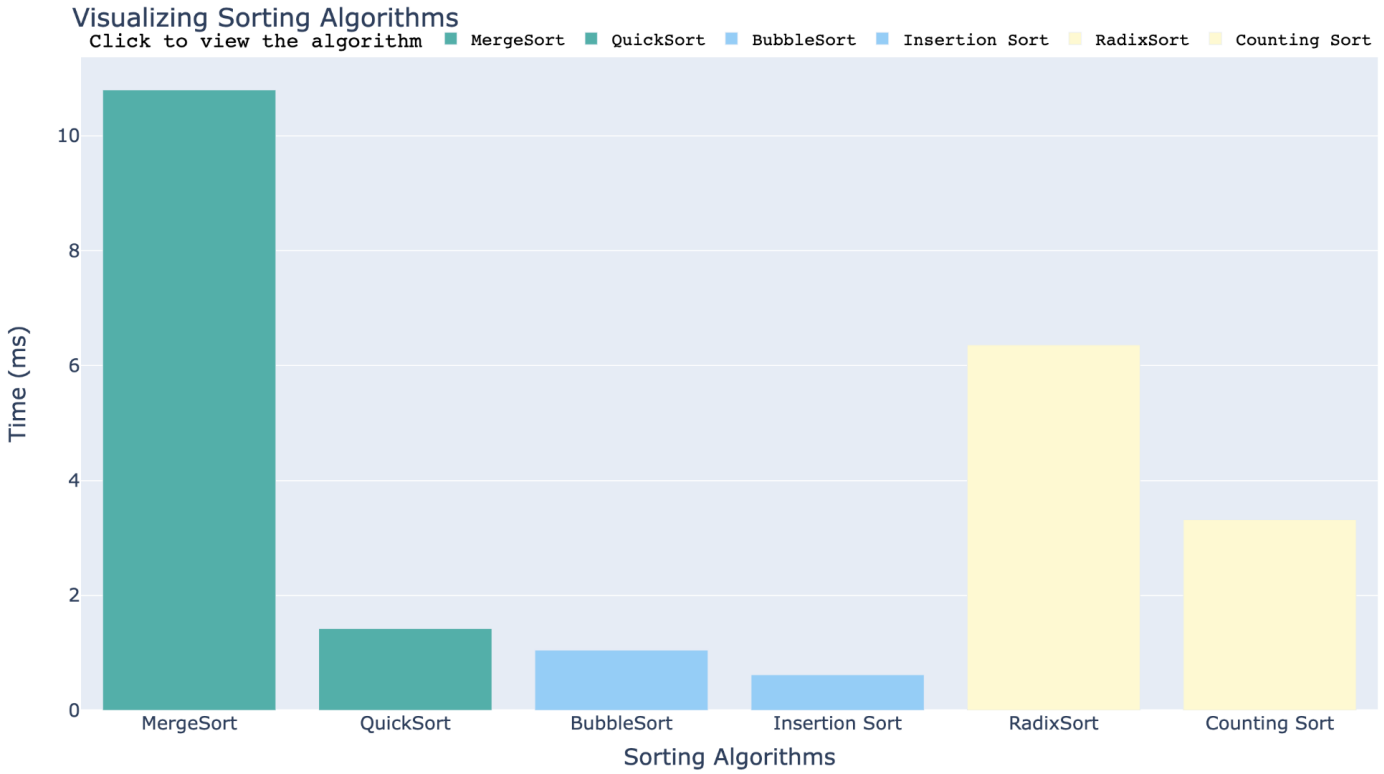
Visualization and Quantitative Comparisons of Sorting Algorithms for Large Data Sets

Shriya Dhaundiyal, Mariah Maynard, Aditya Puttigampala, Dmitrii Troitskii



GitHub: https://github.com/mariah-may23/finalproject.git
Youtube: https://youtu.be/wfBnkc-BRWE

Introduction

I. Context

Everyone has their own unique way of learning. From adapting to digital coursework to staying disciplined with minimal face-to-face interactions, getting used to this new type of education may cause students to struggle. This is especially true if their individual learning style isn't being addressed. Gaining momentum in the 1960s through tests like the Myers-Briggs Type Indicator, the learning style theory posits that different students learn best when information is presented to them in a particular way. The learning style theory was popularized in 1992 when Fleming and Mills suggested a new model of learning. The VARK Model is used to explain the different ways that students learn - Visual, Auditory, Reading-Writing, and Kinesthetic.

Visual learning refers to a mode of learning where students rely on graphic aids to remember and learn the material. Visual learners can easily visualize objects, have a great sense of balance and alignment, are very color-oriented, and can effortlessly envision imagery. Visual learners learn best by color-coding their notes, making to-do lists, and using concept maps to organize their thoughts.

When it comes to algorithms, some people consume the information by reading through pseudocode to understand the specifics while others may process that same information by visualizing how an algorithm works. Our group aims to dive further into the concept of visual learning and make algorithms more intuitive for their users by implementing visualizations for some of the most frequently used sorting algorithms.

Each of us has our own unique motivation for driving and implementing this goal. Coming into the Align program with diverse backgrounds, we have handled large volumes of data in our respective fields. At the same time, we have noticed the lack of educational tools and knowledge available for non-CS professionals. We wanted to implement a sorting program that could easily access and categorize the required parameterized data for these learners to utilize.

Aditya's motivation for pursuing this topic is because he loves to build financial products that users love. Having more context with tradeoffs for selecting

one algorithmic implementation versus another gives him more perspective during the costs/benefits analysis phase of product development. These insights are extremely valuable for making decisions and prioritizing which feature to build based on accurate data.

Coming from a healthcare background, Shriya is passionate about this topic because she finds learning visually very effective. Her experience of conducting various types of research while working with tons of patient information in the medical field exposed the problem of not being able to effectively categorize data and motivated her to understand the driving forces behind how this data can be more effectively sorted through. Visualizing the ways different data sets are being sorted and categorized satiates this curiosity and she believes this would serve as a great aid for non-tech grads to be able to make decisions regarding their choice of algorithms.

Mariah strongly believes in helping students connect the dots more quickly by giving them a tool to learn about multiple algorithms. Understanding when to use a specific algorithm and why is just as important as how an algorithm is implemented. Developers in training can get a better grasp on the tradeoffs of selecting one algorithm over another to augment their skillset. She believes this project will serve as a great educational tool for students trying to get a grip on the subject of sorting as a whole.

The goal of this project fits Dmitrii's pursuits of being better prepared to handle the multitude of problems that will be thrown at him on the job once he moves to the industry. Data visualizations make it easier to choose the right algorithm on the job by comparing and benchmarking the performance of various algorithms on different inputs.

II. The Question We Wish To Answer

Problem: The main aim of this project is to be able to build an educational, visualization tool for analyzing various sorting algorithms. Most learners struggle to see how different sorting algorithms work between each other for a common dataset. There are many websites and tools online like VisuAlgo that help understand the implementation of different algorithms for usually small data sets and only provide implementation knowledge to the user. Based on our research, we did not find any tools that would allow the user to understand and pick the best algorithm for their datasets based on comparisons of these sorting algorithms.

Solution: We want to design a visual tool for sorting algorithms that not only has the preliminary features of the prevalent online websites for visualizing algorithms i.e. understanding how the algorithms work by showing a step-by-step breakdown, but we want to take a step ahead and allow the users to see which algorithm would work better for their data sets based on a comparison chart. Our ultimate goal is to be able to implement basic visualizations along with comparisons specific to their data set and further, be able to implement a side-to-side visualization tool to help them understand the algorithms better. Our tool will allow a user to easily compare the performance of different algorithms on different inputs. We understand that the undertaking of this project is ambitious and therefore have divided the project into MVP and stretch goals dependent on the progress of our work as time permits.

III. Scope

We will examine various sorting algorithms for numerical datasets. More specifically, we intend to look at six key algorithms belonging to three distinct classes. These classes were decided based on their time complexities. They are as follows:

1. Linear – counting sort, and bucket sort

2. Linearithmic – merge sort and quicksort

3. Quadratic – insertion sort and bubble sort.

Though there are many more classes of algorithms based on this criterion, we believe that the examination of algorithms belonging to these unique categories will provide a diverse view of varying runtimes.

Time permitting, this tool can be expanded to include more algorithms in other classes for the purpose of comparison to expand the range of complexities investigated. Potentially, we will further develop our tool to include algorithmic-specific visuals to aid in the display of any algorithm. For example, this may include creating buckets to better explain the distribution of numerical elements in bucket sort. It may also include the demonstration of breaking an original array into singular fragments so they may be placed back together in ascending order for merge sort. These more intricate visualization ideas could further supplement the algorithms we wish to examine and would serve to create a further comprehensive tool.

Analysis

I. Getting Started

When our group met to discuss ideas for the final project we realized that we were facing the same question over the course of CS5800 – how can we improve our understanding of algorithms presented in the course, how can we get more hands-on practice with them, or put in programmer slang, – how can we grok the algorithms.

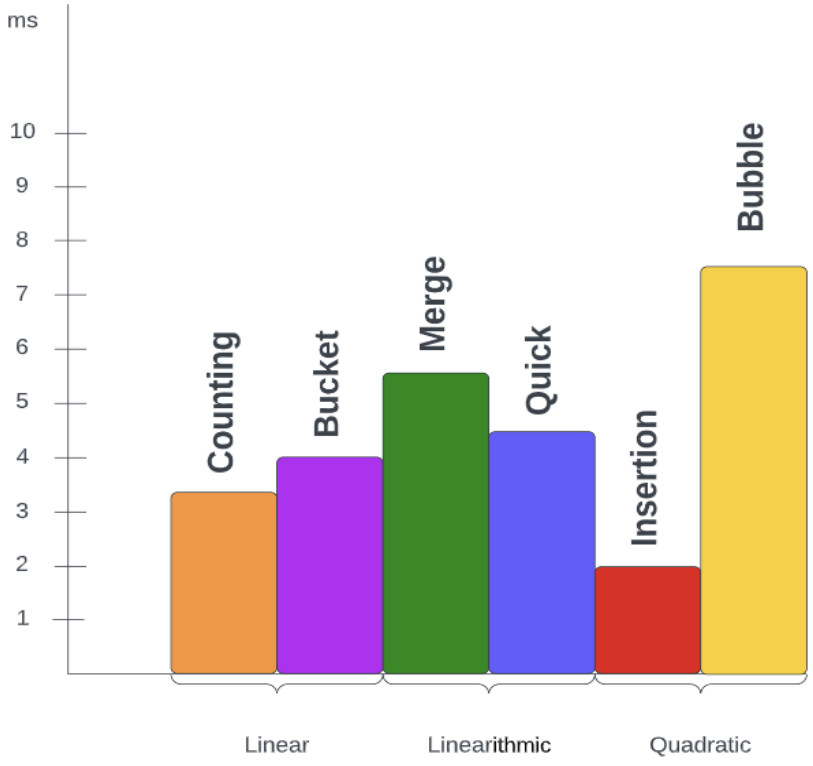
To define our approach to this problem we conducted a review of the algorithm visualization tools available on the web. Among the tools, we'd looked at

were Visualgo and the project of the USFCA. We determined that all the tools we've reviewed lacked two components – side-by-side algorithm comparisons and an actual performance benchmark for measuring which algorithm works better with the dataset of the user. Thus the idea of our project was born.

As a programming language to implement the project we chose Python for its ease of use, a wide variety of user-friendly libraries to use for visualization, and clean and neat implementations of sorting algorithms available.

Being aware of the time constraints, we decided to focus on implementing an MVP as a first step of the project. We chose a performance benchmark graph as a key feature of the MVP.

We have created a mock-up of the interface of the benchmark:



All remaining features were chosen to be stretch goals after we successfully implement the MVP. We decided to practice a small-scale team style of development, uploading our code to GitHub without pull requests and doing pair programming over zoom to enhance productivity and simplify time management.

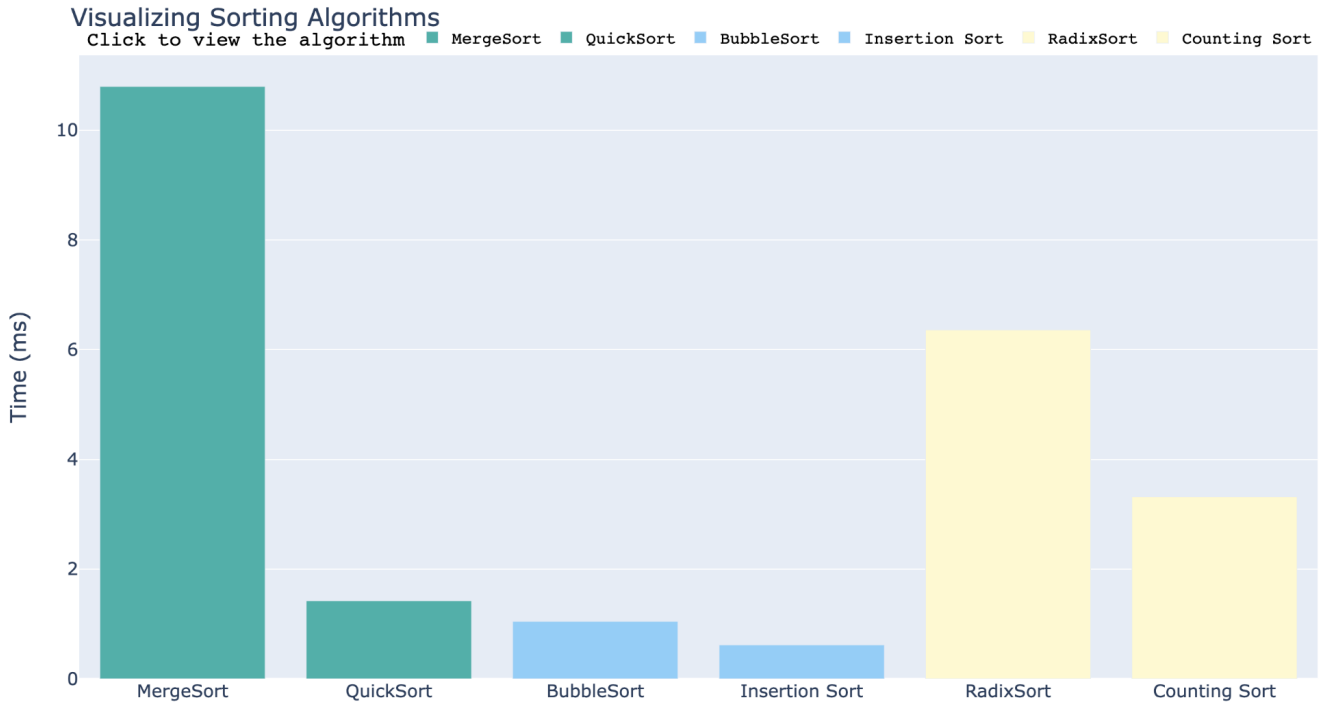
II. Creating the Benchmark Graph

First, to choose the visualization library to best fit our needs, we picked NumPy, Pandas x Matplotlib, and Plotly and tested each library for how fast and how close we can implement a bar chart we put together as an interface mockup. NumPy turned out to be lacking in features to implement detailed bar charts as we wanted, while Matplotlib and Plotly both were great options and allowed for all the features that we needed. Between Plotly and Matplotlib, Plotly looked like the best candidate since it has additional features such as displaying the results on the interactive webpage, showing additional information for each bar on mouse hover, and allowing to add buttons to hide/show the relevant bar charts. We created a Proof of Concept implementation with Plotly and chose it as a framework to work with in this phase of the project.

We initially decided to use pre-existing implementations of the algorithms and test them out using small inputs for the code. We realized that the existing implementations would not be ideal for handling large datasets so we recoded the algorithm using a uniform template for inputs as well as ensuring they can work on large data sets of any given size. Some things we needed to account for was using iterative methods over recursive to avoid reaching memory mark and using data structures that used minimal space for the dataset.

The visualization was created with a vision of being able to display the different sorting times in the most efficient and user-friendly manner. We wanted to use a graph representation that was both easy to display and easy to understand. We used Plotly's bar graphs to show the time taken on the y-axis and the sorting algorithms on the x-axis. We also wanted additional features for users to be able to isolate different algorithms and compare them against each other. This was implemented using the button functionality in which the users just need to click the algorithm names they want to remove (displayed on the top of the screen) from the graph and isolate the ones they want to keep. This allows better visualization of the comparator.

Based on the need for a proper visualizing tool and to analyze the different workings of the sorting algorithms, we coded a program to create a webpage that displayed the various sorting



After achieving our MVP, our major motivation was to be able to sort and display data for any industry, any field. To this end, we used CSVs from transportation, health, jail records, and insurance collections from Kaggle and hooked them to our code as feeder inputs for our algorithm. Our program was able to efficiently sort data, display the running time of different algorithms as well as produce an output file based on the most efficient algorithm.

III. Limitations

In the early stages of building our tool, we majorly focused on designing for the visual aspect of it. Initially, we ran into several issues concerning the look of our interface, including:

1. Mapping the color of each bar to the class of the algorithm
2. Creating an interactive legend as a header
3. Scaling plot columns

Plotly was chosen as the visualization library for the reasons that it best fit our needs and goals for the tool, however, we did not have prior experience using it. In order to develop our interface, further study into the fundamentals of this interactive graphing library gave us the understanding to resolve these issues.

The datasets we used when building the interface were all relatively small. Before a command line interface was created (see part IV), we manually entered datasets into our program so that an interface could be generated rather quickly. Initially, no errors occurred; the simple data was easily run and sorted. As we began to further develop our application, we decided to test with larger numbers and datasets. Here, we came across the following problem: our quicksort algorithm was exceeding the maximum recursion depth.

Originally, the quicksort algorithm used a recursive approach. The initial datasets did not cause this error because the function called itself fewer times than the maximum recursive depth. This differs from larger datasets, which need to call the function more times. When tested with large data, the IDE raised this error to protect against stack overflow that would arise in an infinite recursion. To prevent this error, we decided to use a quicksort algorithm with an iterative approach. This new approach did not raise any errors.

IV. Finding the Data

We sourced our data from Kaggle which is an open source site with publicly available data sets used in many software engineering development projects. Because there were so many formats for usable data ranging from csv, json, sql, etc., we had a plethora of options available for our data acquisition process.

We leveraged data sets ranging from convict data, insurance data, population data, and rental data to validate our sorting algorithms and test against datasets across a variety of industries. To keep our code consistent, we standardized the file format to be exclusively csv, with a consideration to handle additional formats in the future.

The data was fed into the application as a csv file along with the column to sort. Both of these parameters were then passed to a plot function that would take in the column to be sorted along with the file stored as a data frame. The plot function would then apply each of the predefined sorting algorithms to the selected column in the csv dataset.

V. Creating the Command-Line Interface

To provide ourselves and our users a way to comfortably use our application, we decided to implement a command-line interface (CLI).

This was the list of features that we've prioritized for our CLI to have:

1. Take a path to the .csv file from the user as an argument and pass the path to the csv-reader
2. Take the name of the column to sort from the user as an argument and pass the column name to the benchmark graph (plotting) module
3. Process any errors that the user makes while using the CLI

Since we were working in the results- and speed-oriented style, we decided to look for existing implementations of the aforementioned functionality.

Our choice was the CLI library called Typer. Typer provides out-of-the-box argument parsing, API for error handling, and simple style-customization of the command-line output (e.g. adding colors and font styles). It fit perfectly for our task.

Additionally, we implemented an error-handling logic for the file path and column name. Specifically, we are checking that the given path is correct and that the given column exists in the provided .csv file. Also, within the current scope of the project, we decided to limit sorting to only integer numerical values. To that end, we added another error check for the data type of the column.

Though we've only used a small subset of all the features that Typer offers, the ease of use that it opens prompts us to continue using this library in the consequent projects that our team makes.

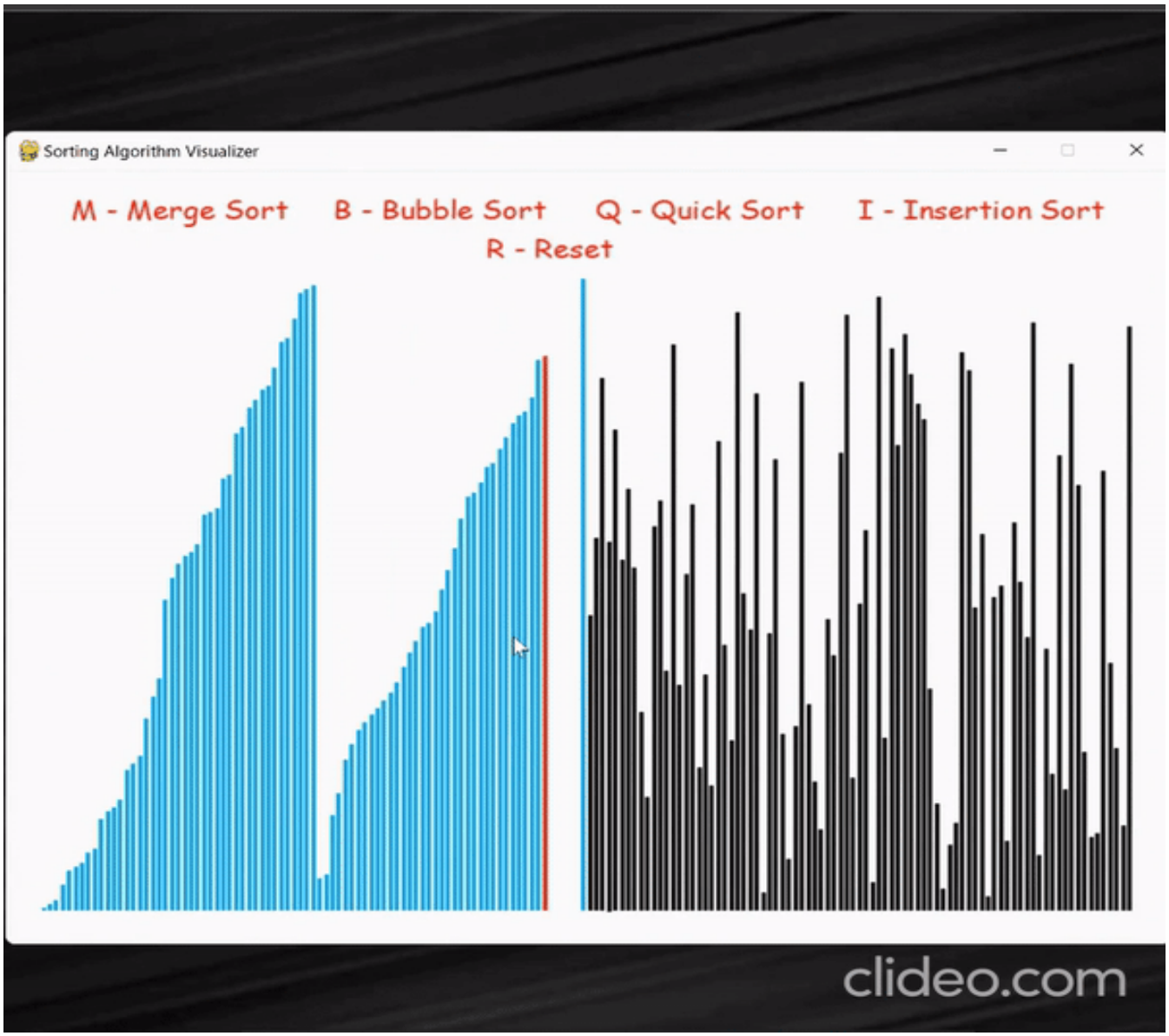
VI. Creating the Sorting Visualizer

After achieving our MVP, which was to create a basic comparator of the sorting time taken to sort the csv data sets. We achieved that, as described above, by showing a basic bar graph representation of the various sorting time taken by the algorithms. The next step was to actually develop the visualizer.

It was challenging initially to come up with a proper strategy for the implementation. The first choice was between the UI for the visualizer. Tkinter, was an easy to understand and apply UI but the biggest hurdle came when trying to implement algorithms that did not sort in place and required extra memory handling. Moreover, the GUI did not work smoothly for datasets greater than 200.

So, scratching the implementation right from the base, we turned to PyGame - the graphical package used to design games in Python. It was definitely a learning curve with the package, but the quality and quantity of sorting took a big leap. It was

easier to implement the setup for the algorithm dataset but the real challenge was to create a uniform platform for the implementation of different classes of algorithms handling both in-place and extra memory-requiring algorithms. The sorting algorithms had to be reformed to take in additional inputs for the graphical representation of various data being sorted at every step. We were able to refactor the code to show a proper visualization of data being sorted in ascending order. The visualizer is shown below.



VII. Refactoring

To make our code more readable and easy to work with – to run it as well as to implement new features, we've completed a refactoring.

We separated our code into two three modules – the sort module, containing all the sorting algorithms that we used, a main file, containing the CLI logic, and visu module, containing the logic responsible for displaying the benchmark graph.

Then the code was split up into logical blocks and abstracted into separate functions. We isolated three functions within the main module – the main function itself, which acts as a Controller and calls every other function in the project, the function to read the path to the file and process the file into a Pandas DataFrame, and the function to check whether the given column exists in the file and its datatype is an integer.

For the visu module, we decided to stick with one function to plot the benchmark graph since it nicely bundled together all the relevant procedures.

We cleaned up our code from the unnecessary comments and code used for testing. On the other hand, we added new comments to clarify certain procedures and commands within the project. We provided naming clarity to our variables and functions and made sure that each of the names follows the Python naming conventions.

Conclusion

In conclusion the team was able to successfully validate the problem statement and design an application to visualize sorting algorithms. Due to time constraints, we took a minimum viable product approach to include the critical functionality needed to demo a usable version of the tool. Some limitations of our project include :

1. The need for more datasets and larger sizes of our data to stress test our algorithms against.

2. Increasing the number of sorting algorithm possibilities to choose from would make this tool more robust by giving the user more flexibility with their analysis.

3. The tool currently uses the command line to run the program and make a selection for the data set and column to use. A better user experience would be to make user selections within the tool either via the menu bar or drop-down.

Further customizations can also be made to the visualizations by drilling deeper into the sorting algorithm and seeing the progress of the data manipulation live.

Aditya learned how to source the right data to fit the needs of the project goals. He quickly ran into some issues with data types the algorithms would be able to process, so the team needed to filter for integer-specific data. Aditya was also able to further develop his git skillset with a better understanding of leveraging branching strategy to collaborate with other software developers on the team. With further time available to spend on this codebase, future use cases could be to release this as an open source project to help other developers learn.

Dmitrii learned the power of the Typer library and found a love for building applications utilizing a command line interface. Dmitrii was able to get into the mindset of a user to creatively find pain points that a user would encounter. He was then able to build in informative error handling messages to improve the user experience. The results of this tool were very informative; In fact, Dmitrii believes he will run similar algorithms and visually compare datasets on the job once he graduates from Northeastern during his solution evaluation process.

Mariah learned how to use the plotly library to depict insightful visuals based on the data that was collected. She also discovered unique ways to make an interactive chart that responds to user clicks by slicing through the data. Mariah's research enabled her to develop a way to parallelly compute and display sorting results. This is a fantastic project that can be highlighted as part of a personal portfolio to be used during the career search and interview process.

Shriya learned how to work with Pandas data frames to manipulate data and pass around column-based sets across the application. She also learned how to parse the csv files and read in key data that needed to be stored. Moreover, she utilized the PyGame animation package of Python to create the visualizations of various sorting

algorithms which helped in achieving another milestone for this project. She hopes that the visual aspects help the user get a basic understanding of how the algorithm works after reading up on basic documentation. She aspires to eventually integrate more algorithms into her code base and turn it into an open learning platform for all students looking for a good resource to learn about algorithms. Shriya is passionate about research at Northeastern and this is a great example of a project that can be further advanced by working with a professor at the school.

Appendix (Application Code)

Benchmark Graph

```
# import the dependencies
import copy
import time

import pandas as pd
import plotly.express as px
from pandas import DataFrame

# import sorting algorithms from our custom sort module
from sort.bubble import bubble_sort
```

```
from sort.counting import count_sort
from sort.insertion import insertion_sort
from sort.mergeSort import merge_sort
from sort.quicksort import quicksort
from sort.radix import radix_sort


# sets up and displays a benchmark graph of sorting the given
file by the given column
def plot(file: DataFrame, column: str):
    # store algorithms to be used
    algos = ['Merge Sort', 'QuickSort', 'Bubble Sort', 'Insertion
Sort', 'Radix Sort', 'Counting Sort']

    # convert the dataframe column into a list
    numbers = file[column].tolist()

    # make a deep copy of the list to reuse it for each algorithm
    numbers_deep = copy.deepcopy(numbers)

    # let user know the program started
    print("Sorting... Wait until a browser window opens")

    # calculate how much time merge sort takes
    start = time.time()
    merge_sort(numbers)
    end = time.time()
    merge = (end - start) * 1000

    numbers = numbers_deep

    # calculate how much time bubble sort takes
    start = time.time()
    bubble_sort(numbers)
    end = time.time()
    bubble = (end - start)

    numbers = numbers_deep

    # calculate how much time counting sort takes
    start = time.time()
    count_sort(numbers)
```

```
    end = time.time()
    count = (end - start) * 1000

    numbers = numbers_deep

    # calculate how much time radix sort takes
    start = time.time()
    radix_sort(numbers)
    end = time.time()
    radix = (end - start) * 1000

    numbers = numbers_deep

    # calculate how much time quick sort takes
    start = time.time()
    quicksort(numbers)
    end = time.time()
    quick = (end - start)

    numbers = numbers_deep

    # calculate how much time insertion sort takes
    start = time.time()
    insertion_sort(numbers)
    end = time.time()
    insertion = (end - start) * 1000

    # store times
    times = [merge, quick, bubble, insertion, radix, count]

    # create dataframe
    df = pd.DataFrame(
        {"Sorting Algorithm": ['MergeSort', 'QuickSort',
'BubbleSort', 'Insertion Sort', 'RadixSort', 'Counting Sort'],
         "Time Complexity": ['LINEARITHMIC (n log(n))',
'LINEARITHMIC (n log(n))', 'QUADRATIC (n^2)',
                             'QUADRATIC (n^2)', 'LINEAR (n)',
'LINEAR (n)'],
         "Time": times,
         })

    # create the figure
```

```
    """
        'x' and 'y' tell what info to store on the axis
        'labels' allows you to name the axis
        'hover_name' allows to to store a title for the bar when
it is hovered over
    """

    fig = px.bar(df, x=algos, y=times, color='Sorting Algorithm',
                 labels={'x': 'Sorting Algorithm', 'y': 'Time
(ms)'},
                 hover_name='Time Complexity',
                 color_discrete_sequence=["lightseagreen",
"lightseagreen", "lightskyblue", "lightskyblue",
                                          "palevioletred",
                                          "palevioletred"],
                 )

    # add the buttons to hide the chosen algorithm bar charts
    fig.update_layout(legend=dict(font=dict(family="Courier",
size=17, color="black"), orientation="h",
                                  yanchor="bottom",
                                  y=1.00,
                                  xanchor="right",
                                  x=1),
                      legend_title="Click to hide the bar chart:",
                      font=dict(
                          size=18))
    fig.show()  # show webpage
```

Command-Line Interface

```
# import the dependencies
import numpy as np
import pandas as pd
import typer
from pandas import DataFrame
```

```
# import the benchmark graph module
from visu import plot


# define main function in a style of the Controller pattern
# expects two CLI arguments: path to the .csv file and a name of
the column in the file
def main(file: str, column: str):
    # read .csv file into a Pandas DataFrame
    df = read_file(file)
    # check that the given column exists and its datatype is
integer
    check_col(df, column)
    # display the benchmark for the given data
    plot(df, column)


# reads file from the given path into a dataframe
def read_file(file: str):
    try:
        df = pd.read_csv(file)
        return df
    # catch any errors during reading the file
    except:
        # let user know that there is a mistake in the path
        typer.secho(
            "Incorrect file path or name", fg=typer.colors.RED  #
displays the message in red
        )
        raise typer.Exit()


# checks that the given column exists in the dataframe
def check_col(df: DataFrame, column: str):
    # let user know that the given column does not exist
    if column not in df.columns:
        typer.secho(
            "A column with the given name does not exist",
fg=typer.colors.RED  # displays the message in red
        )
        raise typer.Exit()
    # let user know that the given column's data type is not
```

```
integer
   elif df.dtypes[column] != np.int64:
       typer.secho(
           "The type of the column should be an integer",
fg=typer.colors.RED  # displays the message in red
       )
       raise typer.Exit()


if __name__ == "__main__":
   typer.run(main)
```

Visualizer Algorithm

```
import pygame
import random

pygame.init()

WIDTH, HEIGHT = 800, 600

FONT_BIG = pygame.font.SysFont('comicsans', 40)
FONT_MEDIUM = pygame.font.SysFont('comicsans', 30)
FONT_SMALL = pygame.font.SysFont('comicsans', 20)

win = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('Sorting Algorithm Visualizer')

DARK_GRAY = '#65696B'
LIGHT_GRAY = '#C4C5BF'
BLUE = '#0CA8F6'
DARK_BLUE = '#4204CC'
WHITE = '#FFFFFF'
BLACK = '#000000'
RED = '#F22810'
YELLOW = '#F7E806'
PINK = '#F50BED'
```

```
GREEN = '#013220'
PURPLE = '#BF01FB'

NUM_BAR = 170
BORDER = 25

SORTED = False

SPACE = (WIDTH - 25 - BORDER) / NUM_BAR

BAR_WIDTH, BAR_HEIGHT = SPACE - 1.2, 2.87

FPS = 80

RUN = True

DOWN = 25

COUNT = 0

sorting = [FONT_SMALL.render('M - Merge Sort', 1, RED),
           FONT_SMALL.render('B - Bubble Sort', 1, RED),
           FONT_SMALL.render('Q - Quick Sort', 1, RED),
           FONT_SMALL.render('I - Insertion Sort', 1, RED)]
reset = FONT_SMALL.render('R - Reset', 1, RED)


class Bar:

    def __init__(self, x, y, width, height, value):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.color = BLACK
        self.value = value

    def reset(self, num):
        self.x = BORDER + num * SPACE
        return self

    def draw(self, win):
```

```
        pygame.draw.rect(win, self.color, (self.x, self.y,
self.width, self.height))

    def check(self):
        self.color = RED

    def done(self):
        self.color = BLUE

    def match(self):
        self.color = YELLOW

    def back(self):
        self.color = BLUE


sort = [x for x in range(1, NUM_BAR + 2)]
random.shuffle(sort)
bar = [Bar((BORDER + i * SPACE), (HEIGHT - DOWN - (BAR_HEIGHT *
sort[i])), BAR_WIDTH, BAR_HEIGHT * sort[i], sort[i]) for
       i in range(NUM_BAR)]


def main(win):
    clock = pygame.time.Clock()

    global RUN
    global bar
    global SORTED
    while RUN:

        clock.tick(FPS)
        draw(win, bar)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                RUN = False
                break

        keys = pygame.key.get_pressed()

        if keys[pygame.K_r]:
            random.shuffle(sort)
```

```
            bar = [Bar((BORDER + i * SPACE), (HEIGHT - DOWN -
(BAR_HEIGHT * sort[i])), BAR_WIDTH, BAR_HEIGHT * sort[i],
                       sort[i]) for i in range(NUM_BAR)]
            SORTED = False
        if keys[pygame.K_s]:
            check()
            selection(bar, win)
        if keys[pygame.K_b]:
            check()
            bubble(bar, win)
        if keys[pygame.K_m]:
            check()
            merge_sort(bar, 0, len(bar) - 1, win)
        if keys[pygame.K_q]:
            check()
            quick_sort(bar, 0, len(bar) - 1, win)

        if keys[pygame.K_i]:
            check()
            insertion(bar, win)

    pygame.quit()


def check():
    global SORTED
    global bar
    if SORTED:
        SORTED = False
        random.shuffle(sort)
        bar = [Bar((BORDER + i * SPACE), (HEIGHT - DOWN -
(BAR_HEIGHT * sort[i])), BAR_WIDTH, BAR_HEIGHT * sort[i],
                   sort[i]) for i in range(NUM_BAR)]


# draw function for the screen
def draw(win, bar):
    global reset
    global sorting

    win.fill(WHITE)
```

```
    pygame.draw.rect(win, WHITE, (0, 0, 800, 180))

    x = 45
    y = 15
    win.blit(sorting[0], (x, y))
    x = x + 180
    win.blit(sorting[1], (x, y))
    x = x + 180
    win.blit(sorting[2], (x, y))
    x = x + 180
    win.blit(sorting[3], (x, y))

    win.blit(reset, (330, 45))

    for i in bar:
        i.draw(win)

    pygame.display.update()




def bubble(bar, win):
    global RUN
    global SORTED
    SORTED = True

    for i in range(len(bar)):
        if not RUN:
            break
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                RUN = False
                break

        for j in range(len(bar) - 1 - i):

            bar[j].check()
            draw(win, bar)
            bar[j].back()
```

```
            if bar[j].value > bar[j + 1].value:
                temp = bar[j]
                bar[j] = bar[j + 1]
                bar[j + 1] = temp
                bar[j].reset(j)
                bar[j + 1].reset(j + 1)

        bar[len(bar) - 1 - i].done()


def merge(bar, left, mid, right, win):
    temp = []
    i = left
    j = mid + 1

    while i <= mid and j <= right:
        bar[i].check()
        bar[j].check()
        draw(win, bar)
        bar[i].back()
        bar[j].back()
        if bar[i].value < bar[j].value:
            temp.append(bar[i])
            i += 1
        else:
            temp.append(bar[j])
            j += 1
    while i <= mid:
        bar[i].check()
        draw(win, bar)
        bar[i].back()
        temp.append(bar[i])
        i += 1
    while j <= right:
        bar[j].check()
        draw(win, bar)
        bar[j].back()
        temp.append(bar[j])
        j += 1
    k = 0
    for i in range(left, right + 1):
        bar[i] = temp[k]
```

```
        bar[i].reset(i)
        bar[i].check()
        draw(win, bar)
        if right - left == len(bar) - 1:
            bar[i].done()
        else:
            bar[i].back()
        k += 1


def merge_sort(bar, left, right, win):
    global SORTED
    SORTED = True
    global RUN

    mid = left + (right - left) // 2
    if left < right:
        merge_sort(bar, left, mid, win)
        merge_sort(bar, mid + 1, right, win)
        if not RUN:
            return
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                RUN = False
                break
        merge(bar, left, mid, right, win)


def quick_sort(bar, low, high, win):
    global RUN
    global SORTED
    SORTED = True

    if len(bar) == 1:
        return bar
    if low < high:

        pi = partition(bar, low, high, win)

        draw(win, bar)

        quick_sort(bar, low, pi - 1, win)
```

```
        if not RUN:
            return
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                RUN = False
                break

        for i in range(pi + 1):
            bar[i].done()

        quick_sort(bar, pi, high, win)


def partition(bar, low, high, win):
    i = low
    pivot = bar[high]
    pivot.color = RED
    for j in range(low, high):

        bar[j].check()
        bar[i].check()
        draw(win, bar)
        bar[j].back()
        bar[i].back()
        if bar[j].value < pivot.value:
            bar[i], bar[j] = bar[j], bar[i]
            bar[i].reset(i)
            bar[j].reset(j)
            i += 1

    bar[i], bar[high] = bar[high], bar[i]
    bar[i].reset(i)
    pivot.back()
    draw(win, bar)
    bar[high].reset(high)
    return i


def insertion(bar, win):
    global RUN
    global SORTED
```

```
    SORTED = True

    for i in range(1, len(bar)):
        bar[i].check()
        draw(win, bar)
        bar[i].back()
        j = i
        while j > 0 and bar[j].value < bar[j - 1].value:
            if not RUN:
                return
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    RUN = False
                    break
            bar[i].check()
            draw(win, bar)
            bar[i].back()
            bar[j], bar[j - 1] = bar[j - 1], bar[j]
            bar[j].reset(j)
            bar[j - 1].reset(j - 1)
            j -= 1
    for i in range(len(bar)):
        bar[i].done()
        pygame.time.delay(1)
        draw(win, bar)


if __name__ == "__main__":
    main(win)
```