Sorting Algorithms – Design, Implementation, and Complexity Analysis

Agenda

- Algorithms.
- Code .
- Examples
- Analysis
- Graphs

Sorting Algorithms:

1-Insertion Sort

Idea:
Builds the final sorted list one element at a time. It picks each element and places it in the correct position among the already sorted elements before it.

Steps:
•Start from the second element.
•Compare it with elements before it.
•Shift larger elements one position to the right.
•Insert the current element in the correct spot.

Sorting Algorithms:

2. Quick Sort

Idea:
Divide and conquer. Choose a "pivot" element, then partition the array so that:
•Elements less than the pivot go left,
•Elements greater go right,
•Then recursively sort both parts.
Steps:
•Choose a pivot (e.g., last or random element).
•Partition the array into two halves.
•Recursively apply quicksort on each half.

Sorting Algorithms:

3. Merge Sort

Idea:
Also divide and conquer. Recursively divide the array into halves until single elements remain, then **merge** them in sorted order.
Steps:
•Split the array into halves.
•Recursively sort both halves.
•Merge the sorted halves into one sorted array.

Code

User Interface:

Python code using tkinter and matplotlib for visualization

```
class User_interface:
    def __init__(self, root):
        self.root = root
        self.algo = tk.StringVar()
        self.array_size = tk.IntVar(self.root, value=10)
        self.main_array = np.array([])

        self.root.title("Algorithms Visualizer")
        self.root.geometry("1024x720")
        self.root.resizable(False, False)
```

```
 self.main_frame = tk.Frame(self.root,
bg="black", padx=20, pady=20)
        self.main_frame.pack(fill=tk.BOTH,
expand=True)

        self.CreateLabels()
        self.CreateMenu()
        self.CreateSize()
        self.CreateButtons()
        self.draw_array(np.array([]))
```

Code

User Interface:

Python code using tkinter and matplotlib for visualization

```
def CreateLabels(self):
        tk.Label(self.main_frame, text="Algorithm Visualization", font=("Arial", 18, "bold"),
                fg="#000080", bg="black").pack(pady=7)
        tk.Label(self.main_frame, text="Select an Algorithm", font=("Arial", 16, "bold"),
                fg="#000080", bg="black").pack(pady=15, padx=200)

    def CreateMenu(self):
        algorithms = ["Insertion Sort", "Quick Sort", "Merge Sort"]
        algo_menu = ttk.Combobox(self.main_frame, textvariable=self.algo,
                                 values=algorithms, state="readonly", width=70)
        algo_menu.pack(pady=10, padx=200)
        algo_menu.bind("<<ComboboxSelected>>", self.CreateMessage)

    def CreateMessage(self, event=None):
        selected = self.algo.get()
        messagebox.showinfo("Algorithm Selected", f"You chose: {selected}")

", command=self.run_algorithm).place(x=420, y=320)
```

Code

User Interface:

Different types of arrays created

```
def Create_sorted(self):
        self.main_array = np.arange(self.array_size.get())
        self.draw_array(self.main_array)

    def Create_reversed(self):
        self.main_array = np.arange(self.array_size.get())[::-1]
        self.draw_array(self.main_array)

    def Create_random(self):
        self.main_array = np.random.randint(1, 100, self.array_size.get())
        self.draw_array(self.main_array)
```

Code

User Interface:

User can define Size and click button for the type of array (sorted, reversed , random)

```
    def CreateSize(self):
        tk.Label(self.main_frame, text="Array Size:", font=("Arial", 14), fg="white",
bg="black").place(x=400, y=270)
        spinbox = tk.Spinbox(self.main_frame, from_=5, to=100, textvariable=self.array_size, width=10,
font=("Arial", 12))
        spinbox.place(x=500, y=270)

    def CreateButtons(self):
        tk.Button(self.main_frame, text="Sorted", font=("Arial", 12, "bold"),
                  bg="orange", fg="#000080", command=self.Create_sorted).place(x=250, y=200)
        tk.Button(self.main_frame, text="Reversed", font=("Arial", 12, "bold"),
                  bg="orange", fg="#000080", command=self.Create_reversed).place(x=400, y=200)
        tk.Button(self.main_frame, text="Random", font=("Arial", 12, "bold"),
                  bg="orange", fg="#000080", command=self.Create_random).place(x=550, y=200)
        tk.Button(self.main_frame, text="Run Algorithm", font=("Arial", 12, "bold"),
                  bg="green", fg="white", command=self.run_algorithm).place(x=420, y=320)
```

Code

User Interface:

Drawing array for visualizing how an algorithm works

```
def draw_array(self, array, color='skyblue'):
        if hasattr(self, 'canvas_widget'):
            self.canvas_widget.get_tk_widget().destroy()

        fig = Figure(figsize=(9.5, 3.5), dpi=100)
        ax = fig.add_subplot(111)
        ax.bar(range(len(array)), array, color=color)
        ax.set_title("Sorting Visualization")
        ax.set_xlabel("Index")
        ax.set_ylabel("Value")
```

```
 # Set x-ticks intelligently
        if len(array) <= 20:
            ax.set_xticks(range(len(array)))
        else:
            step = len(array) // 10 or 1
            ax.set_xticks(range(0, len(array), step))

        ax.tick_params(axis='x', rotation=45)
        fig.tight_layout()

        self.canvas_widget = FigureCanvasTkAgg(fig,
master=self.main_frame)
        self.canvas_widget.draw()
        self.canvas_widget.get_tk_widget().place(x=70,
y=380)
```

Code

User Interface:

Running the algorithm chosen

```
def run_algorithm(self):
        algo = self.algo.get()
        if len(self.main_array) == 0:
            messagebox.showerror("Error", "Please generate an array first.")
            return

        if algo == "Insertion Sort":
            self.insertion_sort(self.main_array)
        elif algo == "Quick Sort":
            self.quick_sort(0, len(self.main_array) - 1)
            self.draw_array(self.main_array, color='green')
        elif algo == "Merge Sort":
            self.merge_sort(0, len(self.main_array) - 1)
            self.draw_array(self.main_array, color='green')
        else:
            messagebox.showwarning("Warning", "No algorithm selected.")
```

Code

Insertion Sort

```
def insertion_sort(self, arr):
        for i in range(1, len(arr)):
            key = arr[i]
            j = i - 1
            while j >= 0 and key < arr[j]:
                arr[j + 1] = arr[j]
                j -= 1
                self.draw_array(arr, color='orange')
                self.root.update()
                time.sleep(0.1)
            arr[j + 1] = key
            self.draw_array(arr, color='green')
            self.root.update()
            time.sleep(0.1)
```

Code

Quick Sort

```
def quick_sort(self, low, high):
    if low < high:
        pi = self.partition(low, high)
        self.quick_sort(low, pi - 1)
        self.quick_sort(pi + 1, high)
```

```
def partition(self, low, high):
    arr = self.main_array
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
            self.draw_array(arr, color='red')
            self.root.update()
            time.sleep(0.1)
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    self.draw_array(arr, color='green')
    self.root.update()
    time.sleep(0.1)
    return i + 1
```

Code

Merge Sort

```
def merge_sort(self, left, right):
        if left < right:
            mid = (left + right) // 2
            self.merge_sort(left, mid)
            self.merge_sort(mid + 1, right)
            self.merge(left, mid, right)
 def merge(self, left, mid, right):
        arr = self.main_array
        L = arr[left:mid + 1]
        R = arr[mid + 1:right + 1]

        i = j = 0
        k = left
```

```
while i < len(L) and j < len(R):
            if L[i] <= R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            self.draw_array(arr, color='purple')
            self.root.update()
            time.sleep(0.1)
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
            self.draw_array(arr, color='blue')
            self.root.update()
            time.sleep(0.1)
```

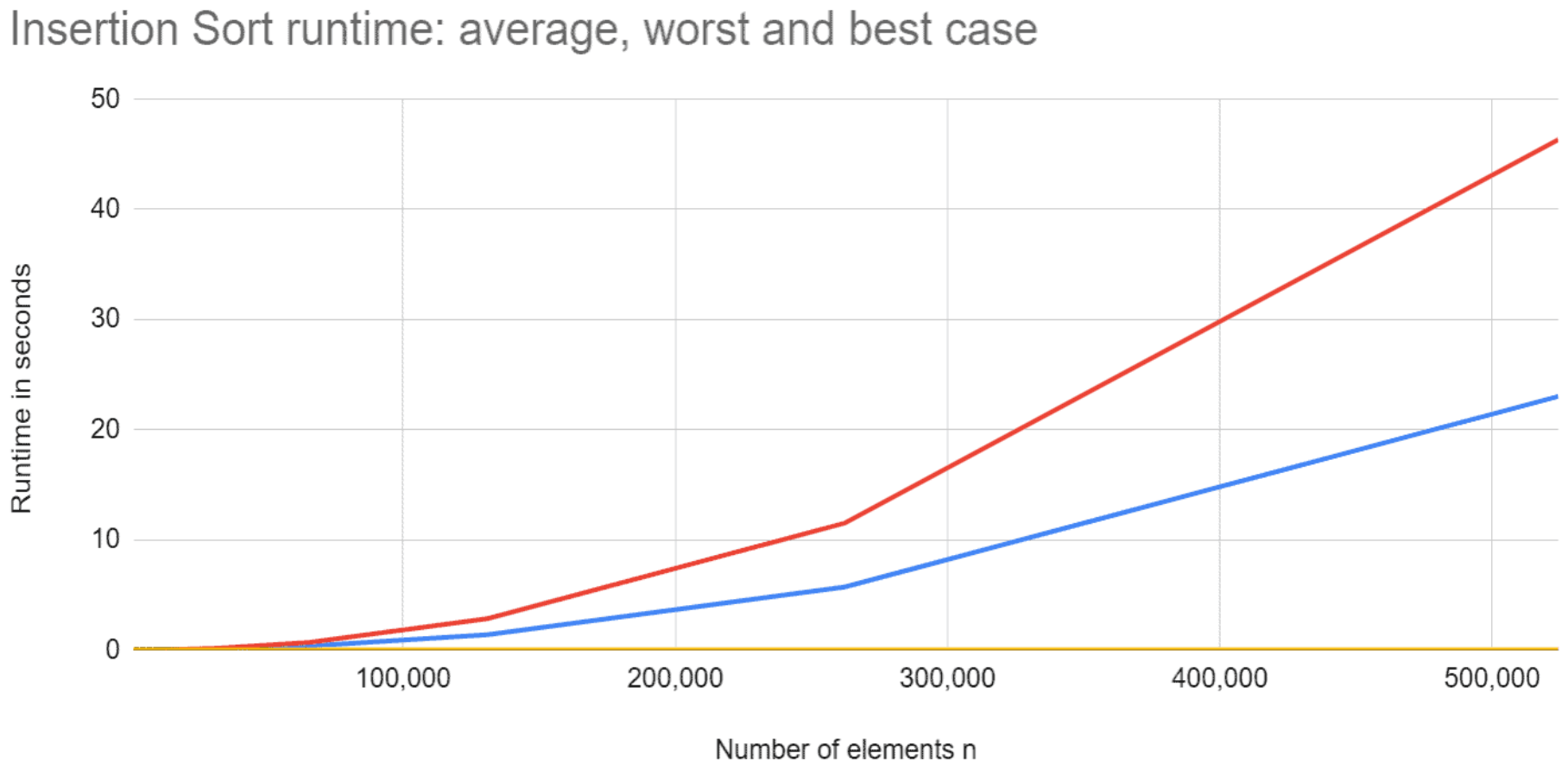
```
while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
            self.draw_array(arr,
color='blue')
            self.root.update()
            time.sleep(0.1)
```

How each sort works is shown in the code Visualization

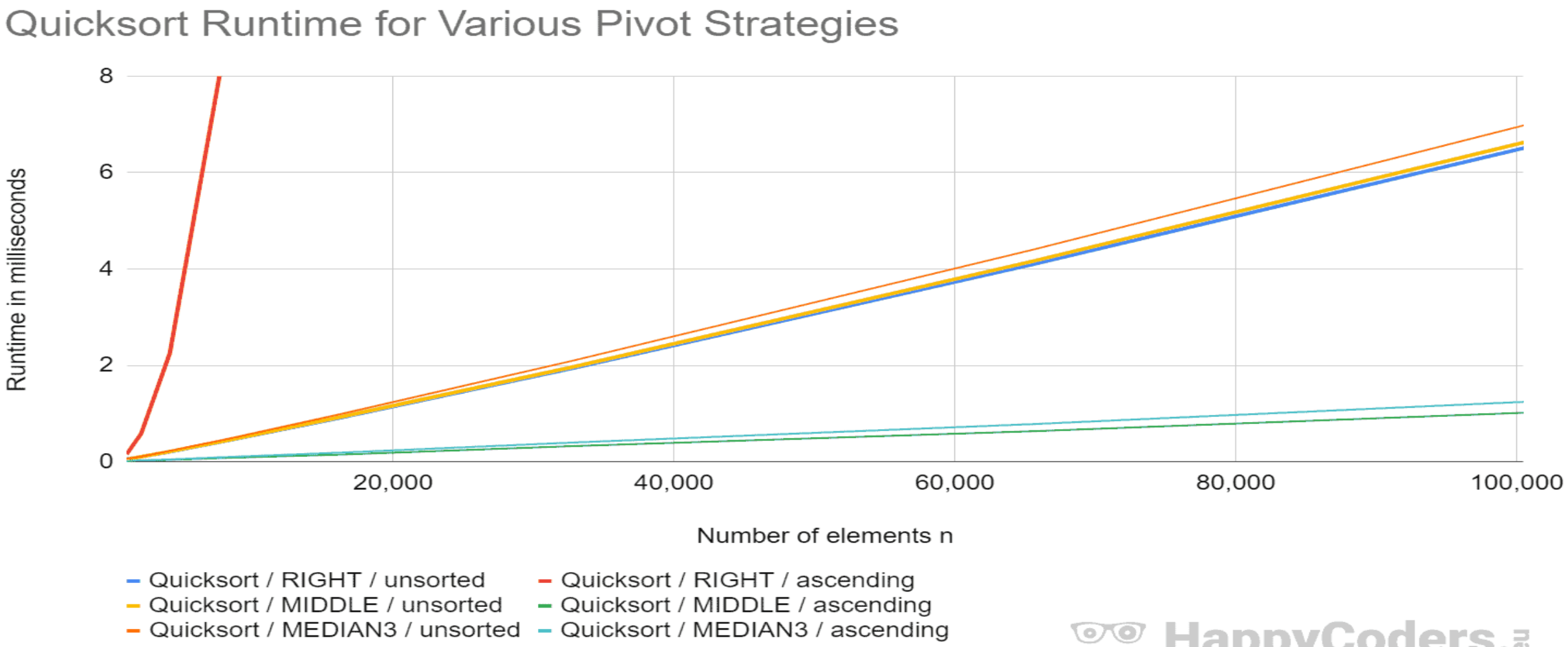
Analysis

Algorithm	Best Case	Average Case	Worst Case	Space Complexity	Stab
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n) (in-place)	No
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes

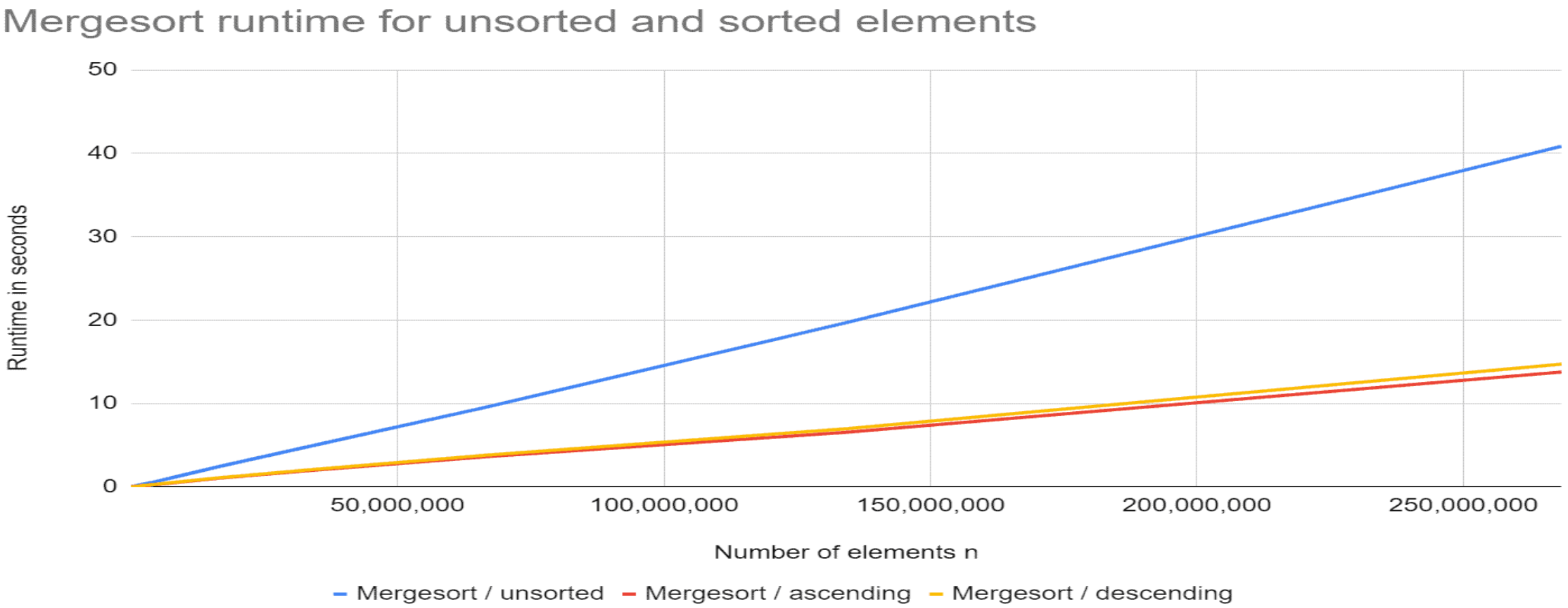
Insertion Sort Graph



Quick Sort Graph



Merge Sort Graph



Thank You

Prepared by :
Mariam Ahmed Gahreeb
192200346

Supervised by:
DR Rasha Saleh
Eng Mohamed Tarek