WEEK - 2

Papers:

- OptiFlexSort: A Hybrid Sorting Algorithm for Efficient Large-Scale Data Processing - Abuba, Nelson Seidu, Edward Yellakuor Baagyere, Callistus Ireneous Nakpih, and Japheth Kodua Wiredu
- Comparison of Insertion, Merge, and Hybrid Sorting Algorithms Using C++, by Masuma M. Alqattan, Shahad B. AlQarni, Zainab R. Alramadan, and Razan A. Al Sari
- Development of a Novel Hybrid Sorting Algorithm for Energy Efficiency in Resource-Constrained Devices, by Ayodele, Oluwakemi Sade, Owoeye, Sheidu Audu Yakubu, Oloruntoba Leke Joseph
- On the Worst-Case Complexity of TimSort - Nicolas Auger, Vincent Jugé, Cyril Nicaud, and Carine Pivoteau

Classical Results:

- Hybrid sorts prove to be more efficient than traditional sorting algorithms.
- OptiFlexSort is built on the fundamental features of QuickSort but is enhanced with better pivot selection (median-of-three) and adaptive partitioning—techniques found in other sorting algorithms—and demonstrates a 10-15% improvement in execution time compared to traditional algorithms like Merge Sort and Heap Sort when processing large datasets.
- The research from Alquttan et al. explored a simpler hybrid approach by dividing an array into two parts, using insertion sort on one half and merge sort on the other. By comparing the execution time, the results showed that while hybrid sort outperformed insertion sort in worst-case scenarios, it still didn't exceed the performance of pure merge sort, especially on larger arrays.
- The KSU Hybrid Sort, as presented by Ayodele et al., combines Quick Sort and Insertion Sort with a cut-off size threshold, leveraging Quick Sort's speed for large sub-arrays and Insertion Sort's efficiency for smaller ones, achieving significant energy savings (e.g., 10.5 joules vs. 61.5 joules for Quick Sort at 100,000 elements in Python), particularly on resource-constrained devices.
- Classical hybrid approaches often rely on fixed strategies, such as predetermined split points or cut-offs, which can limit adaptability to varying data sizes and types, as seen in both Alqattan et al.'s fixed split and KSU's static cut-off tuning.
- Timsort is a hybrid sorting algorithm that combines insertion sort and merge sort, widely used in Python and Java. Auger et al., provides the first accurate proof that Timsort operates in $O(n \log n)$ time complexity in the worst case. They also show that Timsort is adaptive to naturally ordered sequences and achieves $O(n \log n + \rho)$ near-linear time, where ρ is number of runs (already sorted subarrays).

What is new about our proposal:

- We are trying to create a new hybrid sort by combining two or three sorts or incorporating features from other sorts into one sort.
- The hybrid method by Alquttan et al. uses a fixed structure: a fixed split point and fixed sorting methods. Our proposed idea is more dynamic. Instead of the hardcoded split, we will test different thresholds and find the optimized value for each array. This allows the hybrid sort to adapt better to different input sizes and data characteristics.
- Unlike the KSU Hybrid Sort, which uses a static cut-off size tuned for energy efficiency on a specific setup, our approach will dynamically adjust the threshold based on real-time analysis of the input data (e.g., size, degree of sortedness), potentially improving both time and energy efficiency across diverse environments.
- We aim to explore integrating adaptive techniques inspired by OptiFlexSort, such as enhanced pivot selection, with our dynamic threshold mechanism, creating a hybrid that not only optimizes for execution time but also remains flexible for large-scale and resource-constrained scenarios.