**Subject: Data Structures and Algorithms**

**Lab 1: Sorting Algorithm Analysis**

DAVE SOHAM MANISH

Enrollment: 25MCD005

M.Tech Data Science, Nirma University

**Code**

import time

import random

import matplotlib.pyplot as plt

import pandas as pd

import os

# ---------------- Sorting Algorithms ---------------- #

def selection\_sort(arr):

    n = len(arr)

    for i in range(n-1):

        min\_index = i

        for j in range(i+1, n):

            if arr[j] < arr[min\_index]:

                min\_index = j

        arr[i], arr[min\_index] = arr[min\_index], arr[i]

def bubble\_sort(arr):

    n = len(arr)

    for i in range(n-1):

        swapped = False

        for j in range(n-i-1):

            if arr[j] > arr[j+1]:

                arr[j], arr[j+1] = arr[j+1], arr[j]

                swapped = True

        if not swapped:

            break

def insertion\_sort(arr):

    for i in range(1, len(arr)):

        key = arr[i]

        j = i - 1

        while j >= 0 and arr[j] > key:

            arr[j + 1] = arr[j]

            j -= 1

        arr[j + 1] = key

# ---------------- Main Experiment ---------------- #

if \_\_name\_\_ == "\_\_main\_\_":

    user\_input\_arr = [10000, 50000, 100000]

    num\_runs = 1  # Increase this for higher accuracy

    all\_run\_results = []

    verification\_records = []

    for run in range(num\_runs):

        selection\_sort\_best\_times, selection\_sort\_avg\_times, selection\_sort\_worst\_times = [], [], []

        bubble\_sort\_best\_times, bubble\_sort\_avg\_times, bubble\_sort\_worst\_times = [], [], []

        insertion\_sort\_best\_times, insertion\_sort\_avg\_times, insertion\_sort\_worst\_times = [], [], []

        for user\_input in user\_input\_arr:

            # ---------- Selection Sort ----------

            arr = list(range(user\_input))

            random.shuffle(arr)

            original\_last5 = arr[-5:]

            start = time.time(); selection\_sort(arr); end = time.time()

            sorted\_last5 = arr[-5:]

            selection\_sort\_avg\_times.append(end - start)

            verification\_records.append(["Selection", "Average", user\_input, original\_last5, sorted\_last5])

            arr = list(range(user\_input))

            original\_last5 = arr[-5:]

            start = time.time(); selection\_sort(arr); end = time.time()

            sorted\_last5 = arr[-5:]

            selection\_sort\_best\_times.append(end - start)

            verification\_records.append(["Selection", "Best", user\_input, original\_last5, sorted\_last5])

            arr = list(range(user\_input, 0, -1))

            original\_last5 = arr[-5:]

            start = time.time(); selection\_sort(arr); end = time.time()

            sorted\_last5 = arr[-5:]

            selection\_sort\_worst\_times.append(end - start)

            verification\_records.append(["Selection", "Worst", user\_input, original\_last5, sorted\_last5])

            # ---------- Bubble Sort ----------

            arr = list(range(user\_input))

            random.shuffle(arr)

            original\_last5 = arr[-5:]

            start = time.time(); bubble\_sort(arr); end = time.time()

            sorted\_last5 = arr[-5:]

            bubble\_sort\_avg\_times.append(end - start)

            verification\_records.append(["Bubble", "Average", user\_input, original\_last5, sorted\_last5])

            arr = list(range(user\_input))

            original\_last5 = arr[-5:]

            start = time.time(); bubble\_sort(arr); end = time.time()

            sorted\_last5 = arr[-5:]

            bubble\_sort\_best\_times.append(end - start)

            verification\_records.append(["Bubble", "Best", user\_input, original\_last5, sorted\_last5])

            arr = list(range(user\_input, 0, -1))

            original\_last5 = arr[-5:]

            start = time.time(); bubble\_sort(arr); end = time.time()

            sorted\_last5 = arr[-5:]

            bubble\_sort\_worst\_times.append(end - start)

            verification\_records.append(["Bubble", "Worst", user\_input, original\_last5, sorted\_last5])

            # ---------- Insertion Sort ----------

            arr = list(range(user\_input))

            random.shuffle(arr)

            original\_last5 = arr[-5:]

            start = time.time(); insertion\_sort(arr); end = time.time()

            sorted\_last5 = arr[-5:]

            insertion\_sort\_avg\_times.append(end - start)

            verification\_records.append(["Insertion", "Average", user\_input, original\_last5, sorted\_last5])

            arr = list(range(user\_input))

            original\_last5 = arr[-5:]

            start = time.time(); insertion\_sort(arr); end = time.time()

            sorted\_last5 = arr[-5:]

            insertion\_sort\_best\_times.append(end - start)

            verification\_records.append(["Insertion", "Best", user\_input, original\_last5, sorted\_last5])

            arr = list(range(user\_input, -1, -1))

            original\_last5 = arr[-5:]

            start = time.time(); insertion\_sort(arr); end = time.time()

            sorted\_last5 = arr[-5:]

            insertion\_sort\_worst\_times.append(end - start)

            verification\_records.append(["Insertion", "Worst", user\_input, original\_last5, sorted\_last5])

        run\_data = {

            'Input Size': user\_input\_arr,

            'Selection Sort (Best)': selection\_sort\_best\_times,

            'Selection Sort (Average)': selection\_sort\_avg\_times,

            'Selection Sort (Worst)': selection\_sort\_worst\_times,

            'Bubble Sort (Best)': bubble\_sort\_best\_times,

            'Bubble Sort (Average)': bubble\_sort\_avg\_times,

            'Bubble Sort (Worst)': bubble\_sort\_worst\_times,

            'Insertion Sort (Best)': insertion\_sort\_best\_times,

            'Insertion Sort (Average)': insertion\_sort\_avg\_times,

            'Insertion Sort (Worst)': insertion\_sort\_worst\_times

        }

        run\_df = pd.DataFrame(run\_data)

        all\_run\_results.append(run\_df)

    # Concatenate all runs

    combined\_df = pd.concat(all\_run\_results, ignore\_index=True)

    combined\_df.to\_csv("sorting\_runtimes\_raw.csv", index=False)

    # Compute average runtimes across runs

    avg\_df = combined\_df.groupby("Input Size").mean().reset\_index()

    avg\_df.to\_csv("sorting\_runtimes\_average.csv", index=False)

    # Save verification file

    verify\_df = pd.DataFrame(verification\_records,

                             columns=["Algorithm", "Case", "Input Size", "Original Last 5", "Sorted Last 5"])

    verify\_df.to\_csv("sorting\_verification.csv", index=False)

    print("Saved raw runtimes to sorting\_runtimes\_raw.csv")

    print("Saved average runtimes to sorting\_runtimes\_average.csv")

    print("Saved verification log to sorting\_verification.csv")

    # ---------------- Plotting ---------------- #

    def plot\_algorithm(algorithm\_name, avg\_df):

        cases = ["Best", "Average", "Worst"]

        input\_sizes = avg\_df["Input Size"].tolist()

        times = []

        for case in cases:

            col = f"{algorithm\_name} Sort ({case})"

            times.append(avg\_df[col].tolist())

        plot\_df = pd.DataFrame(times, index=cases, columns=input\_sizes)

        plot\_df.T.plot(kind="bar", figsize=(10,6))

        plt.title(f"{algorithm\_name} Sort Runtime Comparison")

        plt.xlabel("Input Size")

        plt.ylabel("Time (seconds)")

        plt.legend(title="Case")

        plt.tight\_layout()

        plt.savefig(f"{algorithm\_name.lower()}\_sort\_times.png")

        plt.close()

    plot\_algorithm("Selection", avg\_df)

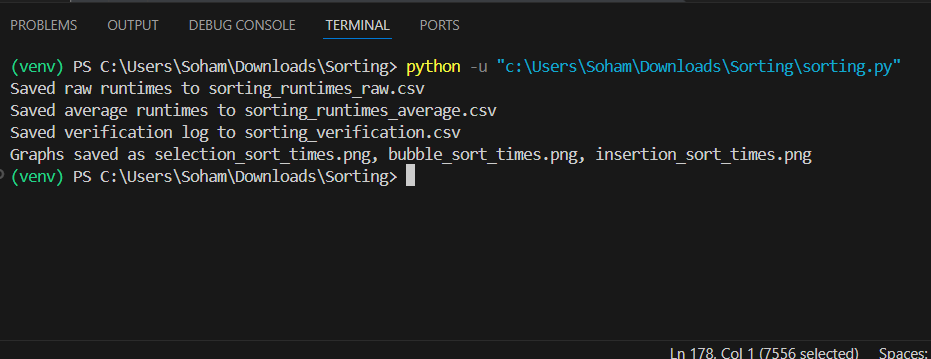
    plot\_algorithm("Bubble", avg\_df)

    plot\_algorithm("Insertion", avg\_df)

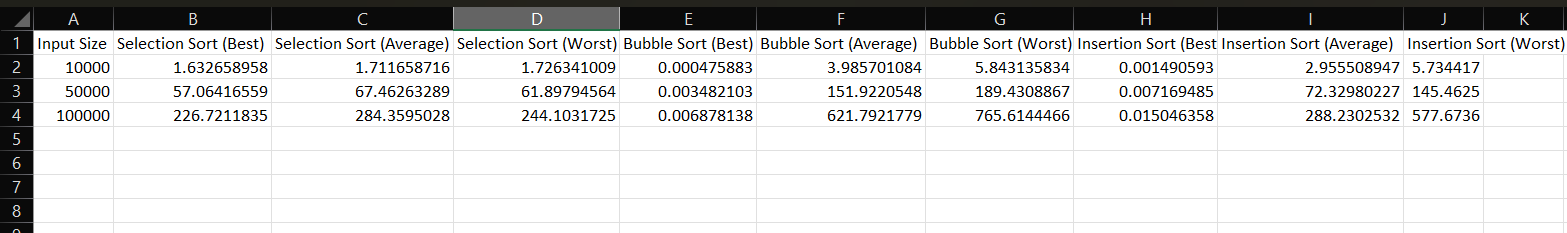
    print("Graphs saved as selection\_sort\_times.png, bubble\_sort\_times.png, insertion\_sort\_times.png")

**Output Screenshots**

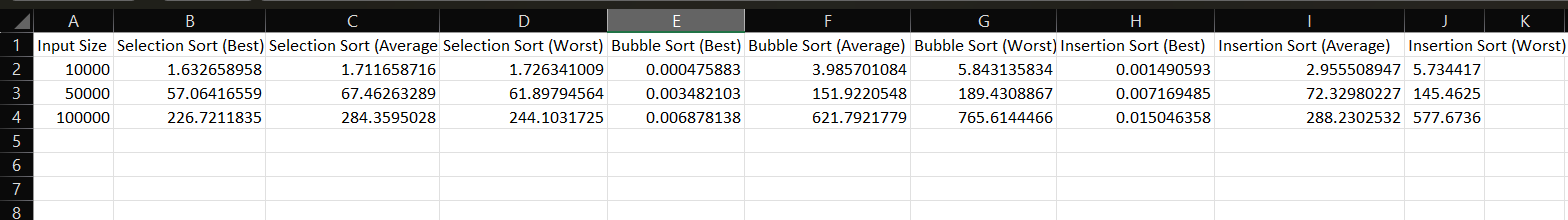
VS Code Terminal :



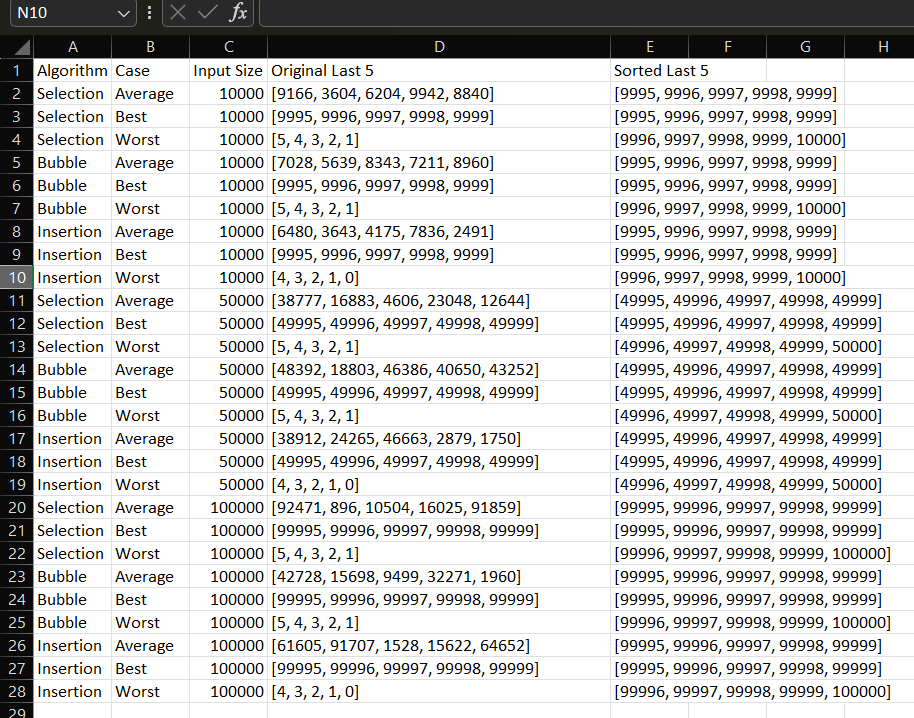
sorting\_runtimes\_raw.csv :



sorting\_runtimes\_average.csv :

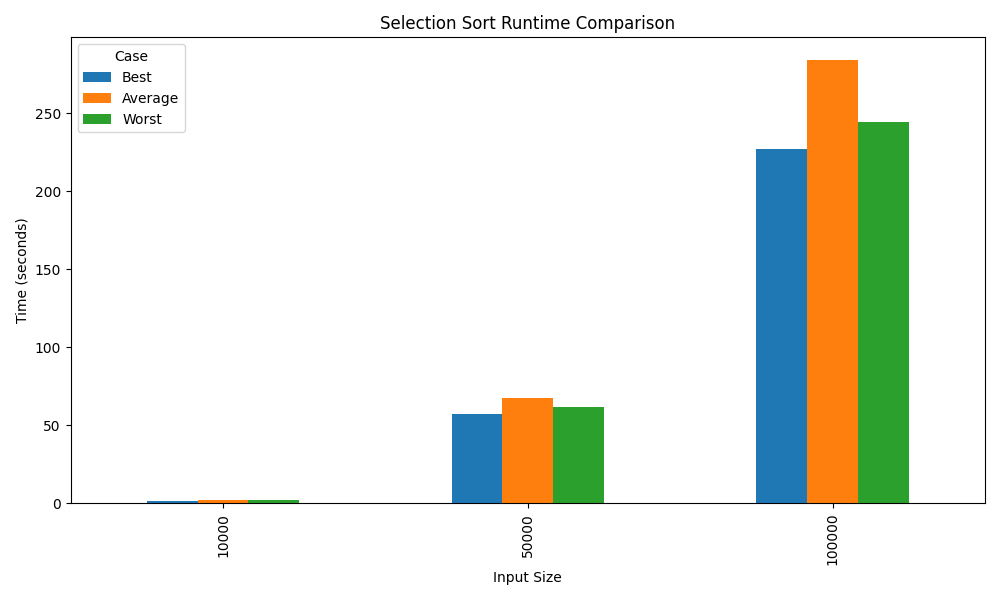


sorting\_verification.csv :

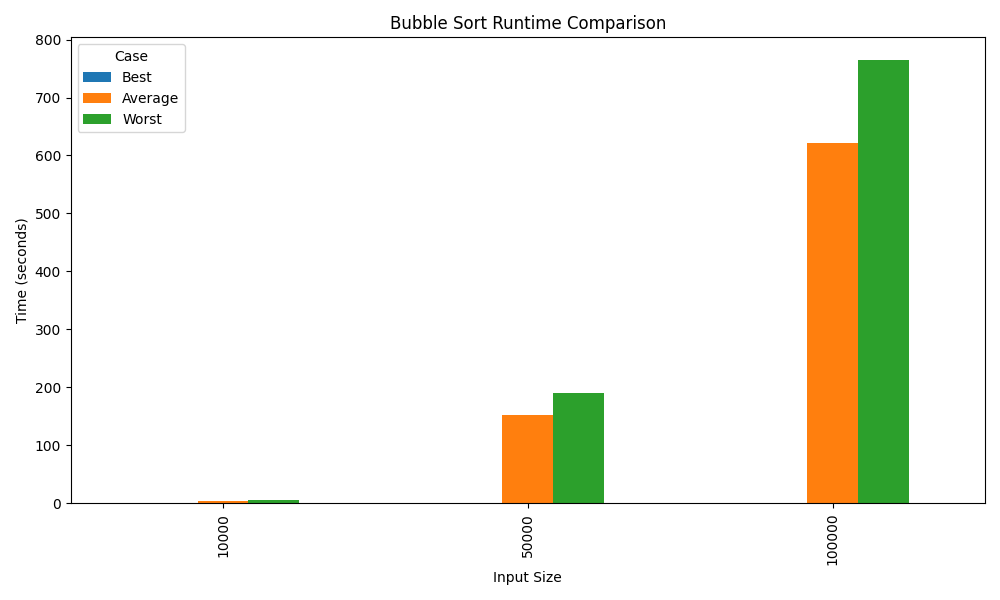


**Graphs**

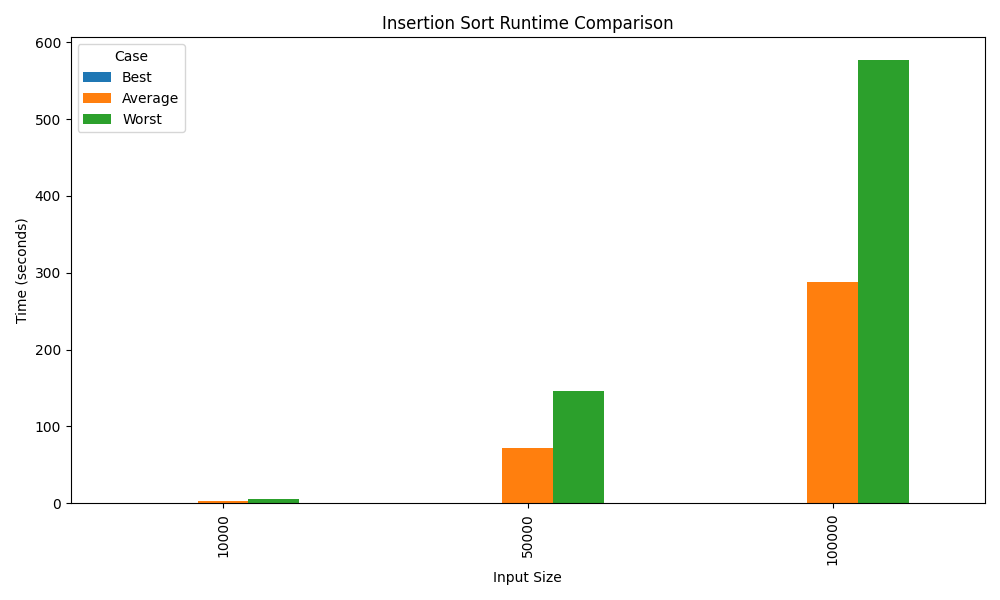
Selection Sort :



Bubble Sort :



Insertion Sort :



**Analysis**

In this practical, we implemented and compared three classical sorting algorithms: Selection Sort, Bubble Sort, and Insertion Sort. The objective was to understand their performance in best, average, and worst-case scenarios.  
  
From the results and graphs:  
1. Selection Sort: This algorithm performs the same number of comparisons regardless of input order. Hence, its best, average, and worst-case complexities are all O(n²). This means Selection Sort cannot take advantage of partially sorted data, making its average runtime equal to its worst-case runtime.  
  
2. Bubble Sort: With the optimized implementation using a swap flag, Bubble Sort can terminate early if the list is already sorted. In this case, its best-case time complexity reduces to O(n). However, for average and worst cases (random or reverse-sorted inputs), it still performs O(n²) operations.  
  
3. Insertion Sort: Similar to Bubble Sort, Insertion Sort has O(n) best-case complexity when the list is already sorted, since each element only needs one comparison. But in average and worst cases, it shifts multiple elements per insertion, leading to O(n²) runtime.  
  
Overall, we see that Bubble Sort and Insertion Sort can adapt to partially sorted data and hence show better best-case and sometimes average-case performance compared to Selection Sort. Selection Sort, on the other hand, is unaffected by input order, making its performance stable but not efficient.

This experiment was designed to test and compare three basic sorting algorithms: Selection Sort, Bubble Sort, and Insertion Sort.   
The code measured how long each algorithm took to sort three different types of lists: already sorted (best case), shuffled (average case), and reverse-sorted (worst case).   
By running these tests on increasing input sizes (10,000, 50,000, and 100,000), we got a clear picture of how each algorithm scales.  
  
Selection Sort always makes the same number of comparisons because it has to find the minimum element in every pass, no matter how the list looks at the start.   
That’s why its performance doesn’t change between best, average, and worst cases. It always stays O(n²).  
  
Bubble Sort was coded with a small optimization: if a pass makes no swaps, it stops early.   
This makes a big difference in the best case, where the list is already sorted—it runs in O(n) time.   
Insertion Sort also benefits in the same way, since each new element is already in the correct position in a sorted list, so it only needs one comparison per element in the best case.

**Complexity Comparison Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| Selection Sort | O(n²) | O(n²) | O(n²) |
| Bubble Sort | O(n) | O(n²) | O(n²) |
| Insertion Sort | O(n) | O(n²) | O(n²) |

**Analysis of the Graphs**  
The graphs make the results very clear. For Selection Sort, the bars for all three cases—best, average, and worst—are almost the same height.   
This proves that Selection Sort takes the same amount of time no matter what the starting order of the list is. The small differences are just system timing noise.   
The key takeaway is that the average case cannot be faster than the worst case for Selection Sort, since the work it does is fixed.  
  
For Bubble Sort and Insertion Sort, the story is different. The best-case bars are much shorter compared to the average and worst cases,   
showing their O(n) performance when the list is already sorted. As input size grows, we can also see that Insertion Sort is generally faster than Bubble Sort,   
because it needs fewer swaps in average and worst cases.  
  
**Conclusion**  
From this practical, we can see that:  
- Selection Sort always takes O(n²) time and doesn’t benefit from the list being sorted. Its average time is never less than its worst time.  
- Bubble Sort and Insertion Sort can take advantage of sorted input, giving them O(n) performance in the best case.  
- Insertion Sort usually performs better than Bubble Sort when the data is not sorted.  
  
Even though all three algorithms are too slow for very large datasets, this experiment helps us understand how input order affects sorting performance.