(a) Abstract:

In this project, we were required to implement the sorting algorithms and show their Time and Space complexity. Input numbers were to be taken from a file. We had to implement this project with a user friendly interface to see sortings.

(b) Introduction:

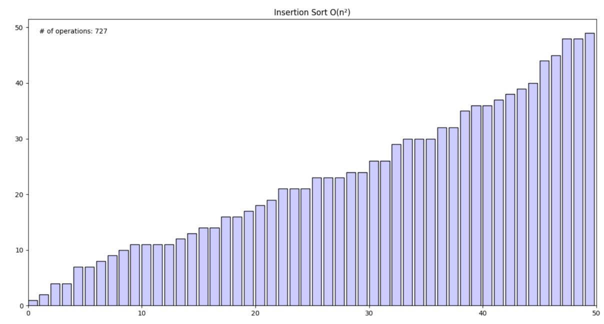
We had to show visualization of 10 algorithms, which is able to identify the pattern each sorting makes when executing. We designed a nice user interface where when the tab opens the user is asked to select the size of the file to input into the array and the sorting algorithm they might want to apply on it. Upon which a graph is displayed showing patterns that form while sorting until the graph is completely sorted. Space and time complexity for each sort was also calculated.

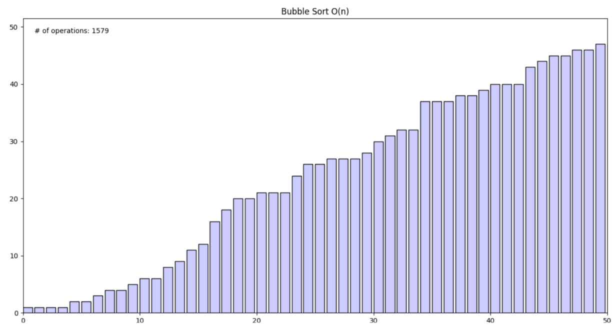
(c) Programming design:

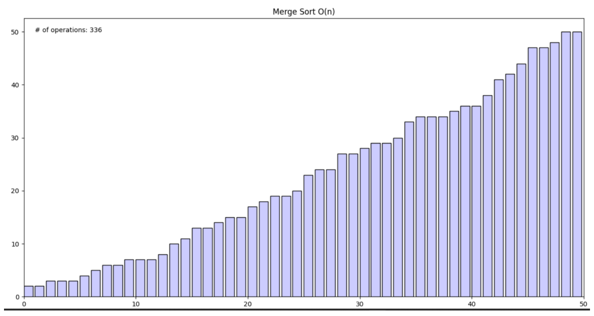
We set up the coding environment on Visual Studio Code. We decided to code in Python because of the various graph libraries it has. We made the user interface of the project using Tkinter and Matplotlib. Tkinter was mainly used to style the front end with different buttons and colors, while Matplotlib was used in making of the graph and showing the sorted values.

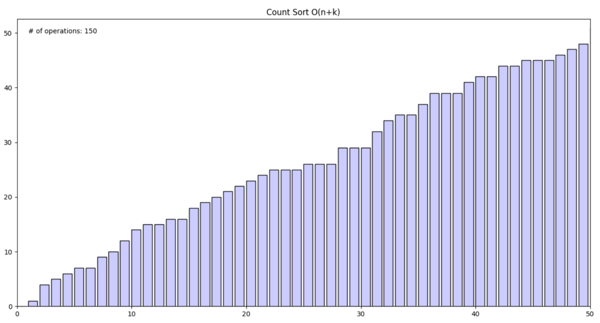
We made a separate file of each sorting algorithm and then imported it in the main file to make the code more usable and understandable. To provide the user with more file size options, we created a buildfiles file, which helped in creating four different files of different size with random values that needed to be sorted out.

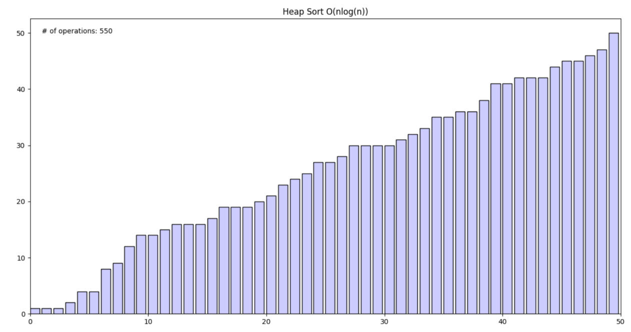
(d) Experimental Setup:

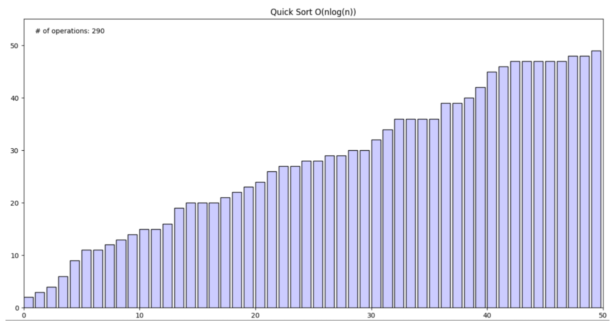


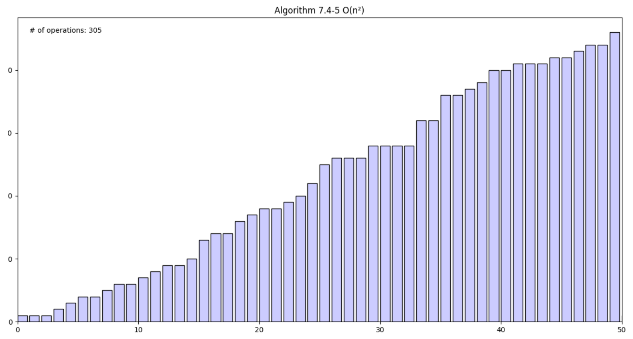


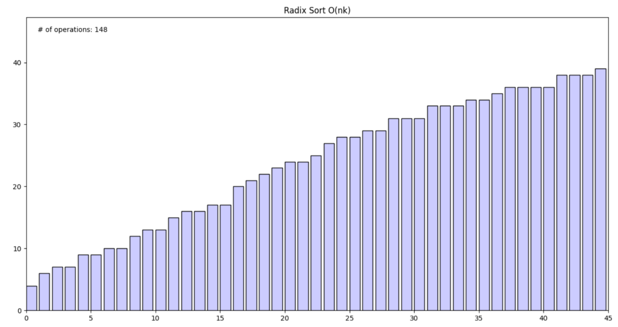












We calculated the number of operations each sort is taking to completely sort the array. These graphs were run on a **data of 50 random values.**

(e) Results and Discussion:

Bubble sort = 1579
Insertion sort = 727
Heap sort = 550
Merge sort = 336
Hybrid quicksort = 305
Quick sort = 290
Count sort = 150
Radix sort = 148

We can see that comparison algorithms are taking longer than counting algorithms. Radix sort takes the least operations to sort 50 values, while bubble sort takes the most. Heap sort is slower than Merge sort and Quick sort. Radix Sort was faster than Count sort.

(f) Conclusion:

We can make a conclusion that comparison algorithms are slower than count algorithms. In comparison algorithms, sorts with time complexity of O(nlog(n)) are faster than sorts with O(n) or O(n^2).

(g) References:

https://www.geeksforgeeks.org/?newui
https://www.programiz.com/
https://stackoverflow.com/
https://realpython.com/python-gui-tkinter/