Week - 4 (Group 2)
Roles and Subtasks

- As discussed in the previous meeting, 2 group members worked on designing the new hybrid sort solution. Our algorithm integrates merge and bubble sort, supplemented by novel optimization techniques. This includes trying and testing incorporating techniques from existing hybrid sort algorithms like Timsort.
    - Shriraj: Designing the pseudocode for the new hybrid sort approach (Static array slicing solution) as well as discussing techniques to optimize the algorithm, and conducting more statistics methods to measure the performance.
    - Yuying: Adding different features and optimizing the algorithm with detect runs and other techniques (reverse, highly disordered, etc), creating code to detect algorithm time baseline.
- The other 2 group members worked on creating the dataset running it with the baseline algorithms and recording the performance.
    - Becky: Organized the existing datasets and generated new datasets based on our needs. Newly generated datasets include: random and half-sorted—new sizes generated for the online dataset: 50M and 100M.
    - Satya: Utilized the neatly generated datasets from above to run for 100 iterations using different sorting algorithms (TimSort, IntroSort, MergeSort, BubbleSort) and generated inferences for the Hybrid algorithm we proposed and the above mentioned conventional sorting algorithms with a wrapper algorithm.

Current Tools and Potential Extensions

- We are using Google Colab for designing and sample testing the new hybrid sort. Github to store the datasets and analysis code. Some data sets are too large for github and those are ignored from github repo.
- Computation resource configuration:
    - Apple Mac M2 Pro
    - 10 core CPU, 16 core GPU, 16 core neural engine
    - 16 GB RAM
- Total Dataset size used for analysis: 4.0 GB

Impact of Different Parameters on Performance

- **Data Distribution:** We have confirmed that half-sorted, random, reverse, and fully sorted data can drastically alter runtimes—especially for insertion-based or bubble-based algorithms.
- **Threshold Sizes:** In our custom hybrid approach, the threshold we use for segmenting the array (e.g., 32 or 64) greatly influences efficiency. For static splitting, the best threshold is 32 for the small arrays (1000 - 100000). However, this value is changeable if we change the input array.
- **Run Detection vs. Fixed Segmentation:** A run is a naturally occurring contiguous segment of the array already in sorted order. In our experiments, we dynamically detect runs and use them for segmentation rather than relying on a fixed static segmentation. This has improved the overall sorting performance. When an array is partially sorted, the runs tend to be longer than a fixed-size block, which means fewer merge passes, reducing time overhead.
- **Different parameters that affect performance:** Referencing the standard implementation of timsort, we tested several array-based condition checks (e.g., reversed, almost sorted, highly disordered) before sorting, aiming to dynamically choose suitable algorithms. However, these checks also introduced overhead and slowed down overall performance instead of improving it.

Comparison with Existing Solutions

- **Baseline:** For a baseline comparison, we implemented and benchmarked common sorting algorithms, including MergeSort, and BubbleSort, along with well-known hybrid sorts like Timsort, and Introsort.
- **Inference**: We conduct multiple inference runs across all the benchmarks with each algorithm and calculate mean execution time, ensuring statistically robust performance evaluation.

Preliminary Results and Conclusion Planning

- **Data Scale Matters**: At 100,000 elements, some algorithms (like Bubble) already display large runtimes, yet we still see minimal separation among advanced methods. We plan to scale up to millions of elements to fully expose performance differences.
- **Array Characteristic Detection:** Although techniques like run detection reduce the time overhead for some cases, they may also add cost. Striking a balance between detection and performance is still a problem. We plan to explore this tradeoff more carefully and at least find out the best detection methods for different types of arrays.