CS 430 - HW 1

Project Management Document

ISSUED BY

Team Glazed Donuts

Members

	
Gladys Toledo-Rodriguez, #59 A20419684	Grace Arnold, #6 A20415197

Tasks:

Gladys will work on the Merge Sort algorithm and Grace will do the insertion sort. Both will work on the GUI. We will take Sunday, Feb 9th to finish the majority of the work. You can look at work on Github.
(https://github.com/gtoledorodriguez/Insertion_merge_sort_gui)

Feb 9:

Gladys:

- Programmed and tested the merge sort algorithm created
    - Created sort() and merge() functions
    - Added test case for a defined unordered array
    - Added test case for an unordered array of random numbers
- Added print statements to understand what the algorithm was doing
- Started working on the GUI
    - Added a merge, go, step and reset button

Grace:

- Programmed and tested the insertion sort algorithm
    - Wrote insertElement class with insertElement() function
    - Wrote InsertionSort class with insertionSort() function
    - Added test cases for a small unordered array
    - Added test cases for a small unordered array of random numbers
- Created Graphical User Interface and Runner
    - Started programming SortGUI class and GUIRunner class
    - Made basic frame and buttons
    - Created Session class for methods that the GUI will run

Feb 10:

Gladys

- Worked on the GUI to fix buttons

Grace

- Added the insertion() method to the Session class

Feb 11:

Gladys

- Updated merge sort to print out each step of the algorithm
- Adding the text area to the GUI.
    - Trying to figure out how to change textArea after each button click
- Made go, step, and reset button invisible unless the insertion or merge button is clicked

Grace

- Updated the insertion sort function to print out each step
- Added the insertion function to the GUI
    - Started working on making each step visible in the GUI

Feb 12:

Gladys

- Made the text area scrollable
- Adding an ArrayList that will collect all the steps of the merge sort.
- Information from ArrayList won't appear in the GUI
    - trying to find the error
- Added guiSort() so the actionPerformed calls it and goes through the merge sort
    - Trying to find the error
    - Works when calling the method separately but not when called through the GUI
    - Found error and mergeSort is done
- Cleaned up code and adding more comments

Grace

- Added functionality to the reset button in the GUI
    - Fixed error in Session class, allowing the unsorted array of random numbers to be updated in the GUI
    - Fixed error which made the textArea not scrollable after the reset button was pressed
- Fixed the Insertion sort button to add the steps to the textArea
- Added comments and removed unnecessary code

Analysis:

Insertion Sort:

The general case for Insertion Sort has a complexity of $O(n^2)$.
This is because it has to compare every pair of elements in the array, so it has to run through the array for each element.
If we take c_1 to be the amount of times the algorithm in the first for loop is run, c_2 to be the amount of times the algorithm in the nested for loop is run, and c_3 to be the swapping operation, we come up with this equation:

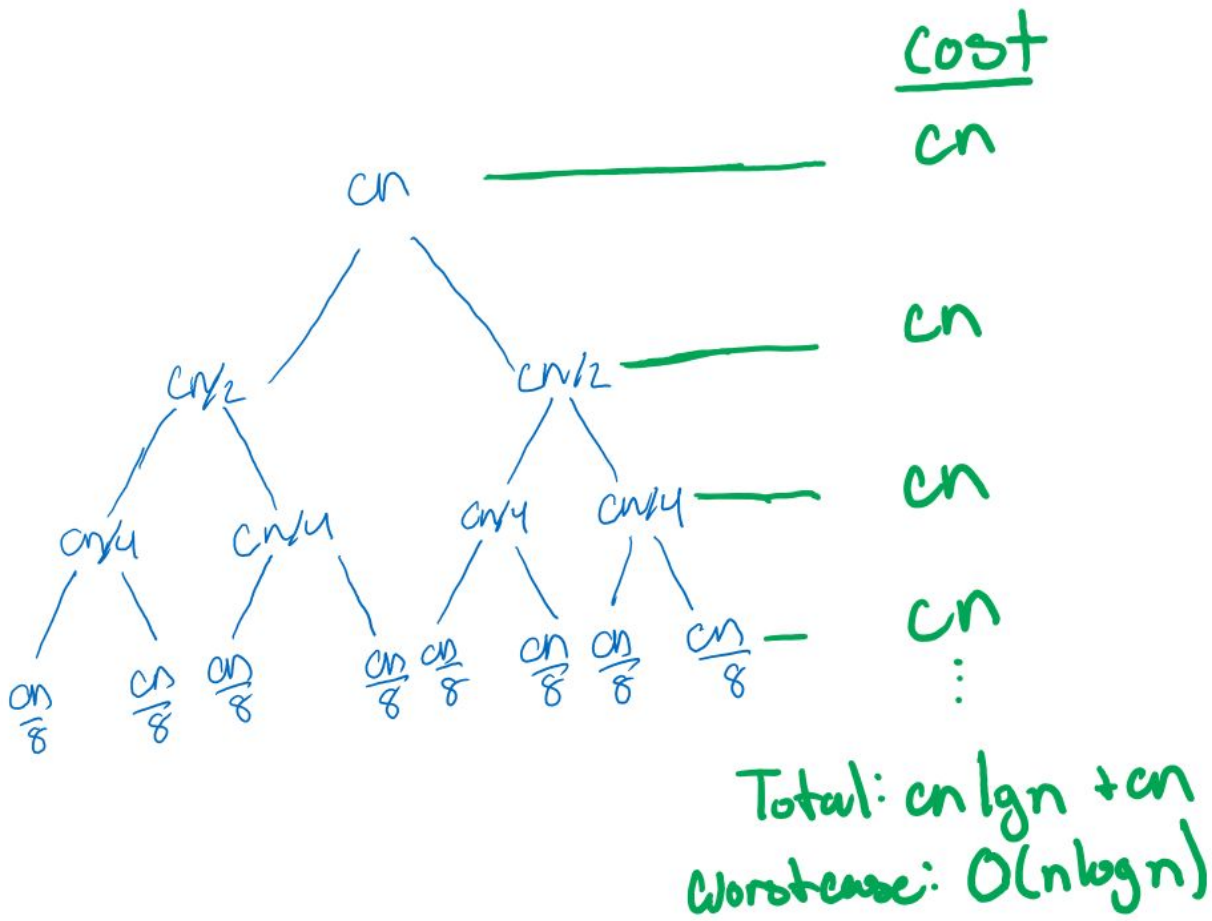
$$(c_1*(n-1))+(c_2*\sum_{j=2}^{n}(j-1)+(c_3*(n-1))$$

Which can then be simplified to:
$d_1n^2 + d_2n + d_3$
If we discard the constant d_3, the low-order term d_2n, and the constant d_1 then we are left with a time complexity of $O(n^2)$.
This is for the average case and the worst case, where the array to be sorted is in decreasing order and thus every pair must be compared.
The best case, however, where the array is already sorted in increasing order, is O(n), because each element only needs to be compared once.

Merge Sort:

$$T(n) = \begin{cases} c & \text{if n is small, or n = 1} \\ 2T(\frac{n}{2}) + D(n) + C(n) & \text{otherwise, or n > 1} \end{cases}$$

D(n) is the cost of splitting the array. Since it's only computing the middle of the subarray, then it only takes O(1). C(n) is the cost of merging the array. In the worst-case scenario, the whole array is unsorted so it would take O(n) to go through the array. We are recursively calling 2T(n/2), with each array being the (n/2) of the whole array.



By following the tree as shown above we know that a merge sort is of O(n log(n)).

Users Manual:

The program was created using the Integrated Development Environment (IDE) Eclipse. This can run without Eclipse but works best with Eclipse.

How to use the insertion sort

To run the insertion sort algorithm, run the GUIRunner, and when the window pops up, click the insertion button. The unsorted random number array displayed at the top will be sorted with insertion sort. The white box in the middle of the window will be populated with the steps the algorithm performs in order. At the very bottom of this white box, the final sorted array will be displayed.

You can now click the reset button to create a new unsorted random number array, which you can sort again, or you can click the merge sort button to see how the merge sort would work on the same array.

How to use the merge sort

To run the merge sort algorithm, you'll want to click the run button in Eclipse. This will pop up a window. The user will already be given an unsorted array. From there they click on the button labeled merge, to run the merge sort algorithm. Afterward, you can choose to either try out the insertion sort, or you can use the reset button to get a new unsorted array.

Additional notes:

If you want to sort an array of a size different than 100, edit the Values interface. Change the SIZE variable to be the size of the array you would like to sort.