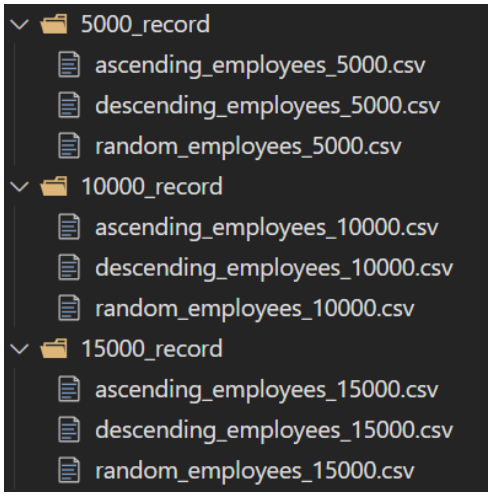
Lab 2 Sorting Analysis

23BAI1229

SHOBHIT SINGH

First Generate Employee Data using python and store them



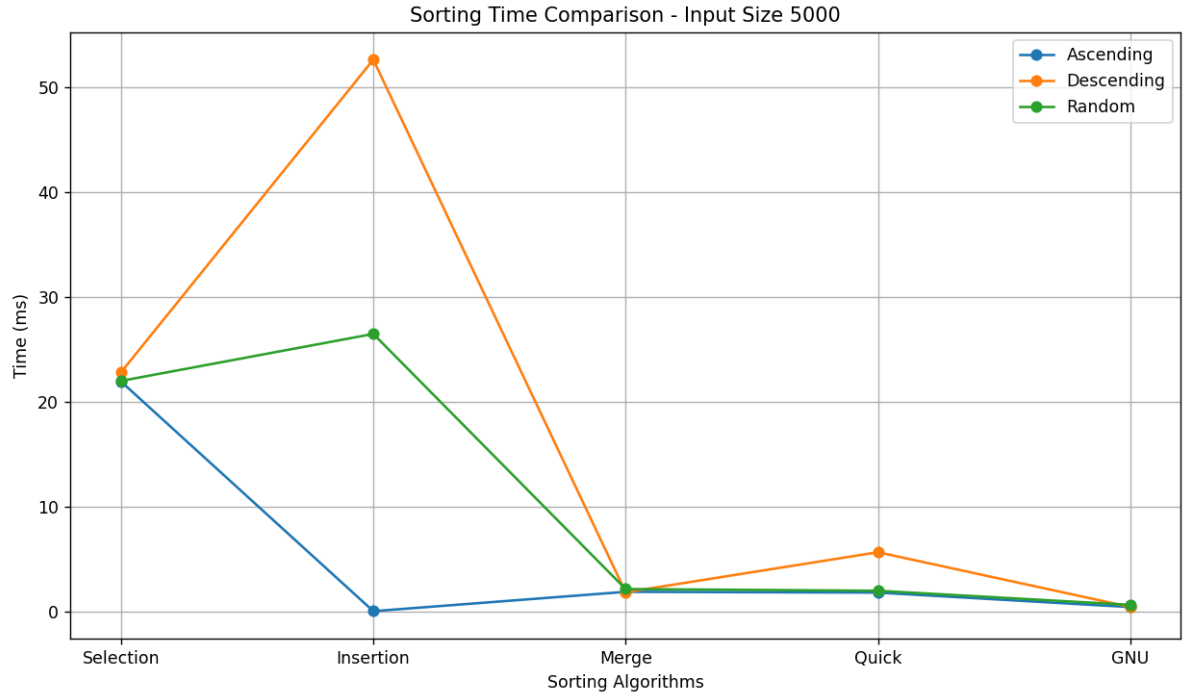
Looks something like this

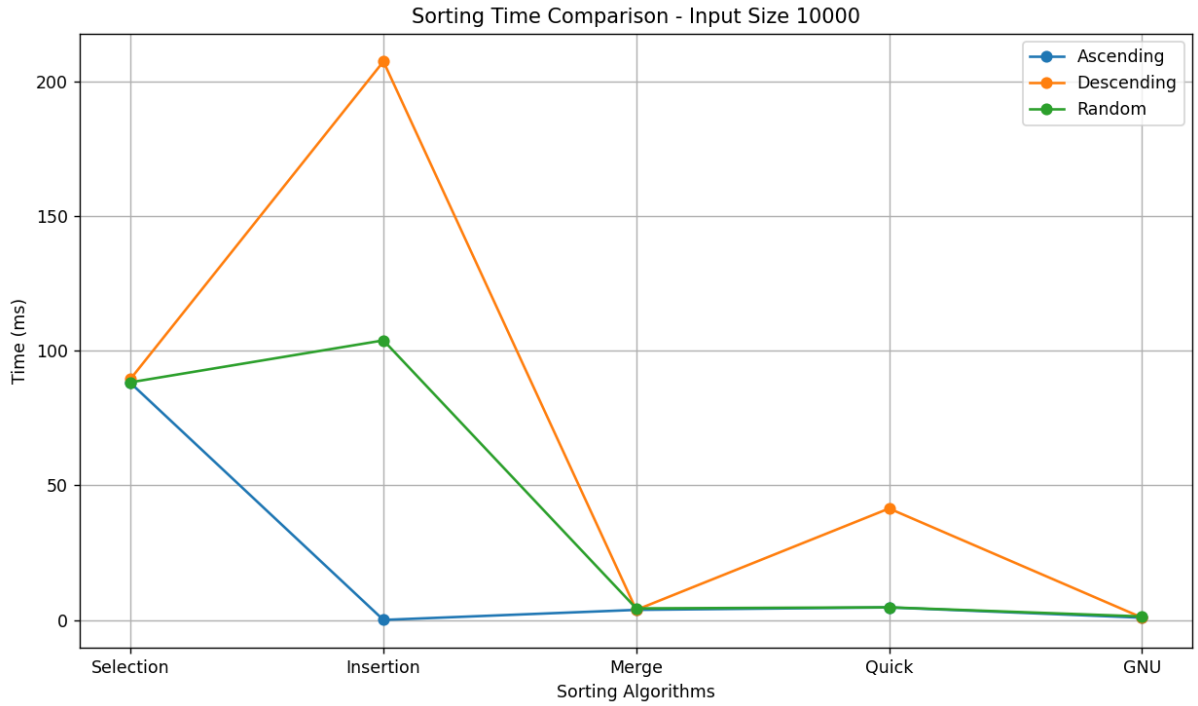
```
empID,Name,Salary
0,Becky Nguyen,61111
1,Eric Mccoy,50000
2,Melissa Davis,112635
3,Shannon Hoffman,30000
4,William Smith,140000
```

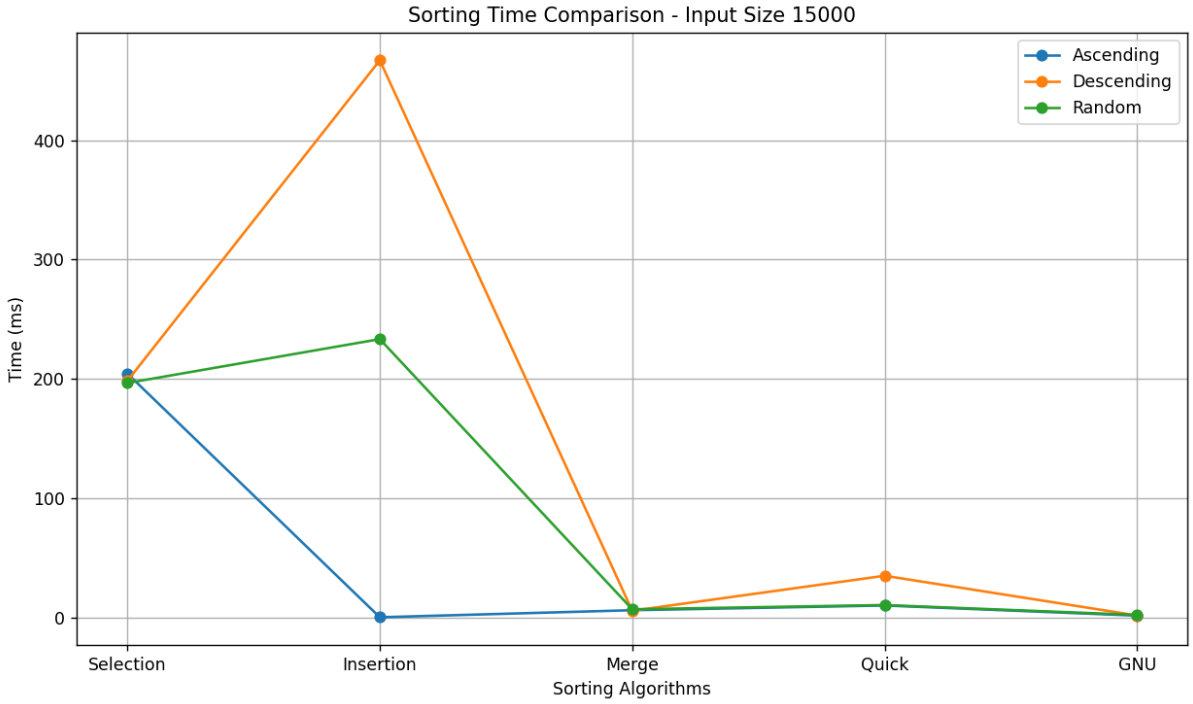
Now sorting all of them in C++ and noting down the time.

```
C:\Users\Shobhit\Desktop\Death by code\Github Repos Examples\Sorting-Graphs-using-C-and-python>python run_this_file.py

Results for 5000 elements:
Data Type   Selection      Insertion      Merge          Quick          GNU Sort
Ascending   21787          22             1870           1350           393
Descending  22533          51811          1799           8238           423
Random      22018          26097          2067           1370           619
---------------------------------------------------------------------

Results for 10000 elements:
Data Type   Selection      Insertion      Merge          Quick          GNU Sort
Ascending   87299          45             3762           4685           875
Descending  88531          208968         3755           27550          923
Random      87857          103023         4246           4728           1332
---------------------------------------------------------------------

Results for 15000 elements:
Data Type   Selection      Insertion      Merge          Quick          GNU Sort
Ascending   196732         68             6105           10036          1443
Descending  199380         470400         5738           84343          1523
Random      196285         231707         6530           10140          2051
---------------------------------------------------------------------
```

Graphs


Sorting Time Comparison - Input Size 5000
Ascending
Descending
Random
Time (ms)
0
10
20
30
40
50
Selection
Insertion
Merge
Quick
GNU
Sorting Algorithms


Sorting Time Comparison - Input Size 10000
Ascending
Descending
Random
Time (ms)
0
50
100
150
200
Selection
Insertion
Merge
Quick
GNU
Sorting Algorithms



Conclusion –

Trend Analysis & Alignment with Theoretical Complexities

Selection Sort ($O(n^2)$)

- Consistently slow, regardless of data type.
- Time increases significantly with input size.
- Not adaptive to data order.

Insertion Sort ($O(n^2)$, best case $O(n)$)

- **Extremely fast** for ascending data: works in almost linear time (best-case).
- **Extremely slow** for descending data: worst-case performance visible.
- **Performance highly sensitive** to input order.

Merge Sort ($O(n \log n)$)

- Stable, predictable performance across all inputs.
- Handles all types of data uniformly well.
- Slight time increase with input size is as expected.

Quick Sort (Avg: O(n log n), Worst: O(n^2))

- Performs very well on ascending/random data.
- **Degraded performance on descending input** due to poor pivot choice (expected worst-case).
- Still competitive overall.

GNU Sort (std::sort, Introsort-based)

- **Fastest in almost all cases**.
- Uses hybrid of QuickSort, HeapSort, and InsertionSort.
- Adapts to input type and size efficiently.

Conclusion

- **Insertion Sort** is best for nearly-sorted data but poor for others.
- **Merge Sort** is stable and consistent.
- **Quick Sort** is fast on average but risky without pivot strategy.
- **GNU Sort** is the most optimized in practice.
- **Selection Sort** is inefficient and mainly for educational/demo purposes.