Algorithm Efficiency Analyzer Tool

Group Project 1

Advanced Algorithm Section 4 Fall 2023

Date of Submission - 16th September, 2023

Team Members	
Keshav Daga	GUI Developer
Vedita Deshpande	GUI Developer
Akash Avinash Butala	Algorithm Specialist and Testing
Vaishnavi More	Algorithm Specialist and Testing
Sri Sai Navya Manchikalapudi	Documentation and Presentation
Chathrapathi Nikhil Kandagatla	Documentation and Presentation
Hari Preetham Reddy Takuru	Data Visualization
Mourya Velampati	Data Visualization

Table of Contents

ABSTRACT

The “Algorithm Efficiency Analyzer” is a collaborative project, done by a team of 8 people with different roles and responsibilities. This helps users with a deeper understanding of algorithms efficiency, GUI design, and data visualization techniques. Five key roles and responsibilities include:

- GUI Developer: Responsible for designing an intuitive graphical user interface (GUI) that allows users to input a list of numbers, select a sorting algorithm, and visualize its efficiency.
- Algorithms Specialist and Testing: Tasked with implementing sorting algorithms while ensuring optimization and correctness.
- Documentation and Presentation: Responsible for developing a comprehensive test suite to ensure the application's reliability, documenting the code.
- Data Visualization: Charged with creating visual representations of algorithm efficiency using the matplotlib library and collaborating closely with the GUI Developer to seamlessly integrate these visualizations.

The team follows a set of instructions for successful project completion:

- Emphasizing regular updates and communication to maintain synchronization among team members.
- Expanding the existing codebase with the implementation of additional sorting algorithms.
- Creating an interactive and user-friendly GUI capable of handling various input scenarios.
- Crafting visual aids, such as bar graphs and scatter plots, to showcase the efficiency of each sorting algorithm.
- Rigorously testing the tool with diverse input sizes and datasets to guarantee its reliability.
- Comprehensively documenting the code, processes, roles, and outcomes, accompanied by a short presentation to showcase the tool's features.

Through the collaborative efforts of the team and adherence to these instructions, the "Algorithm Efficiency Analyzer Tool" seeks to provide a valuable resource for those interested in understanding and comparing the efficiency of various sorting algorithms, ultimately enhancing their knowledge of algorithmic principles and data visualization techniques.

INTRODUCTION

The 'Algorithm Efficiency Analyzer Tool' project is a collaborative effort focused on deepening our understanding of algorithm efficiency, GUI design, and data visualization while making these critical concepts accessible to a wider audience. It serves as a potent educational tool, providing hands-on experience in algorithm implementation, GUI development, and data representation. This multifaceted project equips us with skills directly applicable in real-world scenarios, enhancing our problem-solving abilities and teamwork.

The core of the project revolves around education and practical learning. Through the creation of a user-friendly platform where users can input data and select from a variety of sorting algorithms, we gain valuable experience in implementing, optimizing, and comparing these algorithms. This practical exposure enhances our comprehension of their intricacies and helps us grasp their relative efficiencies.

The GUI design component challenges us to bridge the gap between technical complexity and user-friendliness. Crafting an intuitive interface that enables smooth data input, algorithm selection, and real-time visualization necessitates creative problem-solving, refining our ability to communicate complex ideas to a diverse audience.

Furthermore, data visualization plays a pivotal role in conveying algorithm efficiency concepts to users, a skill with broad applications in fields like data science. Incorporating diverse sorting algorithms into our tool provides us with a versatile problem-solving toolkit, an asset valued in various professional contexts. This project also emphasizes collaboration, enhancing our teamwork and communication skills.

By documenting our progress and sharing our experiences and insights, we aim to inspire and educate others about the importance and practicality of algorithm efficiency, GUI design, and data visualization, encouraging them to embark on similar educational journeys. In summary, the 'Algorithm Efficiency Analyzer Tool' project is a comprehensive learning experience, combining theory and practice, fostering creativity and teamwork, and promoting knowledge sharing, all of which equip us with valuable skills for the future.

ROLES AND RESPONSIBILITIES

GUI Developer (Keshav and Vedita):

1. Conceptualize and design the layout and visual elements of the tool.
2. Develop an interactive interface that enables users to input data and choose sorting algorithms effortlessly.
3. Ensure consistency in design elements and adhere to the project's overall aesthetic.

Algorithm Specialists and Testing (Akash and Vaishnavi):

1. Optimize algorithms to solve complex problems or perform specific tasks as required by the project.
2. Test and validate algorithms to ensure they produce accurate and reliable results and make necessary adjustments as needed.
3. Build CI/CD workflow that automates the process of building code changes, and testing them, enabling faster and more reliable software releases.

Documentation and Presentation (Navya and Nikhil):

1. Project Documentation: Thoroughly document the project's codebase, including comments, explanations, and usage instructions.
2. Presentation Design: Design an engaging and informative presentation that effectively communicates the project's features, functionalities, and achievements.
3. Demonstration: Conduct a live demonstration during the presentation, highlighting how users can interact with the tool and interpret visualizations.
4. Conclusion: Summarize the project's objectives, achievements, and the skills acquired by the team.

Data Visualization (Preetham and Mourya):

1. Understand algorithm complexity.
2. Collect and prepare data.
3. Design effective Matplotlib visualizations.
4. Code and implement visualizations.
5. Customize visualizations for the tool's needs.
6. Integrate visualizations into the tool's interface.
7. Optimize performance for large datasets.
8. Document on how to interpret visualizations.
9. Test and validate visualizations.
10. Incorporate user feedback for improvements.

METHODOLOGY

Our project, the "Algorithm Efficiency Analyzer Tool," was developed through a coordinated and well-structured methodology, with each team member assigned specific roles and responsibilities. The following outlines our approach, the tools we employed, and the technologies used throughout the project:

1. **Agile Development Framework:** We embraced an agile development methodology, fostering collaboration, adaptability, and regular iteration cycles. This approach enabled us to address challenges swiftly and refine our work continuously.
2. **Version Control and Collaboration:** Git served as our version control system, allowing seamless collaboration on code development. Our Git repository was hosted on platforms like GitHub, facilitating version tracking and team coordination.
3. **Communication Tools:** For real-time communication and coordination, we utilize tools like Zoom. These platforms facilitated team discussions, file sharing, and meeting scheduling, ensuring constant updates and synchronization.
4. **Programming Languages and Frameworks:** Our project was predominantly developed using Python, a versatile language suitable for algorithm implementation and data visualization. We employed Python's matplotlib library for creating insightful visualizations. The GUI was designed using frameworks like Tkinter depending on team members' preferences and familiarity.
5. **GUI Development (Keshav and Vedita):** As the GUI Developers, Keshav and Vedita were responsible for designing an intuitive and user-friendly interface. They focused on creating an interactive environment that allowed users to input data, select sorting algorithms, and view visualizations with ease.
6. **Algorithm Implementation and Testing (Akash and Vaishnavi):** Akash and Vaishnavi assumed the roles of Algorithm Specialists and Testing Leads. They meticulously implemented sorting algorithms, including Heapsort, Quicksort, Linear Sorting, and Medians & Order Statistics. Additionally, they created a robust test suite, ensuring correctness and reliability.
7. **Data Visualization (Preetham and Mourya):** Preetham and Mourya specialized in Data Visualization. They harnessed the matplotlib library to convert algorithm efficiency data into visually engaging representations. Bar graphs, scatter plots, and other visuals were designed to illustrate sorting algorithm performance effectively.
8. **Documentation and Presentation (Navya and Nikhil):** Navya and Nikhil took on the roles of Documentation and Presentation Leads. They meticulously documented the codebase, including comprehensive comments and test cases. Their efforts extended to preparing an informative presentation that effectively communicated the project's features, functionality, and achievements.

This structured methodology allowed our team to collaborate efficiently, manage version control, develop code, test rigorously, and document comprehensively. As a result, the "Algorithm Efficiency Analyzer Tool" was successfully created, providing a user-friendly platform for exploring algorithm efficiency, GUI design, and data visualization techniques.

IMPLEMENTED ALGORITHMS

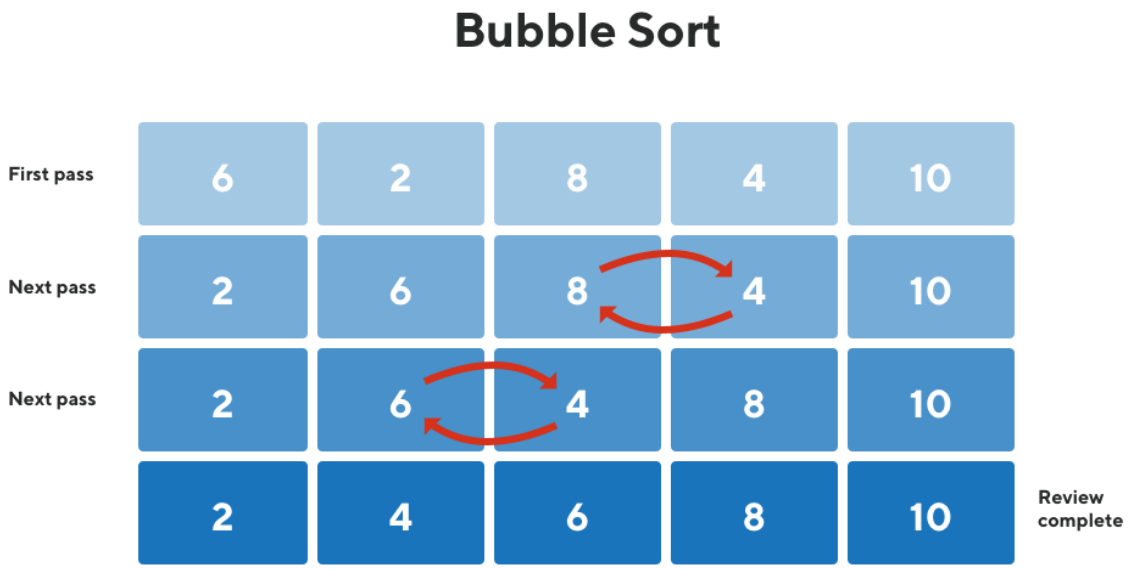
Algorithms Implemented in this tool are as follows-

Algorithm	Time Complexity (Average & Worst Case)	Time Complexity (Best Case)	Space Complexity
Bubble Sort	O(n^2)	O(n)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n^2)	O(n)	O(1)
Merge Sort	O(n log n)	O(n log n)	O(n)
Quick Sort	O(n^2) (worst case) / O(n log n) (average case)	O(n log n)	O(log n) (in-place) / O(n) (not in-place)
Heap Sort	O(n log n)	O(n log n)	O(1)
Radix Sort	O(n * k) (k is the number of digits)	O(n * k)	O(n + k)
Counting Sort	O(n + k) (k is the range of input values)	O(n + k)	O(k)

Bubble Sort -

Bubble sort is the sorting technique in which adjacent elements are swapped until they get into intended order.

This algorithm is not suitable for larger datasets because of the higher time and space complexity.



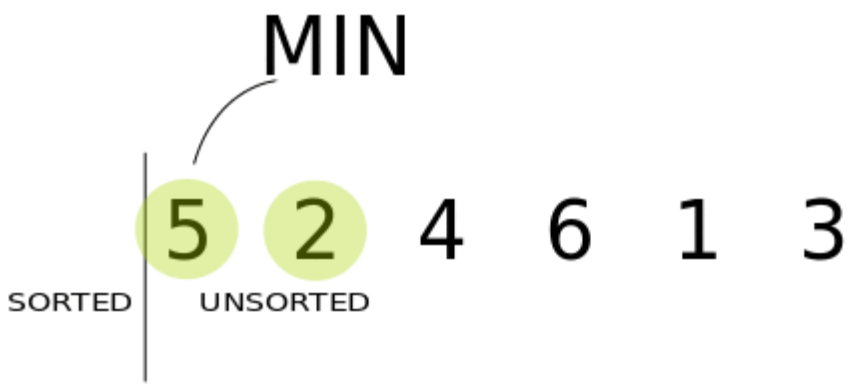
[1]

Pseudocode -

```
def bubbleSort(arr: array)
    n = length(arr)
    for i in range 0 to n-1
        for j in range 0 to n-i-1
            if arr[j] > arr[j+1]
                swap(arr[j], arr[j+1])
    return arr
```

Selection Sort:

Begin with a list of unsorted numbers. Look for the smallest number in the list. Move this smallest number to the beginning of the sorted part of the list. Repeat these steps until all numbers are in the sorted part. In the end, you'll have a list of numbers sorted in ascending order.



Pseudocode-

```
function selectionSort(arr):
    n = length of arr
    for i from 0 to n-1:
        minIndex = i
        // Find the index of the minimum element in the unsorted part of the array
        for j from i+1 to n-1:
            if arr[j] < arr[minIndex]:
                minIndex = j
        // Swap the minimum element with the first element in the unsorted part
        swap arr[i] with arr[minIndex]
    // The array is now sorted
    return arr
```

Insertion Sort:

Insertion sort works similarly as we sort cards in our hand in a card game.
Insertion Sort builds the final sorted array one item at a time. It takes each element from the unsorted part and inserts it into its correct position within the sorted part.

[3]

Pseudocode -

```
def insertionSort(arr: array)
   n = length(arr)
   for i from 1 to n-1
      key = arr[i]
      j = i - 1
      while j >= 0 and arr[j] > key
         arr[j+1] = arr[j]
         j = j - 1
      arr[j+1] = key
    return arr
```

Merge Sort:

Merge sort is a recursive algorithm which splits the array into half until it cannot be divided. Merge Sort is a divide-and-conquer algorithm that divides the input list into smaller sublists, recursively sorts them, and then merges them back together.

[4]

Pseudocode -

```
def mergeSort(arr: array)
    if length(arr) <= 1
        return arr

    mid = length(arr) / 2
    left = arr[0:mid-1]
    right = arr[mid:]

    left = mergeSort(left)
    right = mergeSort(right)

    return merge(left, right)

def merge(left: array, right: array)
    result = empty array
    while left != empty and right != empty
        if left[0] <= right[0]
            append left[0] to result
            remove left[0] from left
        else
            append right[0] to result
            remove right[0] from right
```

```
    append remaining elements of left to result
    append remaining elements of right to result

    return result
```

QuickSort:

QuickSort uses **divide and conquer** algorithm.It selects a "**pivot**" element and partitions the list into elements less than the pivot and elements greater than the pivot. It then **recursively** sorts the sublists.

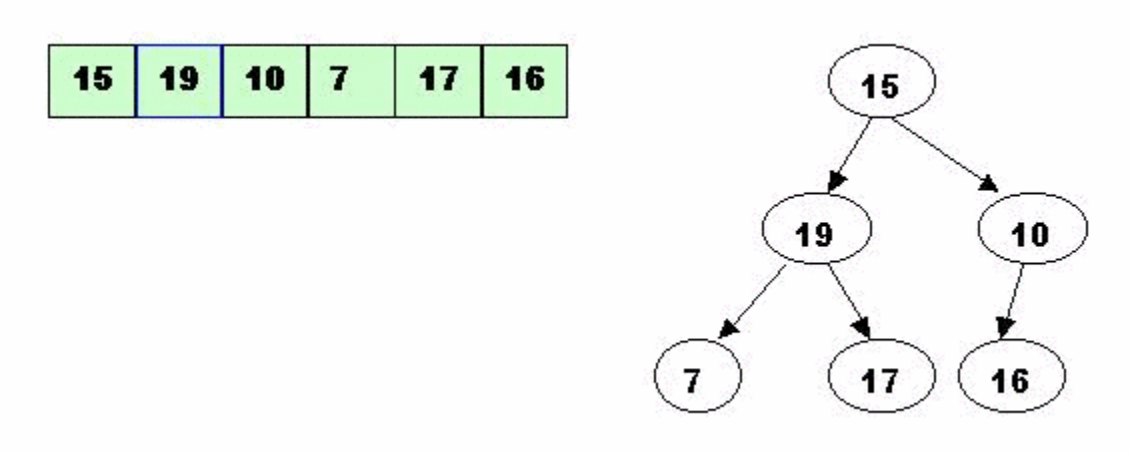
[5]

Pseudocode -

```
def quickSort(arr: array)
    if length(arr) <= 1
        return arr

    pivot = arr[randomIndex]  // Choose pivot element

    left = elements in arr less than pivot
    middle = elements in arr equal to pivot
    right = elements in arr greater than pivot

    left = quickSort(left)
    right = quickSort(right)

    return concatenate(left, middle, right)
```

Heap Sort:

Heap sort is a comparison-based sorting algorithm that uses a binary heap data structure to sort elements in an array, achieving a time complexity of O(n log n). It repeatedly extracts the maximum (in a max heap) or minimum (in a min heap) element from the heap and places it at the end of the sorted portion of the array until all elements are sorted.



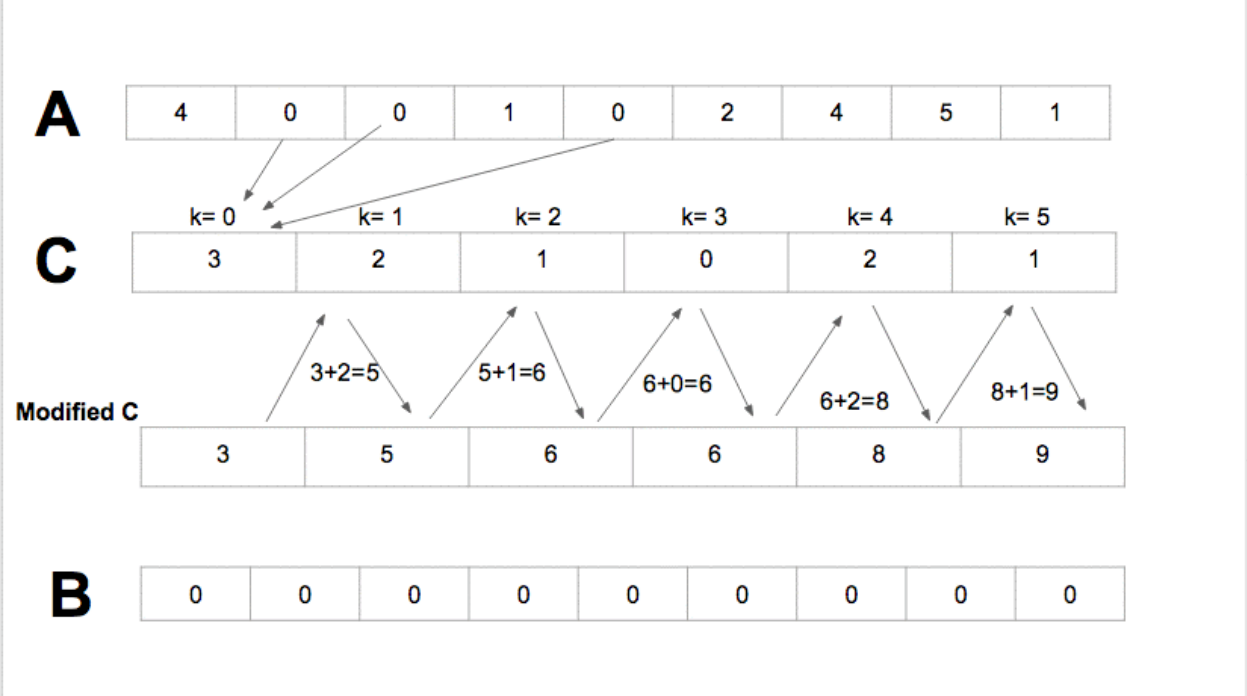
Pseudocode -

```
heapSort(arr):
    n = length(arr)

    // Build a max heap from the input array
    for i from n/2 - 1 down to 0:
        heapify(arr, n, i)

    // Extract elements from the heap and place them in the sorted portion
    for i from n - 1 down to 0:
        swap(arr[0], arr[i]) // Move the current root to the end of the sorted portion
        heapify(arr, i, 0)   // Restore the heap property for the remaining elements

heapify(arr, n, i):
    largest = i
    leftChild = 2 * i + 1
    rightChild = 2 * i + 2

    // Find the largest element among the root, left child, and right child
    if leftChild < n and arr[leftChild] > arr[largest]:
        largest = leftChild

    if rightChild < n and arr[rightChild] > arr[largest]:
        largest = rightChild

    // If the largest element is not the root, swap and continue heapifying
    if largest != i:
        swap(arr[i], arr[largest])
```

heapify(arr, n, largest)

Radix Sort:

Radix sort is a non-comparative sorting algorithm that processes elements by their individual digits, sorting them from the least significant digit to the most significant digit (or vice versa) to achieve linear time complexity relative to the number of digits and elements. It is particularly effective for sorting integers with a limited range of values but is not suitable for arbitrary data types.



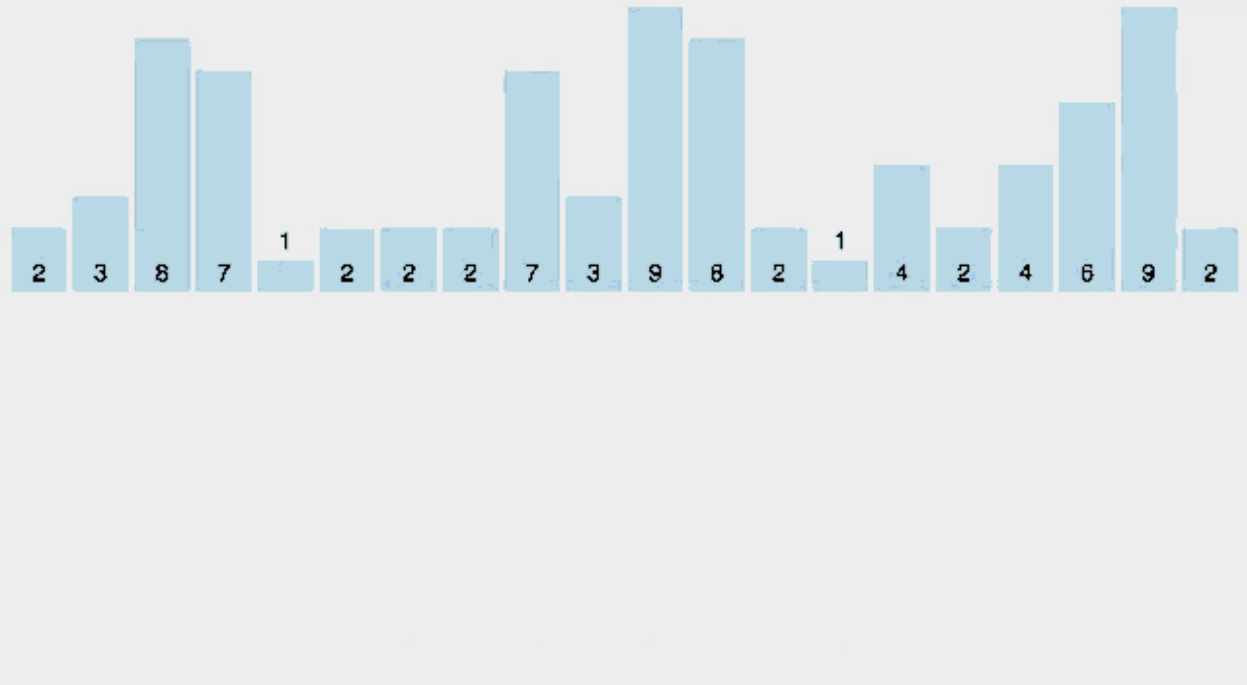
Pseudocode -

```
radixSort(arr):
    // Find the maximum number to determine the number of digits
    maxNumber = findMax(arr)
    // Perform counting sort for each digit position, from LSD to MSD
    for exp from 1 to maxNumber:
        countingSort(arr, exp)
countingSort(arr, exp):
    n = length(arr)
output = [0] * n
```

```
    count = [0] * 10
    // Count the occurrences of each digit at the current position
    for i from 0 to n - 1:
        digit = (arr[i] / exp) % 10
        count[digit]++
    // Update count to store the actual position of each digit in output
    for i from 1 to 9:
        count[i] += count[i - 1]
    // Build the output array based on the count array and the current digit position
    for i from n - 1 down to 0:
        digit = (arr[i] / exp) % 10
        output[count[digit] - 1] = arr[i]
        count[digit]--
    // Copy the sorted output array back to the original array
    for i from 0 to n - 1:
        arr[i] = output[i]
findMax(arr):
    max = arr[0]
    for i from 1 to length(arr) - 1:
        if arr[i] > max:
            max = arr[i]
 return max
```

Counting Sort:

Counting sort is a straightforward sorting algorithm for integers with a known range of values. It counts how many times each integer appears in the input array, then uses this count information to place the integers in sorted order.



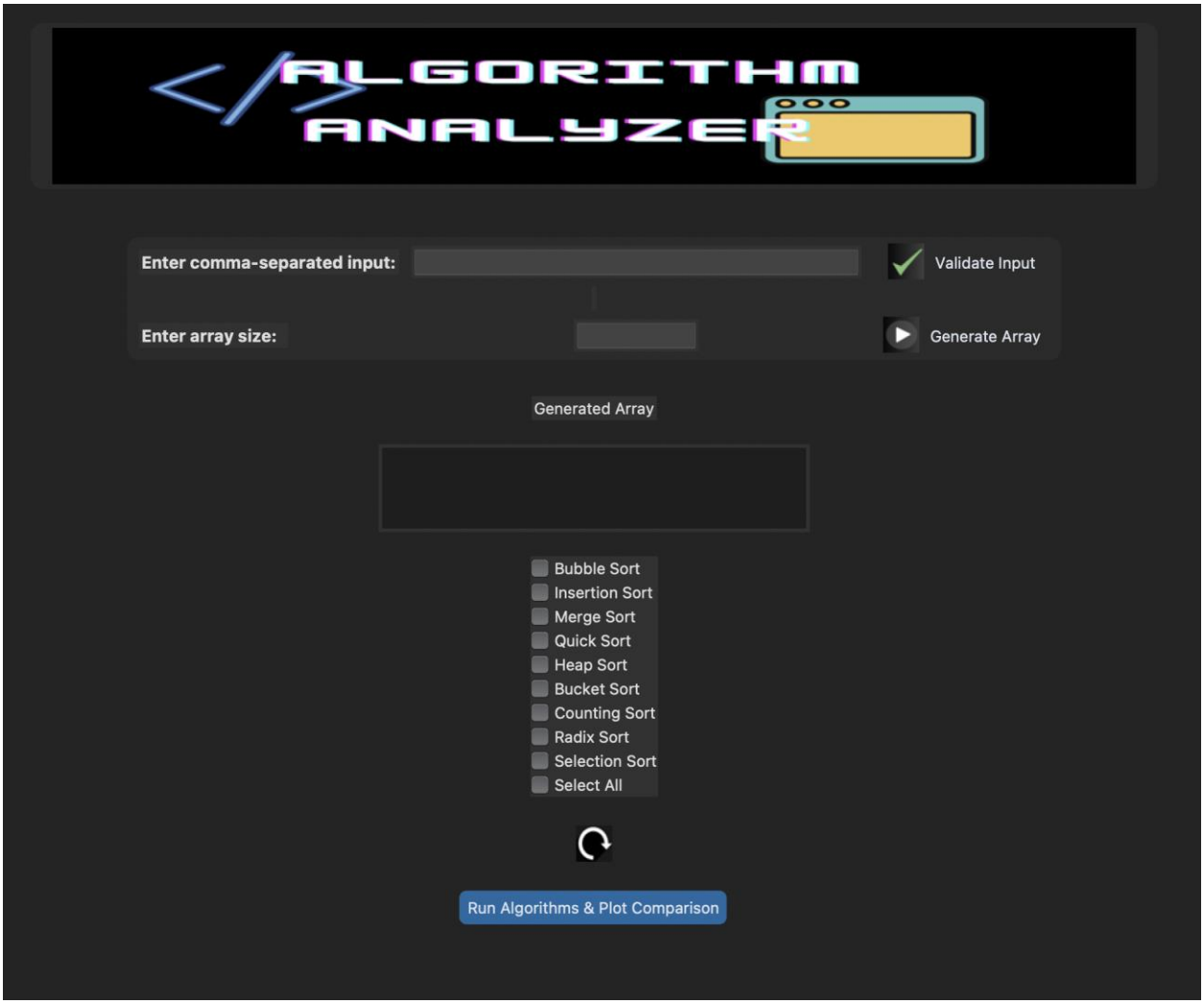
Pseudocode -

```
def counting_sort(arr):
    # Find the maximum and minimum values in the input array
    max_val = max(arr)
    min_val = min(arr)
    # Create a counting array of size k (range of values)
    k = max_val - min_val + 1
    counting_array = [0] * k
    # Count occurrences of each value in the input array
    for num in arr:
        counting_array[num - min_val] += 1
    # Accumulate counts in the counting array
    for i in range(1, k):
```

```
    counting_array[i] += counting_array[i - 1]
# Build the output array
output = [0] * len(arr)
for num in reversed(arr):
    output[counting_array[num - min_val] - 1] = num
    counting_array[num - min_val] -= 1
return output
```

GUI DESIGN

The presented Python script showcases an interactive graphical user interface (GUI) designed to unravel the efficiency of diverse sorting algorithms. Leveraging the versatile **tkinter** library for GUI development, this script offers users an intuitive platform to input data, select preferred sorting algorithms, execute the chosen algorithms, and visually compare their performance. With a seamless blend of input validation, dynamic array generation, and real-time plotting using 'matplotlib', the GUI empowers users to explore the world of sorting algorithms with ease and clarity, making it a valuable tool for both learning and analysis.



Here's a description of the different components and functionalities of the GUI:

1. **Importing Libraries**:
    - The script starts by importing necessary libraries like **tkinter** for GUI, **matplotlib** for plotting, and other specific modules for input handling, sorting, and appearance customization.

```
#Importing necessary libraries and modules
import tkinter as tk
from tkinter import ttk
import random
import time
from input_handling import validate_input
from IntSort import SortInteger
import matplotlib.pyplot as plt
import customtkinter
from PIL import Image, ImageTk
import random
import matplotlib.animation as animation
from matplotlib.animation import FuncAnimation
```

2. **Global Variables**:
    - Various global variables are defined to store user input, sorting results, and flags to control program behavior.

```
# Define global variables for user input and sorting results
unsorted_array = [] # To store random generated array
user_input = "" # To store user input
flag = False # To switch between the user input and random generated array
time_list = {} # To store the time taken to run the algorithm
```

3. **User Input Handling**:
    - The GUI allows the user to input comma-separated numbers and select a data type. Input validation is performed, and feedback is provided regarding the validity of the input.

```
# Function to handle user input and validation
def handle_user_input():
    global user_input, data_type, flag
    user_input = input_field.get()
    if validate_input(user_input, "number"):
        error_label.config(text="Valid input", foreground="green")
        # Update the generated_array_text widget with the new array
        generated_array_text.delete(1.0, tk.END)  # Clear previous content
        generated_array_text.insert(tk.END, ''.join(map(str, user_input)))
        flag = False
    else:
        # To print error message
        error_label.config(
            text="Invalid input. Please check your data type selection.", foreground="red"
        )
```

4. **Array Generation**:
    - The GUI allows the user to generate a random array of a specified size, which is then displayed in the interface.

```python
# Function to generate a random array
def generate_array():
    global unsorted_array, flag
    array_size = array_size_entry.get()
    if array_size.isdigit():
        array_size = int(array_size)
        unsorted_array = [random.randint(1, 1000) for _ in range(array_size)]
        array_size_label.config(
            text="Array generated with size: " + str(array_size)
        )
        flag = True
        # Update the generated_array_text widget with the new array
        generated_array_text.delete(1.0, tk.END)  # Clear previous content
        generated_array_text.insert(tk.END, ', '.join(map(str, unsorted_array)))
    else:
        array_size_label.config(text="Invalid input for array size")
```

5. **Sorting Algorithms Selection**:
    - The user can select which sorting algorithms they want to run from a predefined list, including options like Bubble Sort, Merge Sort, etc.

```python
# Create StringVar instances to track the checkbox states
algorithm_vars = [tk.StringVar() for _ in algorithm_labels]

# Create checkboxes for each sorting algorithm
algorithm_checkboxes = [
    ttk.Checkbutton(algorithm_frame, text=label, variable=var)
    for label, var in zip(algorithm_labels, algorithm_vars)
]

# Place the checkboxes in the algorithm frame
for i, checkbox in enumerate(algorithm_checkboxes):
    checkbox.grid(row=i, column=0, sticky=tk.W)


# Create a "Select All" checkbox
select_all_var = tk.IntVar()
select_all_checkbox = ttk.Checkbutton(
    algorithm_frame, text="Select All", variable=select_all_var
)

select_all_checkbox.configure(command=toggle_all_checkboxes)
select_all_checkbox.grid(row=len(algorithm_labels), column=0, sticky=tk.W)
```

6. **Algorithm Execution and Timing**:
    - When the user initiates the sorting, the selected algorithms are executed on the generated or provided array, and the execution times for each algorithm are recorded.

```python
# Function to run selected sorting algorithms and plot results
def run_algorithms():
    # Dictionary to store execution times for each algorithm
    time_list = {}
    global flag
    selected_algorithms = [algo_var.get() for algo_var in algorithm_vars]
    execution_times = []

    for algo_index, selected in enumerate(selected_algorithms):
        if selected == "1":
            # Determine the array to sort based on user input and flag
            if flag:
                arr_new1 = unsorted_array
            else:
                arr_input = [int(x) for x in user_input.split(",")]
                arr_new1 = arr_input
            print(arr_new1)
            arr_new = arr_new1.copy()  # Make a copy of the input array
            start_time = time.time()
```

7. **Comparison Plot**:
    - After execution, the GUI displays a bar chart comparing the execution times of the selected sorting algorithms.

```python
    # List of colors for graph
    colors = ["red", "yellow", "green", "blue", "purple", "orange", "pink", "cyan", "magenta"]


    # Create a bar graph
    fig, ax = plt.subplots(figsize=(10, 6))
    bars = ax.bar(time_list.keys(), time_list.values(), color=colors)  # Use the colors directly
    ax.set_ylabel("Execution Time (microseconds)")
    ax.set_title("Execution Time of Sorting Algorithms on a Sorted Array")

    # Function to update the bar heights for animation
    def update(heights):
        for bar, new_height in zip(bars, heights):
            bar.set_height(new_height)

    # Animate the graph
    def animate(frame):
        # Replace execution times with animated values (e.g., frame * 0.01 for a simple animation)
        new_execution_times = [time * (frame * 0.01) for time in time_list.values()]
        update(new_execution_times)

    ani = FuncAnimation(fig, animate, frames=100, interval=100, repeat=False)

    # Display the animated graph
    plt.xticks(rotation=15, ha="right")  # Rotate x-axis labels for better readability
    plt.tight_layout()
    plt.show()
```

8. **Reset and Display**:
    - Options to reset the input and checkboxes or display an output are provided through buttons.

```
316   # Create and configure the reset button
317   reset_image = ImageTk.PhotoImage(
318       Image.open("img/reset_icon.png").resize((30, 30))
319   )
320   reset_button = customtkinter.CTkButton(
321       root,
322       text="",
323       fg_color="transparent",
324       command=reset_values,
325       image=reset_image,
326       hover_color="black",
327       width=50
328   )
329   reset_button.pack(side=tk.TOP, padx=5, pady=10, anchor=tk.CENTER)
```

9. **Appearance Customization**:
    - The appearance of the GUI is customized using a custom **customtkinter** module to set themes and colors. We used default themes for this project.

10. **Main Window and Layout**:

    · The main window and layout for the GUI are defined, including frames, labels, buttons, and other widgets.

```
175   # Create the main window
176   root = customtkinter.CTk()
177   root.title("Sorting Algorithm Efficiency Analyzer")
178
179   # configuring the grid
180   root.grid_columnconfigure(1, weight=1)
181   root.grid_columnconfigure((2, 3), weight=0)
182   root.grid_rowconfigure((0, 1, 2), weight=1)
```

Overall, the GUI provides a user-friendly interface for inputting data, selecting sorting algorithms, running the algorithms, and visualizing the performance of the algorithms through a plot. It's a tool to analyze and compare the efficiency of various sorting algorithms based on user input.

VISUALIZATION

The visualization generated by our code displays the execution times (in seconds) of different sorting algorithms. It leverages Matplotlib to create an animated bar graph, illustrating how execution times change over time.

A bar graph is used to display categorical data with rectangular bars. Each bar's length or height is proportional to the value it represents. In this example, we visualize the execution times of different sorting algorithms on a sorted array.

Y-axis represents the execution time in seconds. X-axis shows the various algorithm types.

We used animation to display the change in complexity for each sorting algorithm with time frames in the bar graph such that there would be an eye-catching view of the difference of complexity.

By the usage of color coding for the bar graph we could see the clear difference in the change in complexity of different sorting algorithms.

1. **Importing Libraries:**
    - **matplotlib.pyplot** and **matplotlib.animation.FuncAnimation** are imported to enable graph plotting and animation.
2. **Data Preparation:**
    - **sorting_algorithms**: A list containing the names of sorting algorithms (e.g., Bubble Sort, Selection Sort etc..).
    - **execution_times**: A list containing example execution times (in seconds) for each sorting algorithm.
    - **colors**: A list specifying the colors for each bar in the bar graph.
3. **Creating the Bar Graph:**
    - **fig, ax = plt.subplots(figsize=(10, 6))** creates a figure and axes for the plot with a specified figure size.
    - **bars = ax.bar(sorting_algorithms, execution_times, color=colors)** creates a bar graph with sorting algorithm names on the x-axis and execution times on the y-axis, using the specified colors.
4. **Customizing the Graph:**
    - **ax.set_ylabel('Execution Time (seconds)')** sets the y-axis label.
    - **ax.set_title('Execution Time of Sorting Algorithms on a Sorted Array')** sets the graph title.
5. **Function for Updating Bar Heights:**
    - **def update(heights)** is a function that updates the heights of the bars in the graph based on the provided heights.
6. **Animating the Graph:**
    - **def animate(frame)** is a function that is called repeatedly to update the graph's animation frames.

    - Inside **animate**, the **new_execution_times** list is calculated based on the **frame** parameter, creating a dynamic animation by modifying the execution times of sorting algorithms.
7. **Creating the Animation:**
    - **ani = FuncAnimation(fig, animate, frames=100, interval=100, repeat=False)** creates an animation object using **FuncAnimation**. It specifies:
        - **fig**: The figure to be animated.
        - **animate**: The function responsible for updating the graph.
        - **frames=100**: The number of animation frames.
        - **interval=100**: The time interval (in milliseconds) between frames.
        - **repeat=False**: Ensures that the animation doesn't repeat after all frames are displayed.
8. **Displaying the Graph:**
    - **plt.xticks(rotation=15, ha='right')** rotates the x-axis labels for better readability.
    - **plt.tight_layout()** improves the layout of the graph.
    - **plt.show()** displays the animated graph to the user.

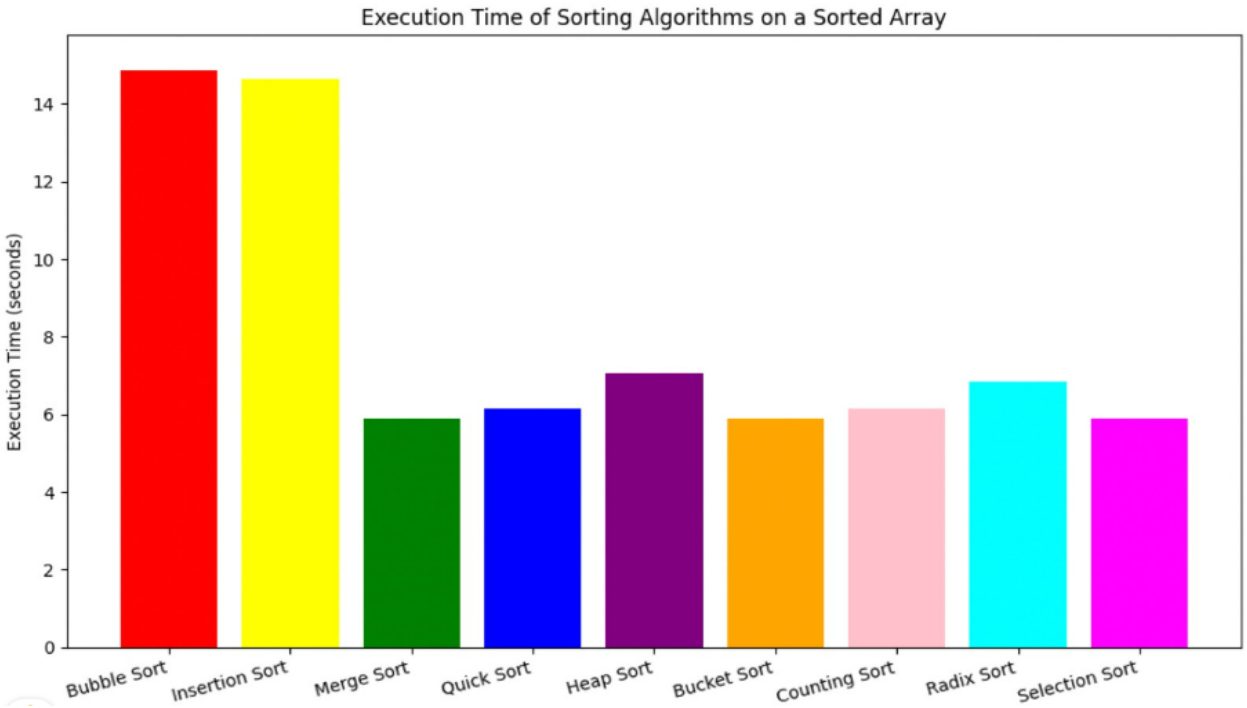
Matplotlib Explanation

Matplotlib is a powerful Python library for creating static, animated, and interactive visualizations. It provides a wide range of functions and classes for creating various types of plots and graphs. In this code, we use Matplotlib to create a bar graph and animate it over a specified number of frames.

Key Matplotlib Concepts Used in the Code:

- **plt.subplots()**: Creates a figure and axes for the plot.
- **plt.bar()**: Creates a bar graph with specified data.
- **ax.set_ylabel()**: Sets the y-axis label.
- **ax.set_title()**: Sets the title of the graph.
- **FuncAnimation()**: Creates an animation object for dynamic updates.
- **plt.xticks()**: Customizes x-axis labels.
- **plt.tight_layout()**: Improves the layout of the graph.
- **plt.show()**: Displays the graph.

This code demonstrates the versatility of Matplotlib in creating dynamic visualization of bar graphs with animations that display the complexities of all the stated sorting algorithms.


Execution Time of Sorting Algorithms on a Sorted Array
Execution Time (seconds)
0
2
4
6
8
10
12
14
Bubble Sort
Insertion Sort
Merge Sort
Quick Sort
Heap Sort
Bucket Sort
Counting Sort
Radix Sort
Selection Sort

TESTING

"IntSortTest.py" is a unit testing script designed to evaluate the functionality and performance of different sorting algorithms implemented in the "*SortInteger.py"* module. It utilizes the "*unittest"* framework to define a series of test cases, each corresponding to a specific sorting algorithm such as insertion sort, bubble sort, merge sort, heap sort, bucket sort, counting sort, quick sort, and radix sort. The script generates an array of unsorted integers to serve as input data for these tests. For each sorting algorithm, the script checks whether the algorithm produces the expected sorted output by comparing it to Python's built-in sorted function's result. Additionally, it ensures that the time consumed by the sorting algorithms is greater than or equal to zero, verifying their correctness and performance. Running this script allows for automated testing and validation of the sorting algorithms provided by the *"SortInteger.py"* module.

```python
class TestSortInteger(unittest.TestCase):

    def setUp(self):
        self.unsorted_array = [211, 34, 545, 6, 1, 0, 23, 98, 76, 12,
                               45, 67, 89, 32, 11, 90, 54, 87, 23, 65,
                               87, 12, 56, 32, 10, 2, 43, 87, 34, 76,
                               8, 99, 31, 78, 54, 20, 7, 19, 65, 77,
                               43, 90, 15, 37, 88, 44, 61, 29, 53, 76]

    def test_insertion_sort(self):
        sorted_array, time_consumed = SortInteger.SortInteger(self.unsorted_array, "InsertionSort")
        self.assertEqual(sorted_array, sorted(self.unsorted_array))
        self.assertGreaterEqual(time_consumed, 0)

    def test_bubble_sort(self):
        sorted_array, time_consumed = SortInteger.SortInteger(self.unsorted_array, "BubbleSort")
        self.assertEqual(sorted_array, sorted(self.unsorted_array))
        self.assertGreaterEqual(time_consumed, 0)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                powershell

PS D:\WebDevelopment\circleci-demo> python3 .\IntSortTest.py
0.0
Bubble Sort :  [0, 1, 2, 6, 7, 8, 10, 11, 12, 12, 15, 19, 20, 23, 23, 29, 31, 32, 32, 34, 34, 37, 43, 43, 44, 4
5, 53, 54, 54, 56, 61, 65, 65, 67, 76, 76, 76, 77, 78, 87, 87, 87, 88, 89, 90, 90, 98, 99, 211, 545]
.Bucket Sort :  [0, 1, 2, 6, 7, 8, 10, 11, 12, 12, 15, 19, 20, 23, 23, 29, 31, 32, 32, 34, 34, 37, 43, 43, 44,
45, 53, 54, 54, 56, 61, 65, 65, 67, 76, 76, 76, 77, 78, 87, 87, 87, 88, 89, 90, 90, 98, 99, 211, 545]
.Counting Sort :  [0, 1, 2, 6, 7, 8, 10, 11, 12, 12, 15, 19, 20, 23, 23, 29, 31, 32, 32, 34, 34, 37, 43, 43, 44
, 45, 53, 54, 54, 56, 61, 65, 65, 67, 76, 76, 76, 77, 78, 87, 87, 87, 88, 89, 90, 90, 98, 99, 211, 545]
.Heap Sort :  [0, 1, 2, 6, 7, 8, 10, 11, 12, 12, 15, 19, 20, 23, 23, 29, 31, 32, 32, 34, 34, 37, 43, 43, 44, 45
, 53, 54, 54, 56, 61, 65, 65, 67, 76, 76, 76, 77, 78, 87, 87, 87, 88, 89, 90, 90, 98, 99, 211, 545]
.Insertion Sort :  [0, 1, 2, 6, 7, 8, 10, 11, 12, 12, 15, 19, 20, 23, 23, 29, 31, 32, 32, 34, 34, 37, 43, 43, 4
4, 45, 53, 54, 54, 56, 61, 65, 65, 67, 76, 76, 76, 77, 78, 87, 87, 87, 88, 89, 90, 90, 98, 99, 211, 545]
.Merge Sort :  [0, 1, 2, 6, 7, 8, 10, 11, 12, 12, 15, 19, 20, 23, 23, 29, 31, 32, 32, 34, 34, 37, 43, 43, 44, 4
5, 53, 54, 54, 56, 61, 65, 65, 67, 76, 76, 76, 77, 78, 87, 87, 87, 88, 89, 90, 90, 98, 99, 211, 545]
.Quick Sort :  [0, 1, 2, 6, 7, 8, 10, 11, 12, 12, 15, 19, 20, 23, 23, 29, 31, 32, 32, 34, 34, 37, 43, 43, 44, 4
5, 53, 54, 54, 56, 61, 65, 65, 67, 76, 76, 76, 77, 78, 87, 87, 87, 88, 89, 90, 90, 98, 99, 211, 545]
.Radix Sort :  [0, 1, 2, 6, 7, 8, 10, 11, 12, 12, 15, 19, 20, 23, 23, 29, 31, 32, 32, 34, 34, 37, 43, 43, 44, 4
5, 53, 54, 54, 56, 61, 65, 65, 67, 76, 76, 76, 77, 78, 87, 87, 87, 88, 89, 90, 90, 98, 99, 211, 545]
.
----------------------------------------------------------------------
Ran 8 tests in 0.011s

OK
PS D:\WebDevelopment\circleci-demo>
```

CI/CD WorkFlow:

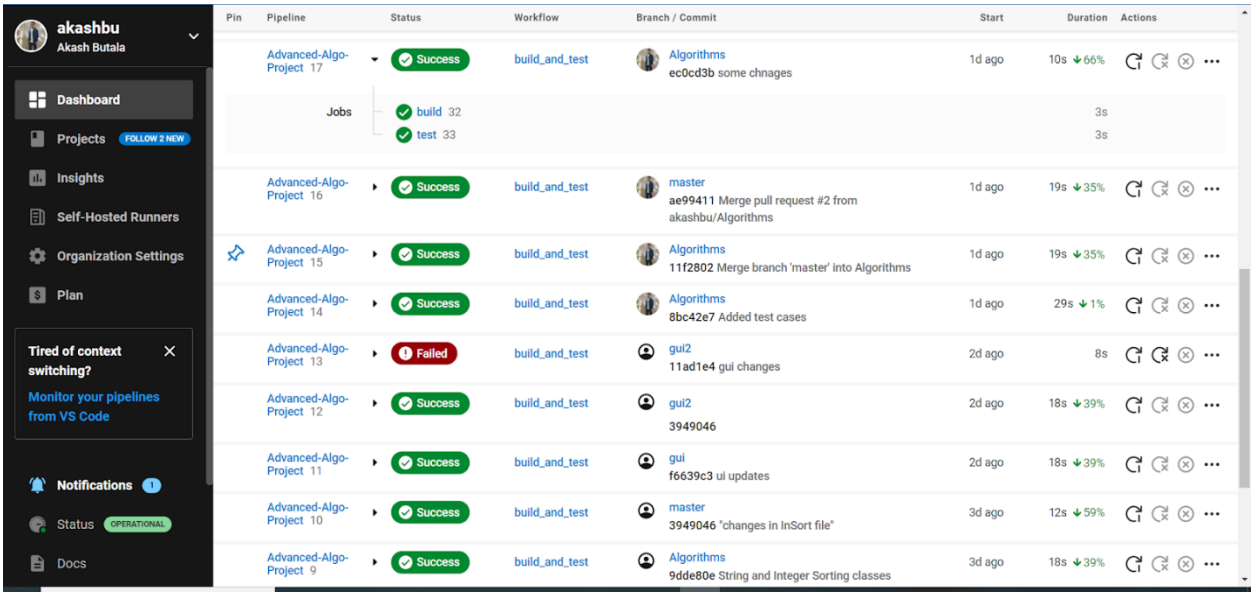
"config.yml" is a configuration file for CircleCI, a continuous integration and continuous deployment (CI/CD) platform. This configuration file specifies the workflow for building and testing our project.

The configuration defines two jobs: "build" and "test." The "build" job sets the working directory, specifies a Docker image with Python 3.6.4, checks out the project's source code, and runs the "main.py" script. This job likely handles building or preparing the project.

The "test" job also sets the working directory, uses the same Python 3.6.4 Docker image, checks out the source code, and runs the "IntSortTest.py" script. This job is responsible for running tests.

Lastly, there's a workflow called "build_and_test" that orchestrates these two jobs. It specifies that the "test" job requires the "build" job, ensuring that the code is built before running tests. This is a common practice in CI/CD pipelines to ensure that code is tested against the latest build.

Overall, this configuration file defines a basic CI/CD workflow , where code is built and then tested.



CHALLENGES FACED

1. Version Control:

One of the key challenges encountered during the development of our "Algorithm Efficiency Analyzer Tool" was effectively managing version control. With multiple team members working concurrently on the project, ensuring that code changes were integrated seamlessly and without conflicts posed a significant hurdle. To address this challenge, we adopted a clear branching and merging strategy within our Git repository. Regular team meetings and discussions were scheduled to coordinate version control efforts and resolve conflicts promptly. This challenge ultimately strengthened our team's proficiency in version control and collaborative development.

2. Code Integration Problem:

Integrating various components of the tool, including the GUI, algorithm implementations, data visualization, and testing suites, proved to be a complex undertaking. Ensuring that these diverse elements functioned harmoniously required meticulous planning and testing. We addressed this challenge by implementing a well-defined integration schedule, conductingY89 continuous integration testing, and maintaining open channels of communication among team members. This challenge highlighted the importance of robust integration practices and the need for seamless collaboration among different project modules.

3. Code Execution:

Another challenge we encountered was related to code execution, where specific portions of the code were functioning correctly while others remained blocked. This challenge was addressed through thorough debugging and code review sessions. By leveraging the expertise of team members and conducting systematic code inspections, we identified and resolved the issues causing code execution problems. This challenge underscored the importance of rigorous code testing and collaborative problem-solving in ensuring the overall functionality of our tool.

4. Documentation Consistency:

Maintaining consistent and up-to-date documentation throughout the project, including code comments, and documentation, can be challenging, especially when code evolves rapidly.

CONCLUSION

In this project "Algorithm Efficiency Analyzer Tool," our team embarked on a transformative exploration of algorithm efficiency, GUI design, data visualization, and collaborative software development. As we conclude this project, we reflect on the valuable lessons learned, the challenges overcome, and the unlocked achievements.

Through our well-structured methodology and the dedication of each team member, we successfully crafted a tool that empowers users to delve into the intricacies of sorting algorithms. Our project's significance lies not only in its practical utility but also in the skills honed during its development.

We navigated challenges with resilience, whether in the realm of version control, integration, or code execution. These challenges, far from being roadblocks, became steppingstones towards our growth as developers and problem solvers. We've learned the art of teamwork, communication, and adaptability, qualities essential in the professional landscape.

Our "Algorithm Efficiency Analyzer Tool" is a testament to what a cohesive team can accomplish. It brings complex concepts to the fingertips of users, fostering a deeper understanding of algorithmic principles and visualization techniques.

As we part ways with this project, we carry forward not only a powerful tool but also the knowledge and camaraderie that come from working together towards a common goal. We are excited to see how our creation will benefit others in their journeys of exploration and learning.

In the end, our project is not just lines of code; it's a reflection of our passion, dedication, and growth. We look forward to the next challenges and opportunities that lie ahead, armed with the experiences and skills gained from this project.

REFERENCES

1. https://www.productplan.com/glossary/bubble-sort/
2. https://codepumpkin.com/selection-sort-algorithms/
3. https://thinkdiff.net/insertion-sort-swift-db14b9a79016
4. https://codepumpkin.com/merge-sort-sorting-algorithm/
5. https://en.m.wikipedia.org/wiki/File:Quicksort-example.gif
6. https://docs.python.org/3/library/tkinter.html
7. https://matplotlib.org/stable/users/index.html