Comparing how Runtimes contrast for Merge and Insertion Sort

Lab #3

By

Michael Harden

CS 303 Algorithms and Data Structures

September 14, 2021

1. **Problem Specification**

The goal of this lab was to implement both a Merge Sort and Insertion Sort Algorithm. Then to sort a lists of integers, contained in multiple input files, of varying length, with both sorting algorithms, and calculate the time taken for each. Then, compare how the times change for each algorithm, and identify when to use Insertion Sort verses Merge Sort. Finally to create an algorithm which will combine both algorithms, running the correct algorithm based on the size of the list being sorted

1. **Program Design**

This program requires two classes, Sorts and SortsDriver. The Sorts class contains the implementations for both Merge Sort and Insertion Sort. The SortsDriver 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 both sorting algorithms.

To implement the Insertion Sort algorithm, the pseudocode from lab #2 was implemented

To implement the Merge Sort algorithm, the following process was implemented.

1. Checks that the list contains more than one element.
2. If it contains one or less elements then the method returns. Otherwise, calls the recursive function, \_\_merge\_helper().
3. Merge helper contains three pointers, left, right, and mid.
4. The method calls itself passing on the array, and the left most pointer. The mid pointer is passed as the new right.
5. The method calls itself a second time, passing mid +1 as the new left pointer and right as the right most pointer. This process continues until the base case is met. The base case is where the left and right pointers are equal and only one element remains in the range if left and right.
6. Then the \_\_merge() method is called. This method merges the two half’s of the partition of the list together.
7. This merge is done by comparing the left most items of the two half’s and whichever is smaller is set as the first item. Then the next item in the half which was pulled from last is compared to the previously compared item in the other half. This process in continued until one half is empty. Then the other half is set as the rest of the sorted partition.

To calculate the run times for each required file, the following procedure was followed.

1. 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.
2. Create an instance of the Sorts class and pass the list of parsed integers into the Sorts constructor’s parameter.
3. Store the current time just before sorting, then call the merge\_sort or insertion\_sort method on the Sorts instance. Once the list has been completely sorted, take the difference of the time directly after sorting and the time just before sorting.
4. Log the time take to sort, to the console.
5. Once the merge\_sort method has been timed, the test repeats for the insertion\_sort method
6. Continue to the next iteration.

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

1. \_\_init\_\_(data):

Defines the instance variable, data, and sets its value to the value passed in the data parameter.

1. Insertion\_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.

Merge Sort is broken up into three methods.

1. merge\_sort():

checks that the length of the list is greater than one and sets the initial state for the recursive method.

1. \_\_merge\_helper(arr, left, right, temp\_arr):

Recursive step as described above. The parameter arr is the array to be sorted, left is the left most index in the current partition, right is the right most index in the current partition, temp\_arr is the temporary array used to store the elements while sorting.

1. \_\_merge(arr, left, mid, right, temp\_arr):

Sorting step as described above. The parameters are the same as \_\_merge\_helper. The parameter mid is the middle index

1. optimal\_sort(cutOff):

Combines both Merge and Insertion sort. When the number of elements being sorted is less than the cutOff value, insertion\_sort() is called. Otherwise, merge\_sort() is called.

The following constructor and methods are defined within the SortsDriver class.

1. \_\_init\_\_():

Defines the instance variable data, and sets its type as a list[int]

1. 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.

1. 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.

1. test\_insertion(lst):

Sorts the lists passed in the parameter, lst, with Insertion Sort and returns the sorted list.

1. test\_merge(lst)

Sorts the list passed in to the parameter, lst, with Merge Sort and returns the sorted lsit.

1. Test\_optimal(cutOff, lst):

Sorts the list passed in the parameter, lst. The sorting algorithm is chosen with th cutOff value. If the list size is greater than the value of lst, Merge sort is ran. Otherwise, Insertion Sort is ran.

To parse the files, the open() and read() built in methods are used.

1. **Testing**

To test both sorting algorithms six test cases were used to attempts and break the algorithms 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** | [] | [] |

1. **Timing**

The times to sort the provided files are as follows:

|  |  |  |
| --- | --- | --- |
| **File name** | **Insertion Sort Time** (seconds) | **Merge Sort Time**  (seconds) |
| input\_100.txt | 0.0064421 | 0.000217915 |
| input\_1000.txt | 0.084224425 | 0.003399849 |
| input\_5000.txt | 1.81907511 | 0.019459009 |
| input\_10000.txt | 6.46251702 | 0.041306019 |
| input\_50000.txt | 182.156554 | 0.24689889 |
| input\_100000.txt | 936.355143 | 0.521799088 |
| input\_500000.txt | Could not sort in a reasonable time on my personal computer. | 3.165700197 |

1. **Analysis and Conclusions**

In conclusion, the tests show how merger sort is a far superior sorting algorithm, particularly as the number of elements reaches into the tens of thousands.

There may be a few situations where Insertion Sort is the better algorithm, however. For instance, if the assumption can be made that the data being sorted is almost entirely sorted, Insertion Sort would be better. In this scenario Insertion Sort would draw near to O(n) time while Merge Sort would stay constant at O(n log n)

Another scenario where Insertion Sort may be a better choice is when a very small number of elements need to be sorted. In the testing done for this lab, no such size was found, and Merge Sort was the better algorithm for all required lists.

So, for the optimal sorting algorithm, which combines both Merge and Insertion Sort, Merge Sort would be called every time for all required text files to sort.

The following graph displays how the sorting time grows in relation to a linearly growing input size. It compares the time complexity of both Merge and Insertion Sort

(data collected with provided text files):

1. **Credits**

* For the Merge Sort algorithm the following resources were utilized:

1. The pseudo code in the lab3 document provided on canvas
2. Algoexpert.com video explanation and code reference of merge sort
3. Khanacademy.org explanation on merge sort
4. Programiz.com explanation on merge sort

* For the Insertion Sort algorithm, the pseudo code in the lab2 document.
* Parts of my lab report 2 were also used in this report.
* For the Report the sample report provided on canvas was used as a guide.