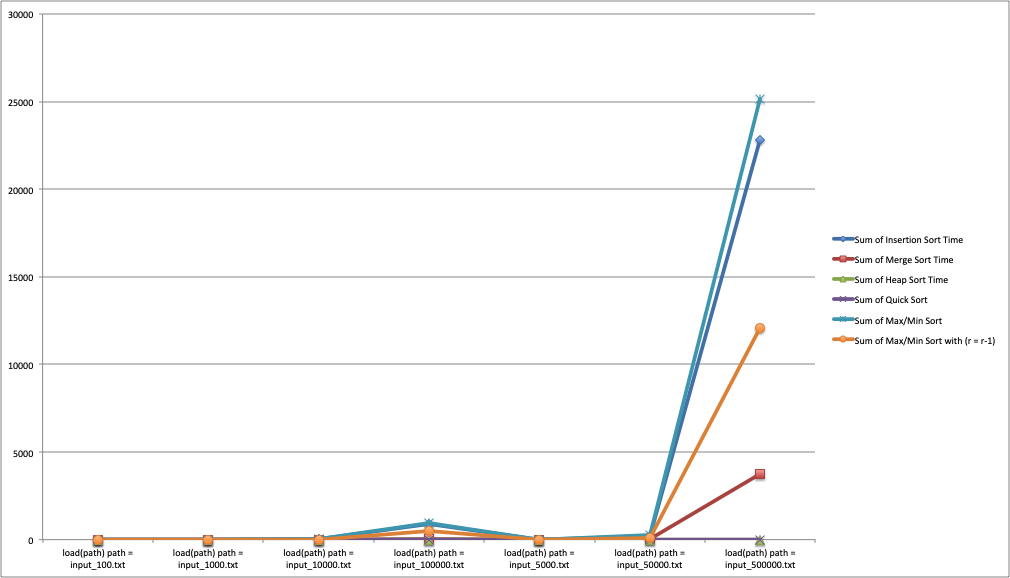
### Test Cases

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

Max/Min 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 Max/Min sort algorithm shows faster than insertion, merge, heap, and quick in cases where data sizes are small. My design of Max/Min sort proves faster than insertion sort when optimized sort but slower than merge, heap, and quick sort in cases where data sizes are larger.

### References

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

**Screen Shots**

