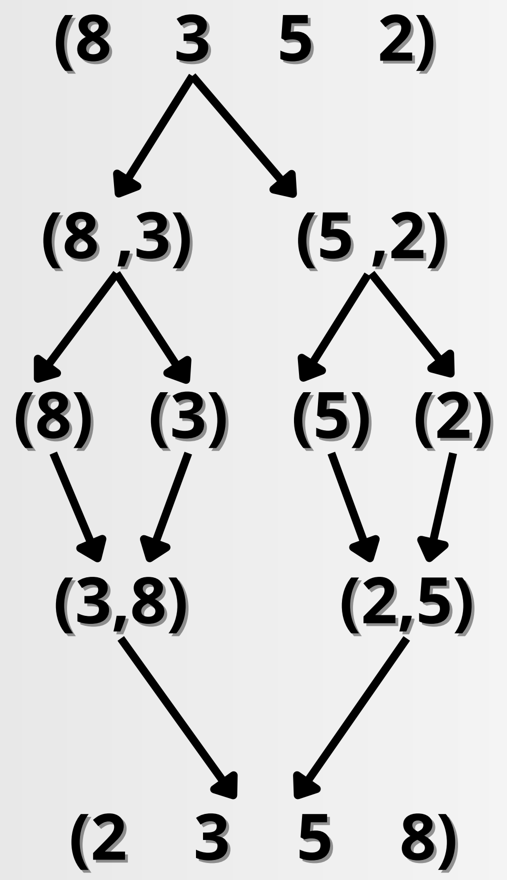
DIVIDE AND CONQUER

MERGE SORT CASE STUDY

Data Structure and Algorithm



(8 3 5 2)

(8 ,3) (5 ,2)

(8) (3) (5) (2)

(3,8) (2,5)

(2 3 5 8)

Name :-

Harshwardhan karanje

Harshal Magar

Uday Barapatre

Snehankit Zhore

Sorting

Sorting

Sor

Sorting in Computer Science

- Merge Sort always runs in $O(n \log n)$
- Works in best, average, and worst case
- Used in databases, searching, data analysis

Background of Merge Sort

- Inventor: John von Neumann (1945)
- Method: Divide & Conquer
- Stable: Maintains order of equal elements
- Needs O(n) extra space
- Scalable for large datasets

Problem Statement

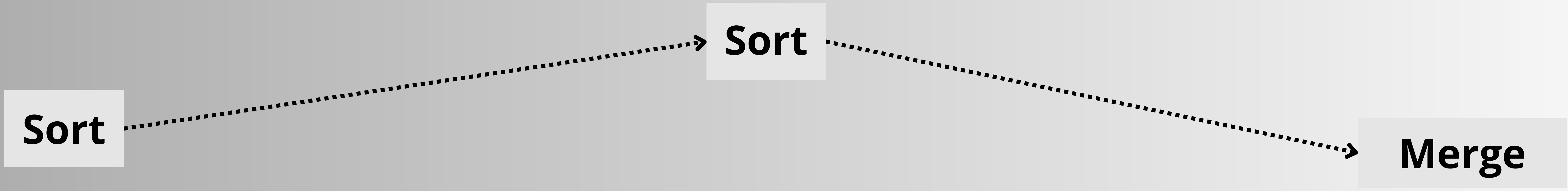
- Company has unsorted scores: [38, 27, 43, 3, 9, 82, 10]
- Sorting needed for reports
- Bubble/Insertion Sort too slow on large data
- Merge Sort chosen for efficiency

Working of Merge Sort

1. **Divide → Split array into halves**
2. **Conquer → Sort sub-arrays recursively**
3. **Merge → Combine sorted arrays**
4. **Example: [38, 27] → [27, 38]**

Pseudocode

```
• function mergeSort(arr, left, right):
•   if left < right:
•     mid = (left+right)/2
•     mergeSort(arr, left, mid)
•     mergeSort(arr, mid+1, right)
•     merge(arr, left, mid, right)
```



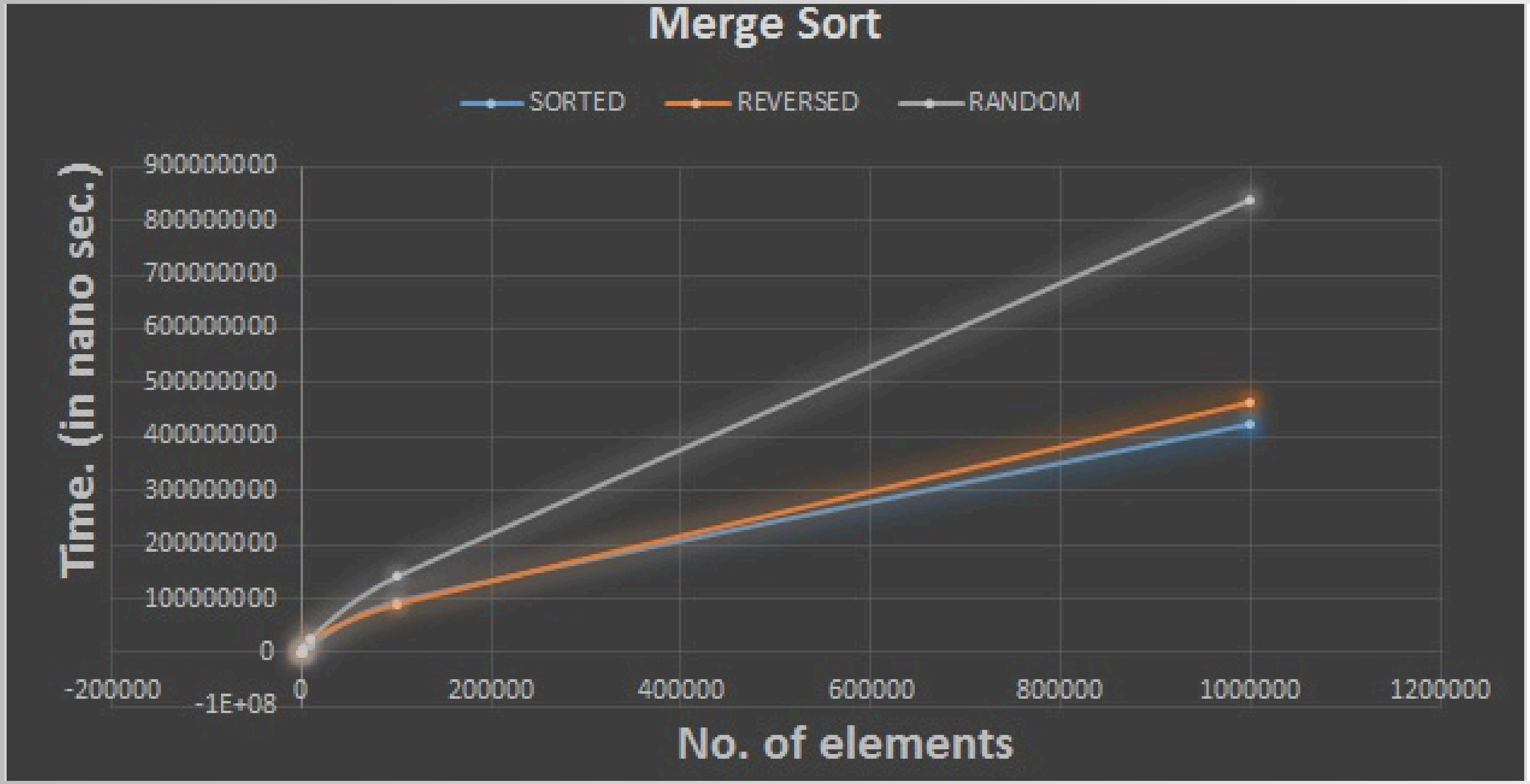
Sort

Sort

Merge

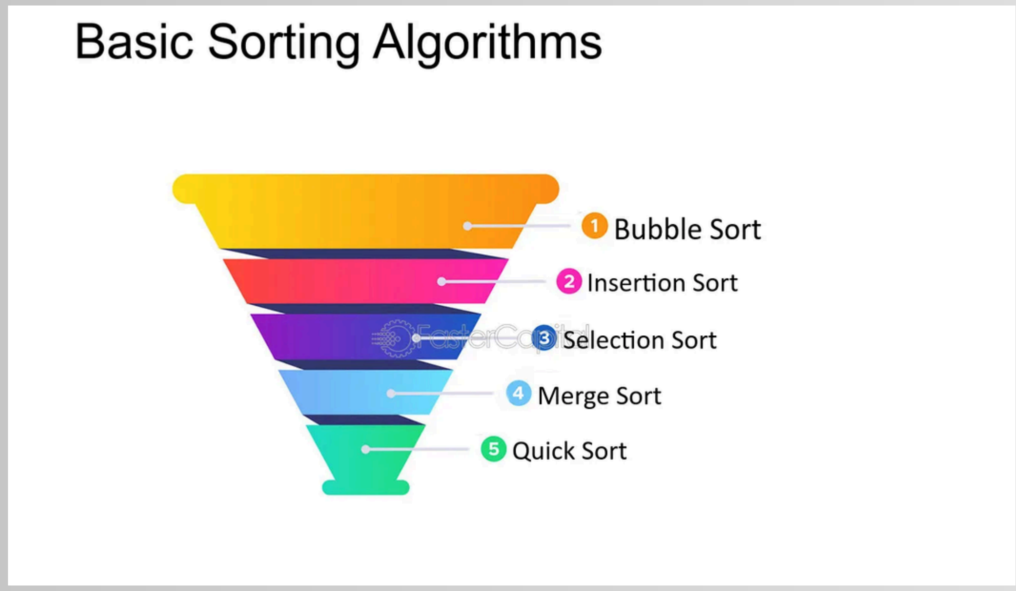
Runtime And Space Analysis

- Time Complexity: O(n log n)
- Divide → log(n) levels
- Merge → O(n) work at each level
- Space: O(n) extra memory
- Stability: Keeps equal elements in order



Comparison with other Algorithms

- Merge Sort: Stable, consistent, scalable
- Bubble/Insertion Sort: $O(n^2)$, not suitable for large data
- Quick Sort: Faster but unstable in worst case



Real-World Applications

- **Big Data: External sorting**
- **Linked Lists: Efficient merge**
- **Parallel Processing: Easy to parallelize**
- **Databases & Scheduling: Stability important**

Conclusion

- Merge Sort is simple yet powerful
- Always runs in O(n log n)
- Stable and scalable
- Uses extra memory but reliable in real-world