Computing reverse sorting algorithms using Python

Homework #7

By

Mario Pendleton

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

October 13, 2019

### Problem Specification

Implement basic selection sort and bubble sort algorithms and convert all the algorithms you have implemented sort in the reverse order. Compare the performance of the sorting algorithms.



**Problem**

1. Implement a method to sort a given array using basic selection sort algorithm.
2. Implement a method to sort a given array using basic bubble sort algorithm.
3. Compare the performance of the selection sort algorithm with 3 cases of input files: sorted, reversed sorted, and random. These files are provided in Canvas in the Input Files folder.
4. Compare the performance of the bubble sort algorithm with 3 cases of input files: sorted, reversed sorted, and random.
5. Modify all algorithms you have implemented so far to sort in the reverse order. Run the modified algorithms using the given input files. Compare the performance of the algorithms.

### Program Design

This program requires an array of data that will be sorted using the reverse sort algorithms. The method bubble sort and selection sort algorithms were design after the pseudo code below.

**Pseudo code for Bubble Sort**

for I = 1 to A.length - 1

for J = A.length to i+1

if A[J] < A[J – 1]

exchange A[j] with A[j-1]

**Pseudo code for Selection Sort**

for I = 1 to A.length – 1

min = i

for J = I+1 to A.length

if A[J] < A[min]

min=j

end if

end for

if i!=min

swap A[min] and A[i]

end if

end for

The following steps were required to develop this program:

1. Write bubble sort methods using the pseudo code in python.

bubbleSort(x)

recBubbleSort(x)

1. Write selection sort methods using the pseudo code in python.

selectionSort(x)

revSelectionSort(x)

1. Modify all sorting methods to sort in reverse.

insertionSort(x)

mergeSort(x,y,l,r)

merge(x,y,l,m,r)

heapSort(x)

maxHeapify(x,heapsize,i)

quicksort(x,l,r)

partition(x,l,r)

qSort(x,l,r)

median3(x,i,j,k)

mySort(x,l,r)

MaxMinSort(x,l,r)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case | Input Values | Output Values | Reverse Insertion Sort | Reverse Merge Sort | Reverse Heap Sort | Reverse Quick Sort | Reverse Max/Min Sort | Bubble Sort | Reverse Bubble Sort | Selection Sort | Reverse Selection |
| (a) | ["apple", "cherry", "mango", "banana", "dragon fruit"] | [“mango”,”dragon” “fruit”,”cherry”,”banana”,”apple”] | 0.000585 | 0.000530 | 0.000532 | 0.000530 | 0.000528 | 0.000528 | 0.000530 | 0.000527 | 0.000524 |
| (b) | [123,"apple",5,6,"green"] | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A |
| (c) | ["123","apple","5","53","5a","6","green"] | ['green', 'apple', '6', '5a', '53', '5', '123'] | 0.000538 | 0.000532 | 0.000533 | 0.000537 | 0.000530 | 0.000528 | 0.000532 | 0.000528 | 0.000529 |
| (d) | [10.1,10.9,9.3,7.4,6.49,2.0,1.999,0.01,5.999] | [10.9, 10.1, 9.3, 7.4, 6.49, 5.999, 2.0, 1.999, 0.01] | 0.000528 | 0.00550 | 0.000566 | 0.000579 | 0.000553 | 0.000545 | 0.000543 | 0.000550 | 0.000540 |
| (e) | sampleList(n) n = 5 | [8, 7, 6, 5, 4, 3, 3, 3] | 0.000524 | 0.000531 | 0.000539 | 0.000547 | 0.000547 | 0.000544 | 0.000525 | 0.000533 | 0.000531 |
| (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] | [98, 96, 95, 94, 94, 94, 93, 92, 89, 88, 87, 86, 85, 85, 84, 83, 83, 81, 81, 81, 80, 78, 77, 77, 76, 75, 75, 75, 74, 74, 73, 73, 72, 70, 69, 69, 68, 67, 67, 64, 64, 63, 61, 60, 57, 54, 54, 54, 54, 53, 52, 52, 50, 50, 49, 47, 47, 45, 43, 41, 40, 40, 39, 38, 38, 37, 36, 36, 34, 30, 29, 29, 27, 25, 25, 24, 24, 23, 22, 22, 22, 18, 16, 15, 15, 14, 13, 12, 12, 10, 8, 7, 5, 4, 3, 1, 1, 0, 0, 0] | 0.001538 | 0.000966 | 0.001058 | 0.000885 | 0.001138 | 0.001829 | 0.001893 | 0.001195 | 0.001183 |
| (f) | load(path) path = input\_1000.txt | Reversed | 0.097623 | 0.009966 | 0.009718 | 0.006963 | 0.056257 | 0.134096 | 0.136158 | 0.063882 | 0.062918 |
| (f) | load(path) path = input\_5000.txt | Reversed | 2.384816 | 0.124562 | 0.055692 | 0.045044 | 1.230478 | 3.41982 | 3.360828 | 1.427476 | 1.414164 |
| (f) | load(path) path = input\_10000.txt | Reversed | 9.721682 | 0.516936 | 0.119687 | 0.06129 | 4.865688 | 13.572276 | 13.946078 | 5.646511 | 5.605808 |
| (f) | load(path) path = input\_50000.txt | Reversed | 243.593738 | 13.40629 | 0.60308 | 0.344831 | 122.99440 | 357.20187 | 356.97511 | 147.98572 | 151.75062 |
| (f) | load(path) path = Input\_Sorted.txt | Reversed | 0.196344 | 0.009527 | 0.009141 | 0.006594 | 0.077788 | 0.085752 | 0.201474 | 0.065968 | 0.071715 |
| (f) | load(path) path = Input\_ReversedSortedtxt | Reversed | 0.001462 | 0.009658 | 0.0104 | 0.00811 | 0.078378 | 0.203752 | 0.085608 | 0.069843 | 0.067118 |
| (f) | Input\_Random.txt | Reversed | 0.104400 | 0.009569 | 0.009942 | 0.006173 | 0.058006 | 0.14611 | 0.144402 | 0.069242 | 0.067242 |

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
* Input\_Sorted.txt
* Input\_Random.txt
* Input\_ReversedSorted.txt

The following methods were defined within the lab7.py:

bubbleSort (x,l,r)

A method designed to swap the adjacent elements if they are in wrong order.

selectionSort(x,l,r)

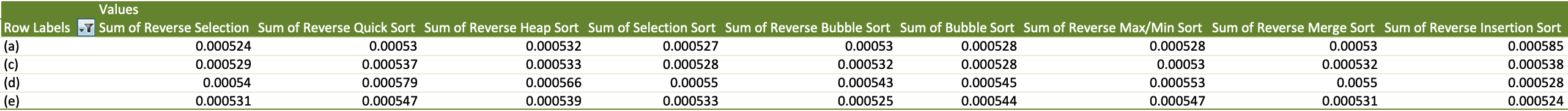
A method that finds the minimum value of an array and places it into a sorted array.

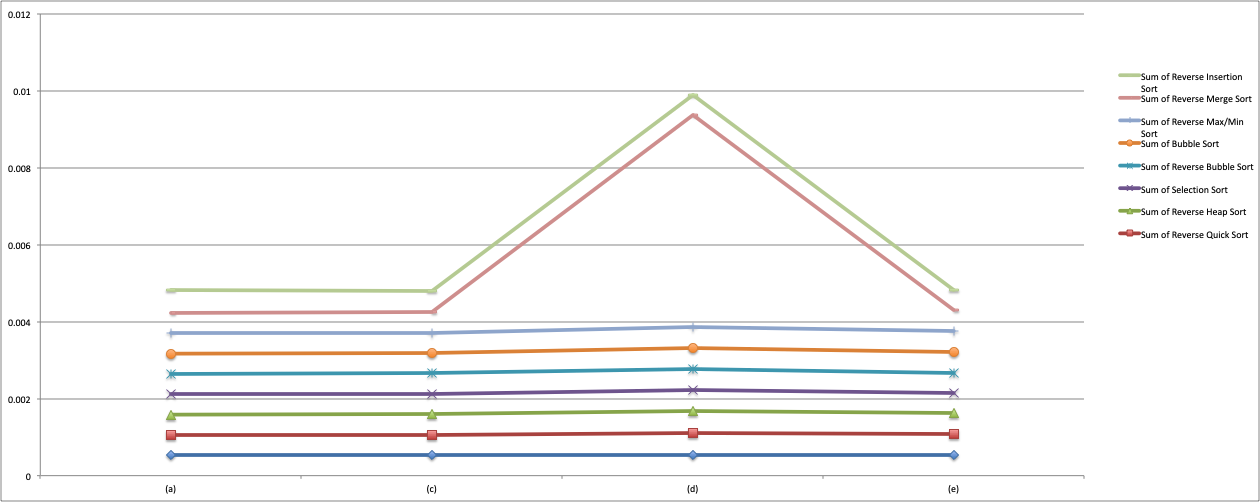
### Testing Plan

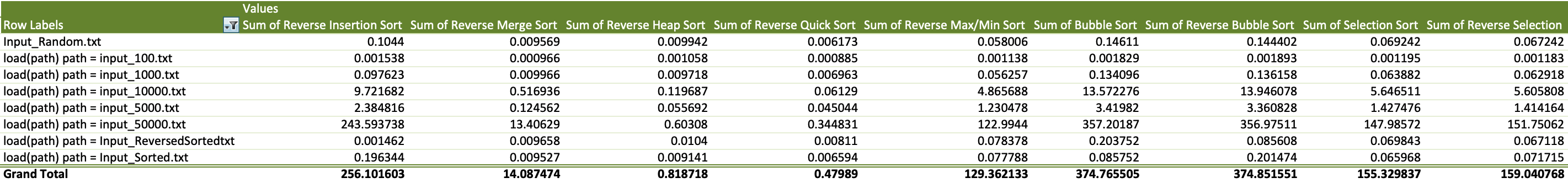
Sample string inputs were selected to see the program could sort (a) none integer values, (b) string and integer values, (c) string integers with strings, (d) floats, (e) random array of integers of 2^n in size, and (f) values loaded from the txt files.

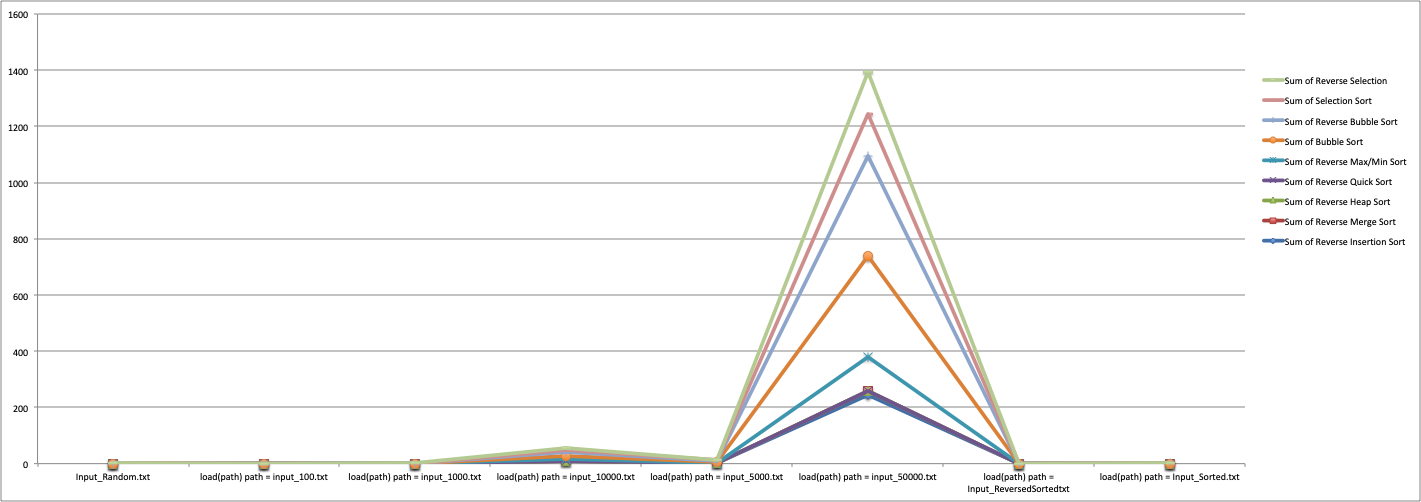
### Test Cases

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

### References

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

**Screen Shots**

