(a) The motivation for your project. Why is the problem you decided to solve important or useful?

The motivation for the project is to find the most efficient and performant sorting algorithm. Sorting algorithms are one of the fundamental concepts in computer science, and they are essential in various applications, ranging from data analysis to web development.

As a programmer, it is crucial to understand the different types of sorting algorithms, their advantages, and their drawbacks. Choosing the suitable sorting algorithm can significantly impact the efficiency of your program and the time it takes to execute. However, it can also impact the resources consumed, such as memory and processing power.

This project aims to compare various sorting algorithms and determine the most efficient and performant one for different use cases. Furthermore, we aim to provide comprehensive analysis, including time complexity, space complexity, and practical implementation issues, to help programmers choose the best sorting algorithm for their specific use case.

Our project is essential because sorting is a joint operation in many computer science applications. By providing accurate and up-to-date information about sorting algorithms, we can help programmers optimize their code for faster execution and improved performance. This can ultimately lead to more efficient and reliable software, which is crucial for companies and organizations that rely on software systems to run their business.

Furthermore, by providing a comprehensive analysis of sorting algorithms, we can help students and developers new to programming learn about this fundamental concept. This can help them develop a strong foundation in computer science, which is essential for their future success.

(b) Background of the proposed problem. What have others done for it already? Include references.

There has been extensive research and development in sorting algorithms, and numerous algorithms have been proposed and analyzed over the years. For example, classical algorithms such as bubble sort, insertion sort, and selection sort have been well-studied and have been shown to have certain advantages and disadvantages in terms of time complexity, space complexity, and ease of implementation.

More recent algorithms, such as quicksort, mergesort, heapsort, and radix sort, have been developed and shown to improve performance and efficiency over their classical counterparts significantly. These algorithms have been extensively studied and are the subject of many research papers and studies.

Bubble Sort is one of the simplest sorting algorithms. It compares adjacent elements in the array and swaps them if they are in the wrong order. This process is repeated until the array is sorted. Bubble Sort is easy to understand and implement but could be faster and more efficient for large arrays.

Insertion Sort takes each element in the array and inserts it into its proper position. It is efficient for small arrays but not ideal for large arrays as it has a time complexity of O(n^2).

Selection Sort finds the smallest element in the array and swaps it with the first element. It then finds the next smallest element and swaps it with the second element, and so on, until the array is sorted. Selection Sort is easy to implement, but like Bubble Sort, it is inefficient for large arrays.

Merge Sort is a divide-and-conquer algorithm that works by dividing the array into two halves, sorting each half, and then merging the two halves. Merge Sort is efficient and has an O(n log n) time complexity, making it ideal for large arrays.

Quick Sort is also a divide-and-conquer algorithm that works by partitioning the array into two halves, sorting each half, and then combining the two halves. Quick Sort is very efficient for large arrays but can be slow for small arrays.

Heap Sort works by creating a heap data structure from the array and then repeatedly extracting the maximum element from the heap and placing it at the end of the array. Heap Sort has a time complexity of O(n log n), making it efficient for large arrays.

Despite the extensive research and development in this field, there still needs to be a one-size-fits-all sorting algorithm that is universally the best choice for all situations. Therefore, the proposed project aims to evaluate the performance and efficiency of various sorting algorithms and determine which are best suited for different data types and use cases.

(c) Detailed project overview, a summary of the proposed implementation, and the methodology used. Include pseudocode, diagrams, and examples if appropriate. If you are extending existing work, briefly describe previous work and include references to it.

The sorting algorithms are implemented using Python in this project, and the performance of the algorithms is visualized using the Matplotlib module. The PyQt5 library is used to create the GUI. It loads a CSV file containing data to be sorted and allows the user to select from seven different sorting algorithms to visualize their sorting process: bubble sort, selection sort, insertion sort, merge sort, quick sort, counting sort, and radix sort. The CSV module is used to load the CSV file, which is then stored in an instance variable called array. The names associated with each array value are stored in a separate list called name_data and are matched with their respective values using a dictionary called name_array. Every sorting algorithm is written as a function. The sorting process is depicted using the update_plot() function, which clears the canvas and plots the array's current state. When the GUI is opened, the algorithms are accessed by clicking on the desired one's button. To allow the GUI to update while sorting, QApplication.processEvents() is called after each plot update.

As the algorithm sorts the provided array of data, a plot is updated in real-time to represent the sorting algorithms. The plot displays the array's values as points on the x-axis and the indices that correspond to those points on the y-axis. In this case, the y-axis is the number that is being sorted while the x-axis is the amount of numbers that is being sorted. The spots on the plot are then rearranged by the sorting algorithms in real-time to reflect the sorted array. All of the sorting methods are implemented in a similar manner, each with its own algorithm. After each comparison or swap, each method updates the plot and calculates the algorithm's running time.

Some of the implementations of these algorithms were inspired by projects and labs that we have completed in class. Expanding upon these examples, we implemented them into the GUI so that many could be easily accessed at once without running multiple programs. Streamlining this process, we felt that this was a much better alternative to a text-based menu where you

would pick the desired algorithm. Also expanding upon what we learned in class, implementing the radix sort and counting sort algorithm required further research to implement. Using the links given in the repository, we were successfully able to implement our own algorithm in our project. While figuring this out and implementing proved to be a challenge, eventually by researching it enough we figured out how it worked. By using the same methodology as implementing the other algorithms, we were able to add it to our code.

(d) Conclusion. Give a short overview of your project and its results. Describe what you learned, what were the biggest challenges, and what the biggest rewards.

In conclusion, our project was aimed at developing a sorting algorithm visualization tool that would enable users to visualize the sorting process in real time. We achieved this by implementing several common sorting algorithms in Python and integrating them with a PyQt5 GUI to enable real-time visualization of the sorting process. Throughout the development process, we encountered several challenges, such as figuring out how to update the plot in real time and optimizing the sorting algorithms for large datasets. However, through collaboration and research, we were able to overcome these challenges and develop a functional sorting algorithm visualization tool. One of the biggest rewards of this project was gaining a deeper understanding of sorting algorithms and their performance characteristics. We also learned a lot about GUI development and data visualization in Python, which will be useful in future projects. Moving forward, we plan to continue improving the tool and expanding its functionality. One of the future enhancements we plan to make is displaying the runtimes of the sorting algorithms as the plotting is being done. We also suggest plugging in the machine to ensure better GPU performance for displaying full

runtimes. Overall, we believe that the sorting algorithm visualization tool we have developed has the potential to be a valuable resource for developers and educators interested in learning more about sorting algorithms and their performance.