	Dr D Y Patil Pratishthan's Dr. D.Y. Patil Institute of Engineering, Management and Research, Akurdi, Pune	
Academic Year: 2025-26	**LP-III Mini-Project Report**	
Term – I	**Department: Computer Engineering**	**Date of Preparation :**

Name of Practical	Lab Practice III
Mini - Project Title	Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case
Batch	B3
Student Name (with roll no)	45 - Om Salunkhe, 46 - Manmath Bhujbal, 53 - Avishkar Chavan, 49 - Ayush Ranjan
Batch Incharge	Mr Jitendra Garud

Introduction

Sorting algorithms play a crucial role in data organization and analysis. Among various sorting algorithms, **Merge Sort** is a classic example of the **Divide and Conquer** strategy, offering stable and predictable performance.

In this mini-project, the **Merge Sort** algorithm is implemented in two versions:

1. **Sequential Merge Sort**, and
2. **Parallel Merge Sort** using Python's multiprocessing module.

The parallel version takes advantage of multiple CPU cores to sort different partitions of data simultaneously, thereby improving performance for large datasets. The project also compares the execution times of both approaches to analyze the effectiveness of parallel computation.

Aim

To implement and evaluate the performance of Sequential and Parallel Merge Sort algorithms using Python's multiprocessing library.

Objective/Motivation

- To understand the working of the Merge Sort algorithm.
- To explore the concept of parallel processing in Python.
- To implement and compare single-threaded and multiprocessing approaches.
- To analyze how parallelization impacts sorting performance for large datasets.

	Dr D Y Patil Pratishthan's Dr. D.Y. Patil Institute of Engineering, Management and Research, Akurdi, Pune	
Academic Year: 2025-26	**LP-III Mini-Project Report**	
Term – I	**Department: Computer Engineering**	**Date of Preparation :**

Implementation and maintenance

1. Sequential Merge Sort
   - The input list is recursively divided into two halves.
   - Each half is sorted independently.
   - The two sorted halves are merged to form the final sorted list.
2. Parallel Merge Sort (Multiprocessing)
   - The dataset is divided into multiple partitions based on the number of CPU cores.
   - Each partition is sorted in a separate process concurrently.
   - Finally, all sorted partitions are merged together iteratively until one sorted list remains.

Output Screenshots

```
PS C:\Users\SANKET\Documents\python> python miniprojectDAA.py 1000000
Time taken for merge_sort: 3.074594497680664
Time taken for merge_sort_mutltithreaded: 2.425234317779541
PS C:\Users\SANKET\Documents\python> python miniprojectDAA.py 100000
Time taken for merge_sort: 0.22265911102294922
Time taken for merge_sort_mutltithreaded: 0.4915142059326172
PS C:\Users\SANKET\Documents\python> python miniprojectDAA.py 10000000
Time taken for merge_sort: 50.31017303466797
Time taken for merge_sort_mutltithreaded: 28.026350259780884
PS C:\Users\SANKET\Documents\python>
```

Conclusion

The mini-project successfully demonstrates the concept of parallel merge sort using Python's multiprocessing module.
Performance comparison reveals that for small datasets, the sequential approach is faster due to multiprocessing overhead.
However, for larger datasets, the parallel version shows noticeable speed improvements, effectively utilizing multiple CPU cores.

Name & sign of student1
Signature of Students

Name & sign of batch incharge
Signature of Batch Incharge