Computing Max Heap Sort using Python

Homework #4

By

Mario Pendleton

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

September 22, 2019

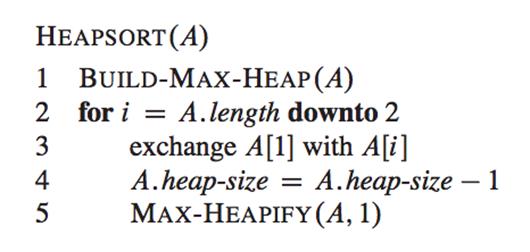
### Problem Specification

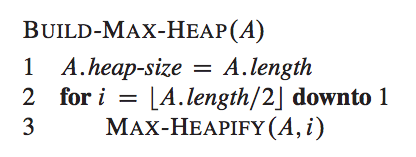
Implement a method that will heap sort a given array using the max heap sort algorithm. Write a driver program to test the max heap algorithms implemented. Read the input file “input\_100.txt” for the input numbers and store them in an array. Sort this array using  max heap sort. Test the program for the different size input files and compare the performance of max heap sort to that of merge and insertion sort. Record the runtime for max heap sort on various sized arrays by using the provided files.  Comment on how the execution time of merge 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.

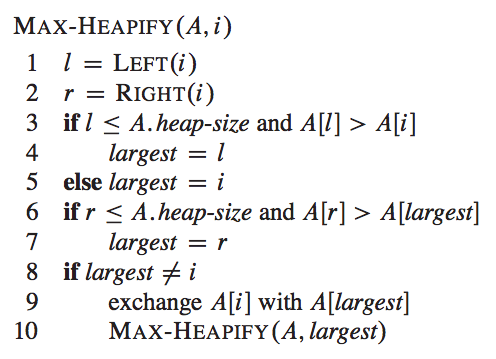
1. Implement a method to sort a given array using the heap sort algorithm. Use the algorithm from  the textbook (see page 2).
2. Write a driver program to test the heap sort algorithm for the arrays of varying lengths provided  in Canvas. Use input\_100.txt file to test your code initially.
3. Compare the execution time of heap sort with insertion sort implemented in Lab-2 and merge sort implemented in Lab-3. Make sure you use the same array to compare the performance. Use a table or plot to summarize the results and document your observations and analysis in the report. Use the following input files only: input\_100.txt, input\_1000.txt, input\_5000.txt, input\_10000.txt, and input\_50000.txt

### Program Design

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







The following steps were required to develop this program:

1. Re-write the recursive max heap sort pseudo code to a python methods (mergeSort(x,y,l,r)) merge(x,y,l,m,r) an.
2. 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 lab2.py:

1. maxHeap (x)

A recursive method that builds a max heap then extracts the indexes one by one calling the maxHeapify (x, heap, i) function to sort them.

1. maxHeapify (x, heap, i)

A recursive method initializes the largest root of an array and calls upon itself to sort the array.

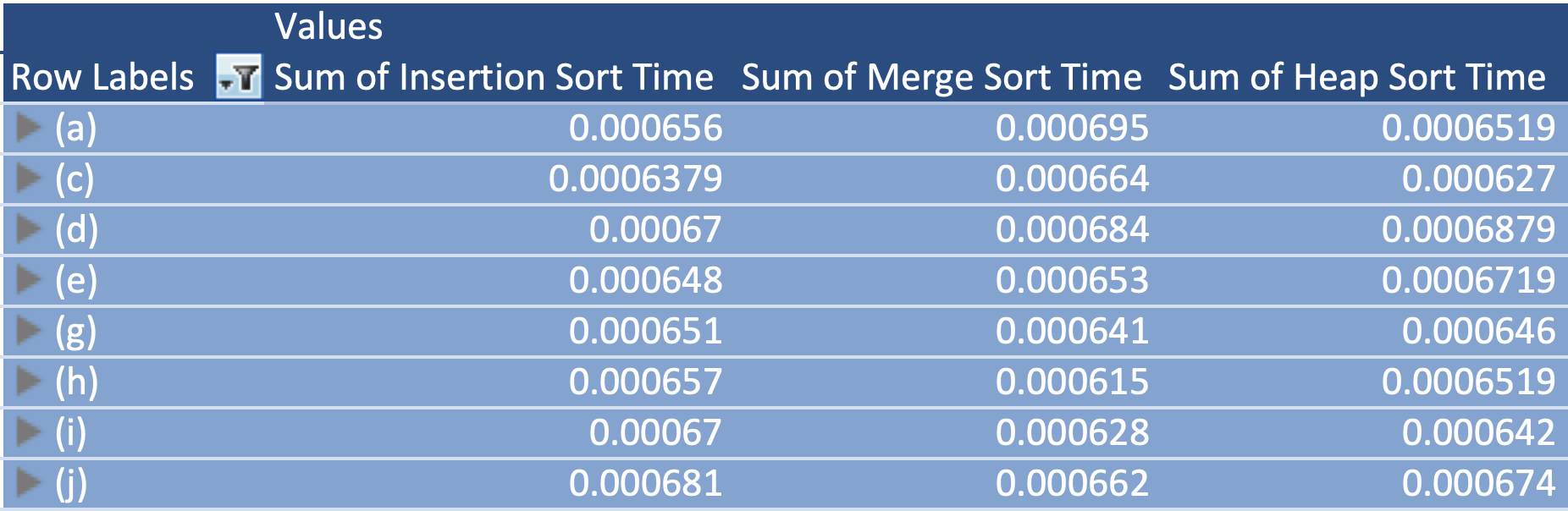
### Testing Plan

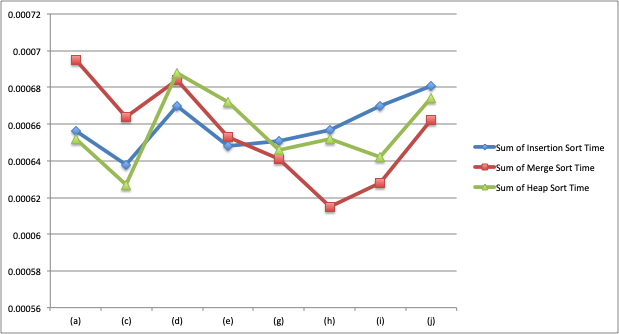
Sample string inputs were selected to see the program could sort (a) none integer values, (b) string and integer values, (c) string integers with strings, (d) floats, (e) random array of integers of 2^n in size, (f) values loaded from the txt files, (g) empty array, (h) single integer, (i) single string,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

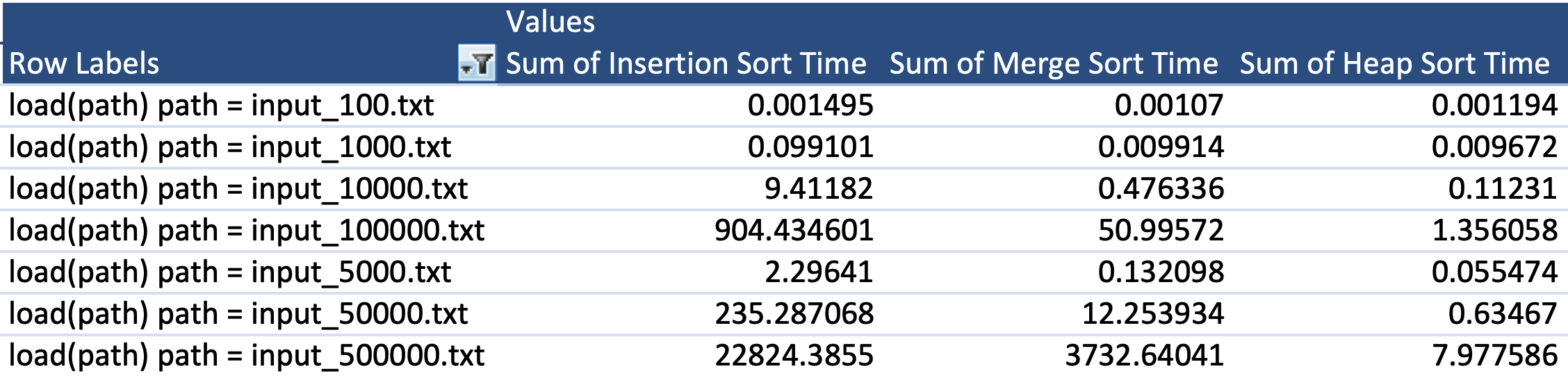
### Test Cases

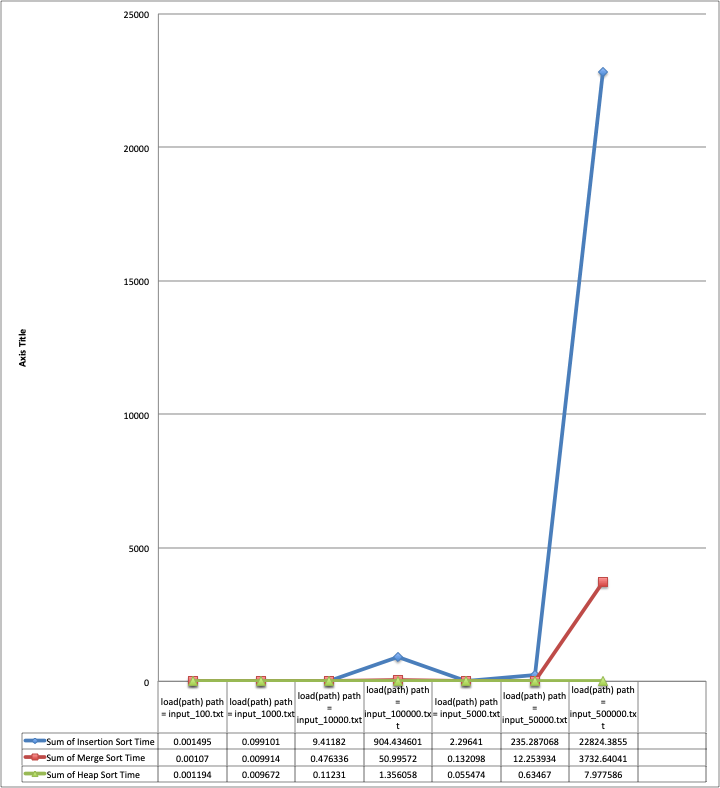
The test cases are shown in the table below using a MacBook Pro 16GB, 8 core 2.3 GHz Intel Core i9:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case | Input Values | Expected Output | Actual Output | Insertion Sort Time | Merge Sort Time | Heap Sort Time |
| (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 |
| (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 |
| (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 |
| (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 |
| (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 |
| (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 |
| (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 |
| (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 |
| (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 |
| (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 |
| (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 |
| (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 |
| (g) | [] | [] | [] | 0.000651 | 0.000641 | 0.000646 |
| (h) | [13] | [13] | [13] | 0.000657 | 0.000615 | 0.0006519 |
| (i) | [“red”] | [‘red’] | [‘red’] | 0.000670 | 0.000628 | 0.000642 |
| (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 |
|  |  |  |  |  |  |  |









### Analysis and Conclusions

Max heap 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 heap sort algorithm using max heap shows that the heap sort is faster than insertion but slightly slower than merge sort in cases where data sizes are small. Heap sorting proves to be much faster than insertion sort and merge sort in cases where data sizes are larger.

### References

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

**Screen Shot**

