SCSA SC2001 Lab Example Class Project 1 Team 7

PU FANYI (U2220175K)

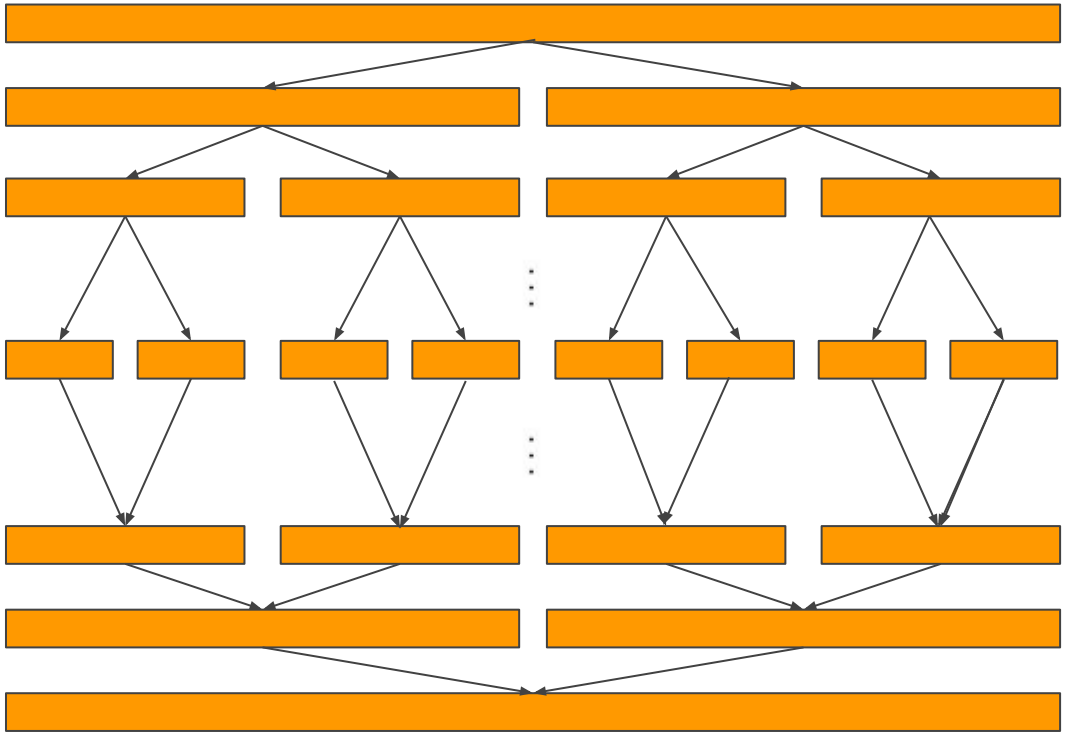
PUSHPARAJAN ROSHINI (U2222546A)

QIAN JIANHENG OSCAR (U2220109K)

RHEA SUSAN GEORGE (U2220116B)

Implementation of Hybrid Sort

Threshold S has been hit



Split

Insertion sort

Merge

Implementation

```
compare_count_t mergeSortWithInsertionSort(int *begin, const int *end, const int threshold) {
    compare_count_t compareCount = 0;
    const size_t size = end - begin;
    if (size <= threshold) {
        compareCount += insertionSort(begin, end);
    } else {
        int *mid = begin + size / 2;
        compareCount += mergeSortWithInsertionSort(begin, mid, threshold);
        compareCount += mergeSortWithInsertionSort(mid, end, threshold);
        compareCount += merge(begin, mid, end);
    }
    return compareCount;
}
```

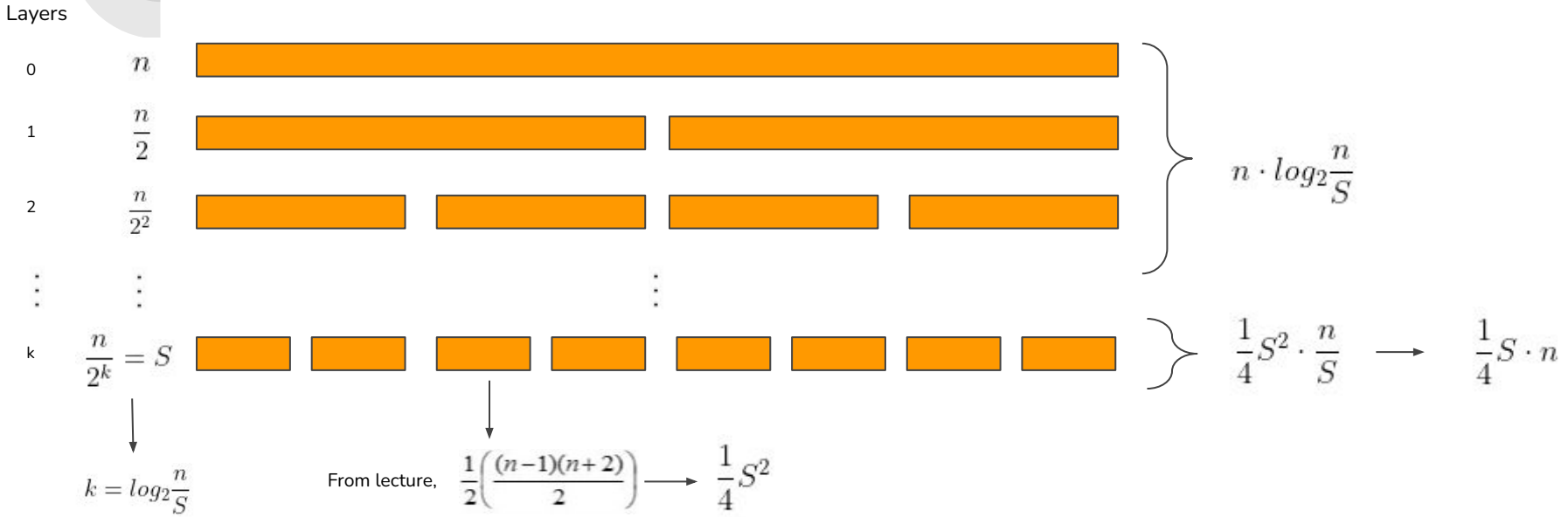
Random data generation

- Multiple test and get the average.
- Mersenne Twister

```
typedef compare_count_t (*SortFunction)(int *begin, const int *end, ...);

EvaluationResult evaluate(SortFunction sortFunction,
                          const int *array_begin, const int *array_end, ...) {
    size_t array_size = array_end - array_begin;
    int *array_copy = (int *) calloc(array_size, sizeof(int));
    memcpy(array_copy, array_begin, sizeof(int) * array_size);
    va_list args;
    va_start(args, array_end);
    clock_t begin = clock();
    compare_count_t compareCount = sortFunction(array_copy, array_copy + array_size, *args);
    clock_t end = clock();
    EvaluationResult result;
    result.time = end - begin;
    result.compareCount = compareCount;
    result.correctness = isSorted(array_copy, array_copy + array_size);
    free(array_copy);
    return result;
}
```

Theoretical Analysis (Hybrid Sort)



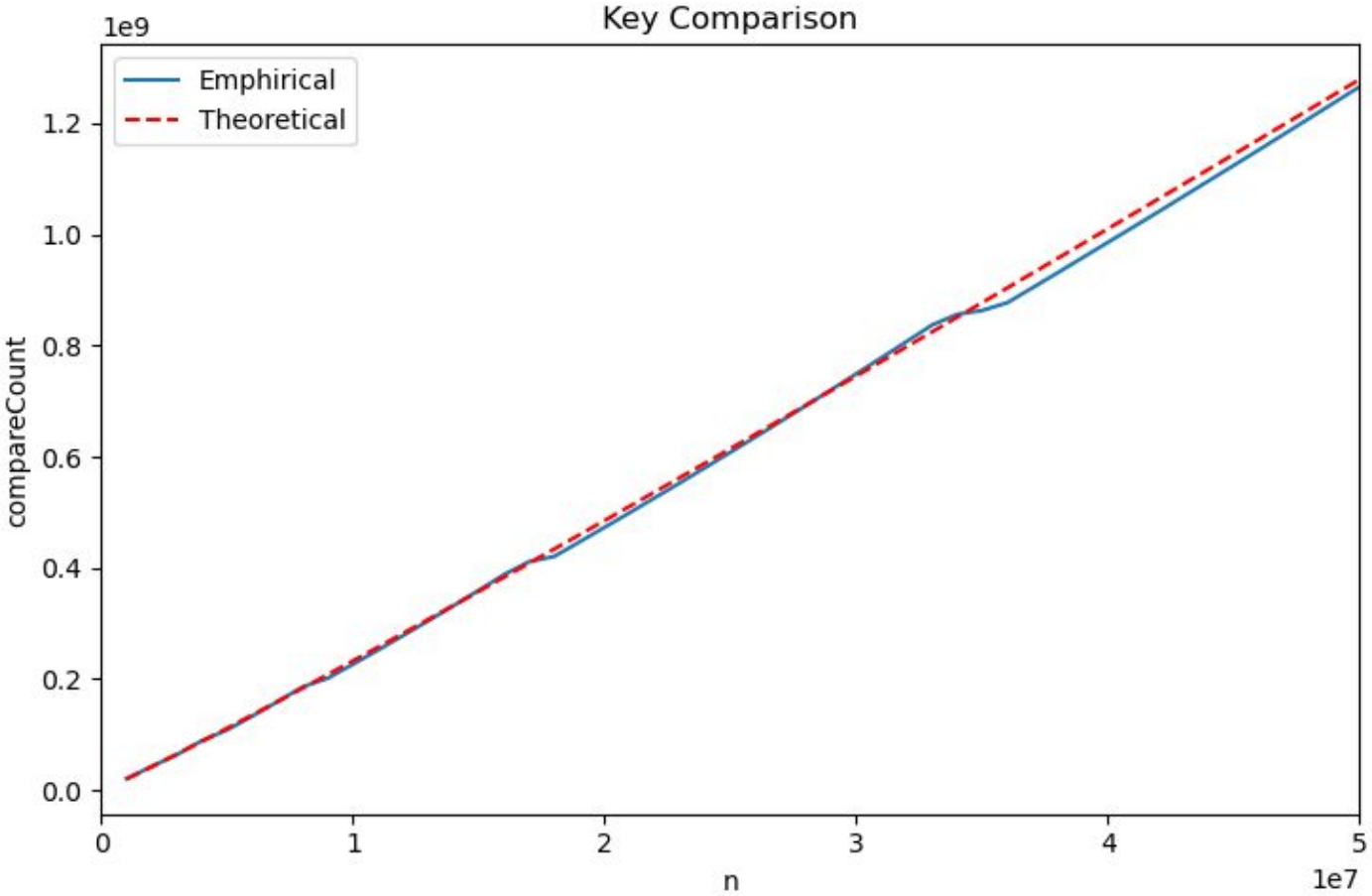
$$Eqn = n \cdot log_2 \frac{n}{S} + \frac{1}{4}S \cdot n \quad = \quad O(n \cdot log_2 \frac{n}{S} + S \cdot n)$$

Part i : Fixed S, Vary n

How we approached :

1. With fixed S threshold of 16, and input size ranging from 1,000,000 to 50,000,000
    a. S = 16 is randomly generated
2. For every size n input, each data point is curated by **taking average of 5 random runs**
3. 50 n,CompareCount and time data results generated
4. Key Comparisons vs Input n sizes & Input n sizes vs Time plotted

Key comparison vs Different Input n

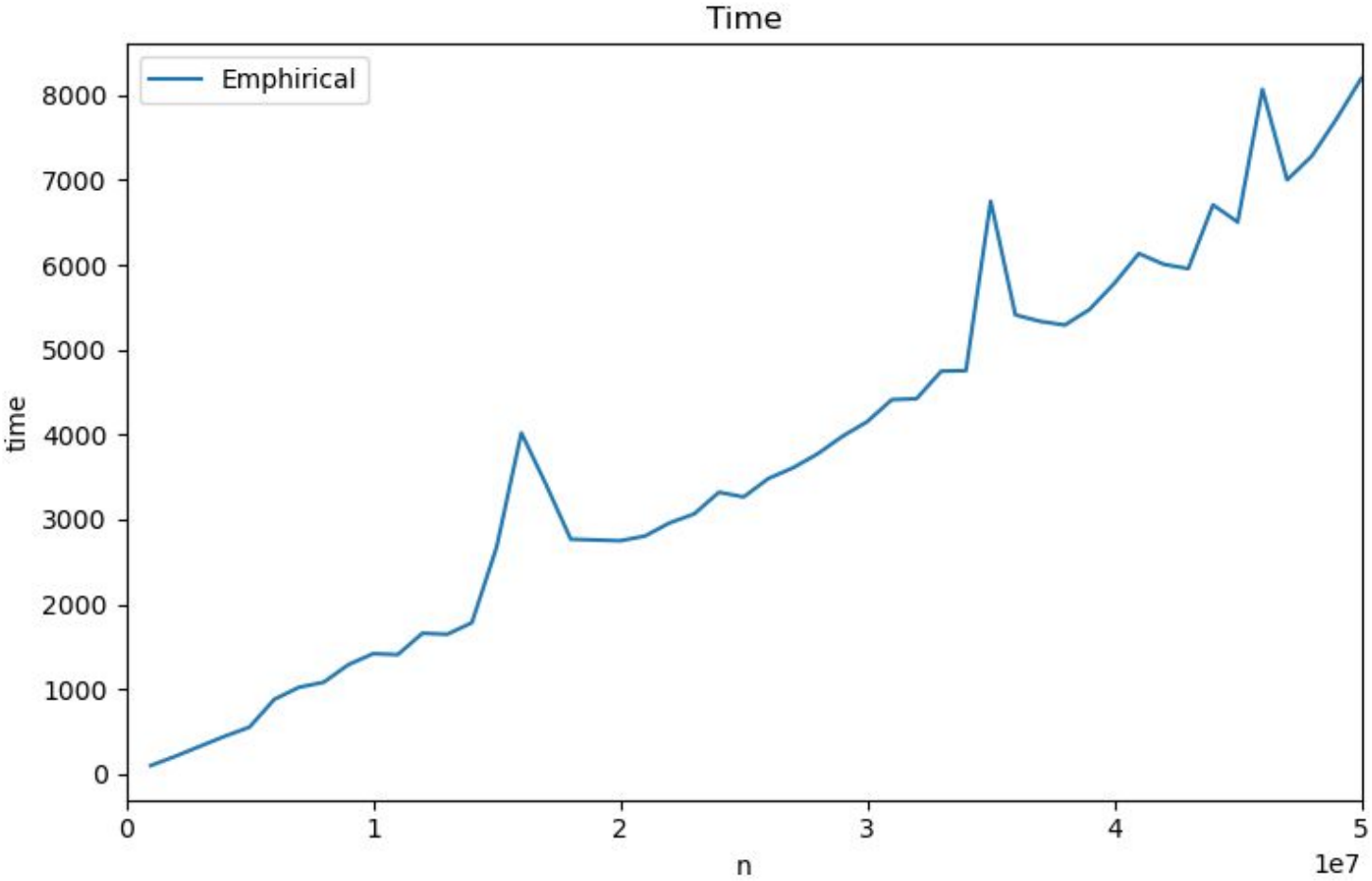


Shape of theoretical analysis largely matches the emphirical results

Theoretical analysis is derived from :

```
theoretical_values =
n_values*np.log2(n_values/16)
+ 1/4 *16*n_values
```

$$y = n \cdot \log_2\left(\frac{n}{16}\right) + \frac{16}{4} \cdot n$$



Possible deviations in the time complexity :

- **Variations in Input Size**
  - Array size is changing whenever new input n size is generated → unstable
- **CPU Distractions**

Part ii: Fixed n, Vary S

Our approach:

1. Generate 5 array with random values of same size, n
2. Run the hybrid sort with S value from 1 to 126 on the 5 array
3. Take the average key comparison and time taken

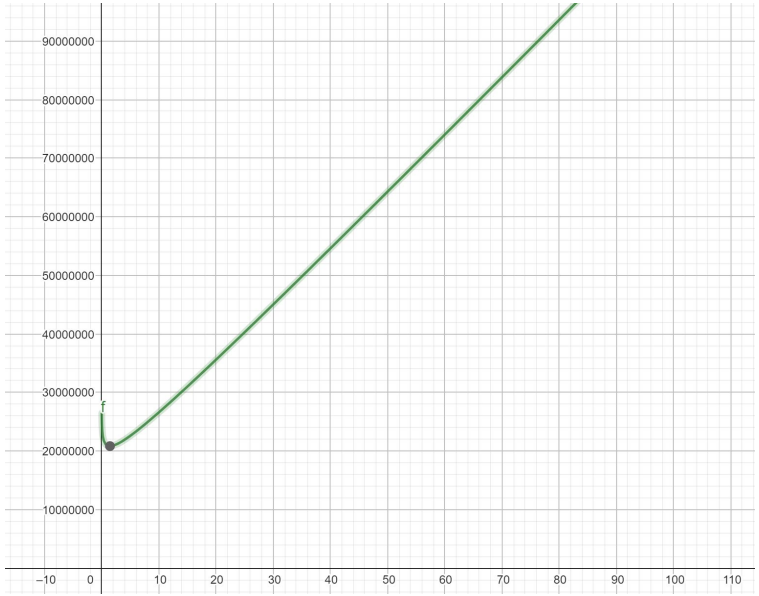
Note*: To simulate the Average case, ideally we should run the Hybrid sort with each S on many arrays.

Theoretical analysis (Key comparison)

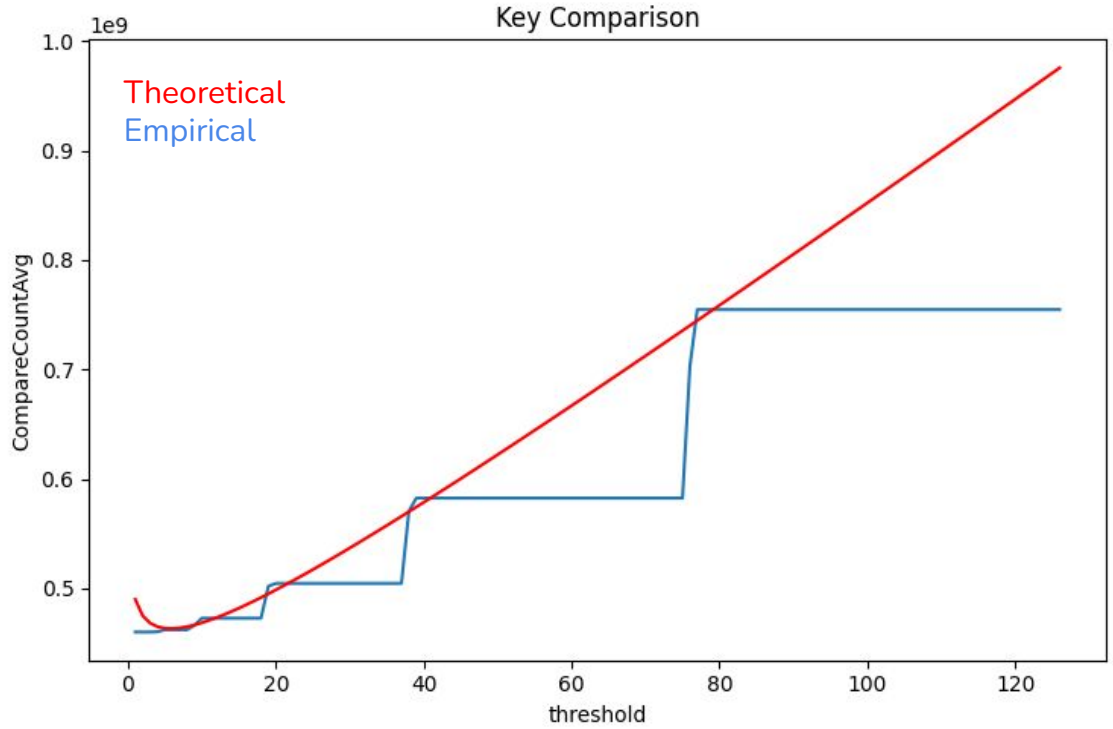
Average case: **Insertion Sort => O(n²)** **Merge Sort => O(nlogn)**

Hypothesis: Key comparison should increase as S increases

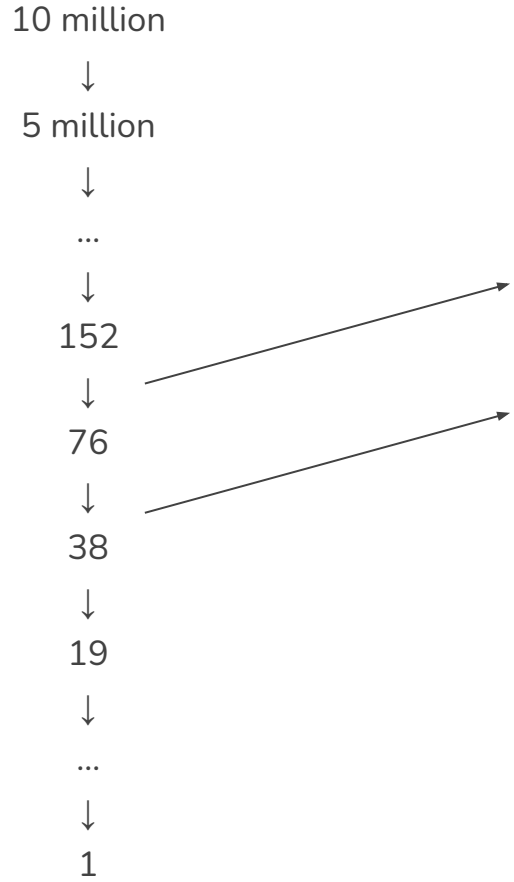
Expected Graph:



Actual result



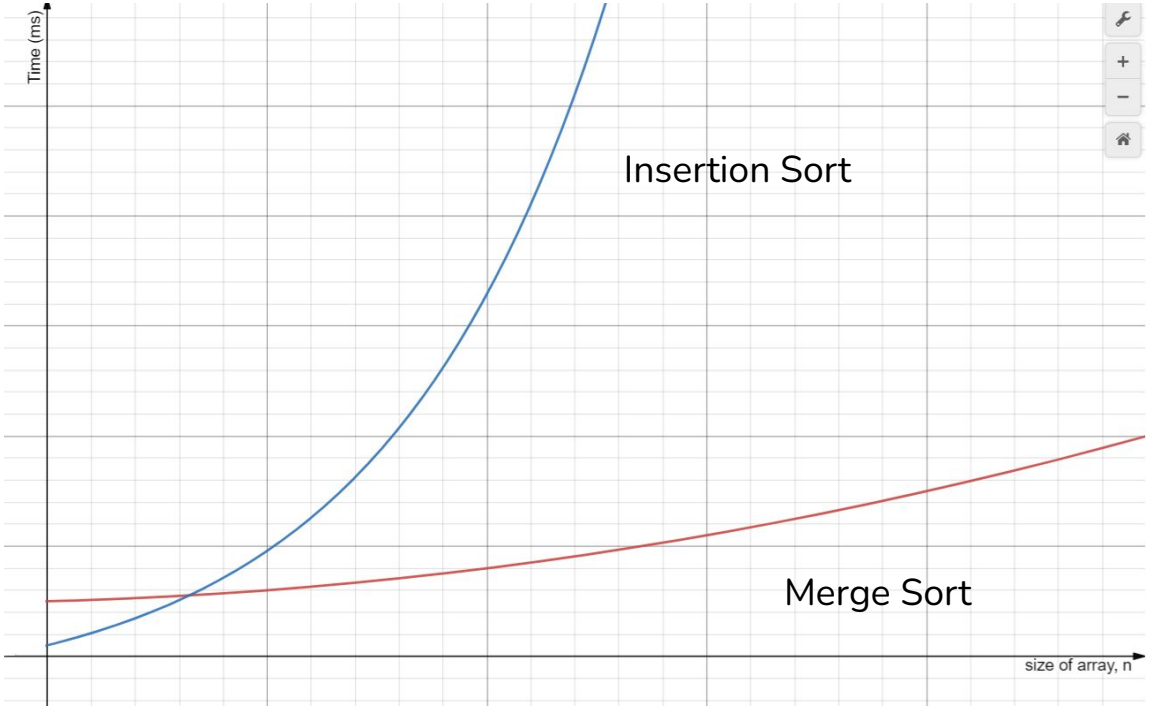
Actual Result



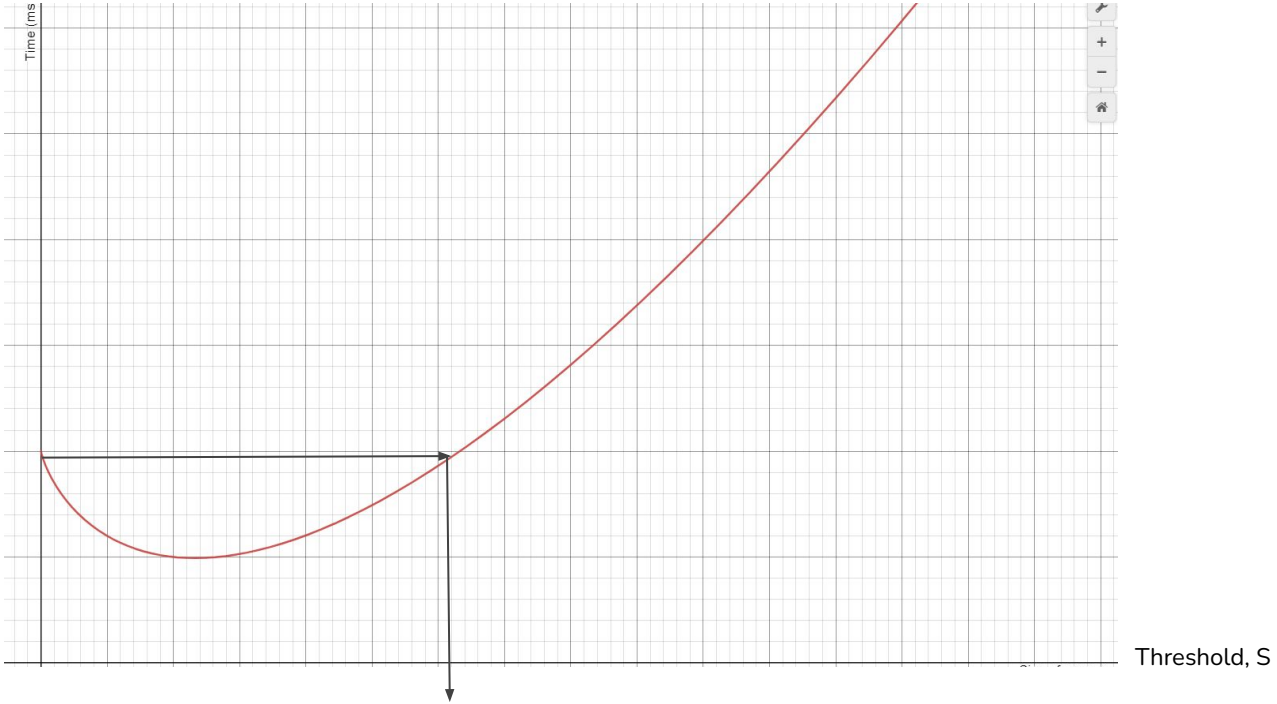
Theoretical Analysis (Time)

Insertion Sort: $C_1 \cdot n^2$
Merge Sort: $C_2 \cdot n \log_2 n$

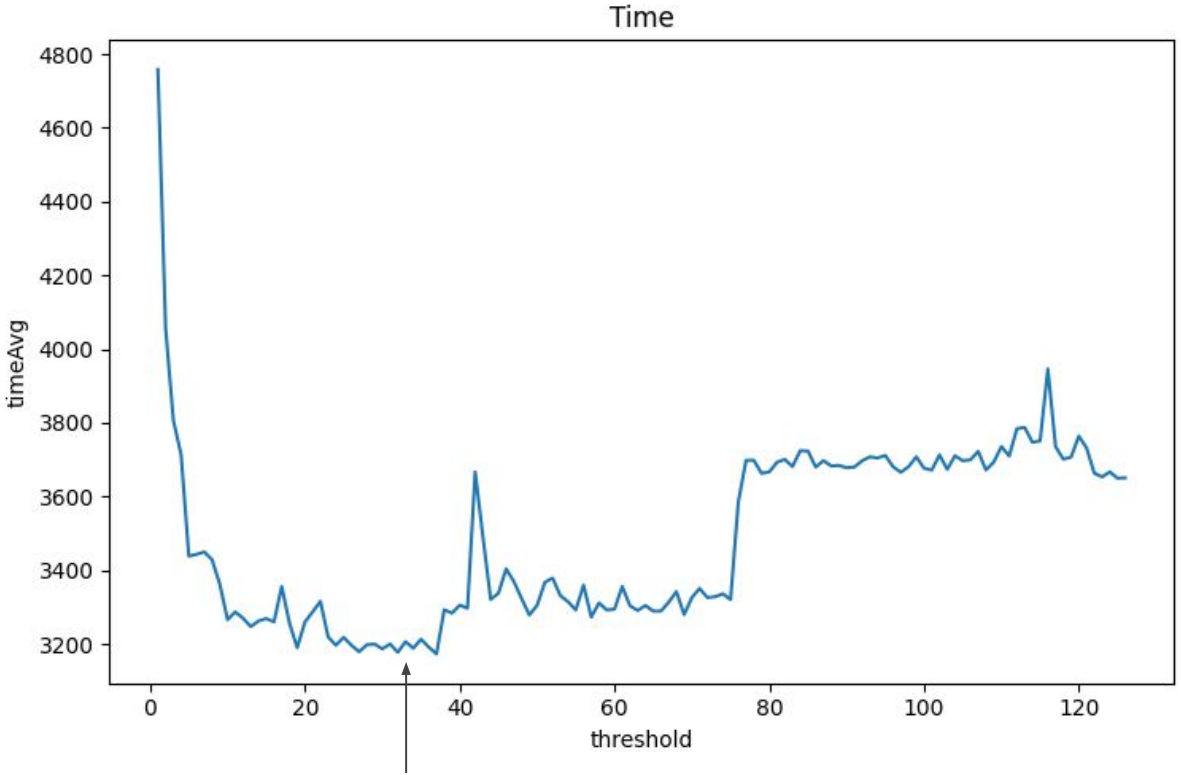
Since $C_2 > C_1$,
When n is **small**, performance of
Merge sort is **worse** than Insertion sort



Hypothesised Graph for Hybrid Sort



Actual Result



Optimal S for this n

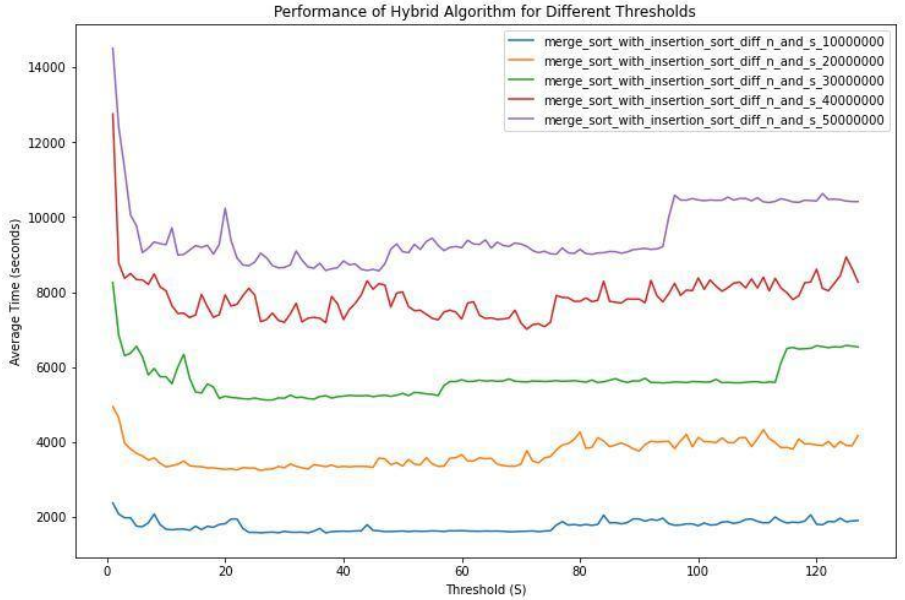
Part iii: Vary n and S

Our approach:

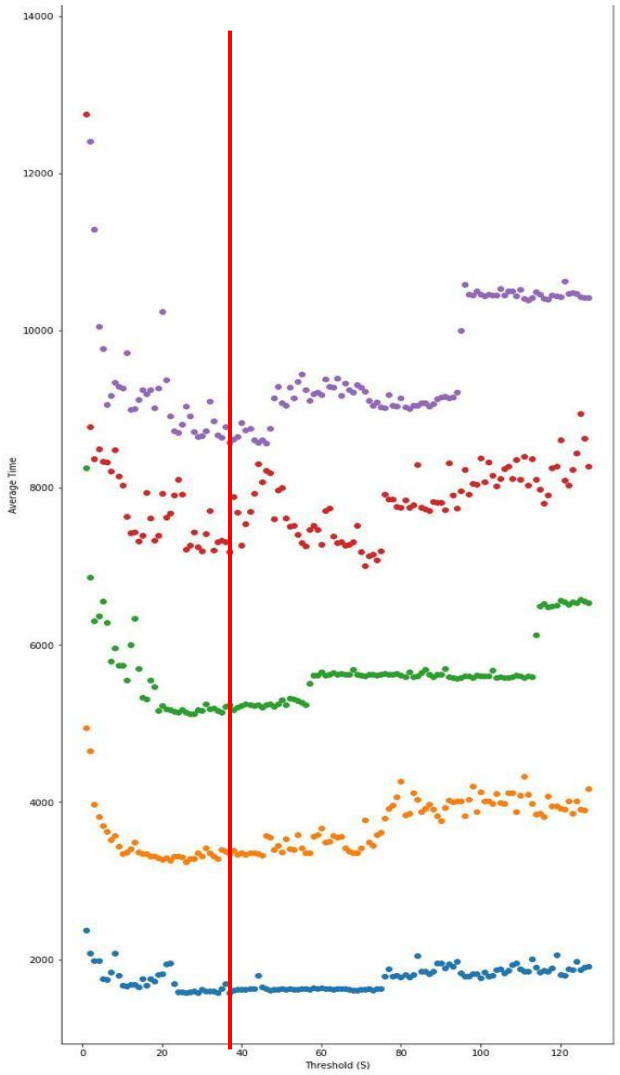
1. part ii, run against 5 different n values

2. The 5 different data sets is plotted into one graph

3. Optimal S derived from the graph

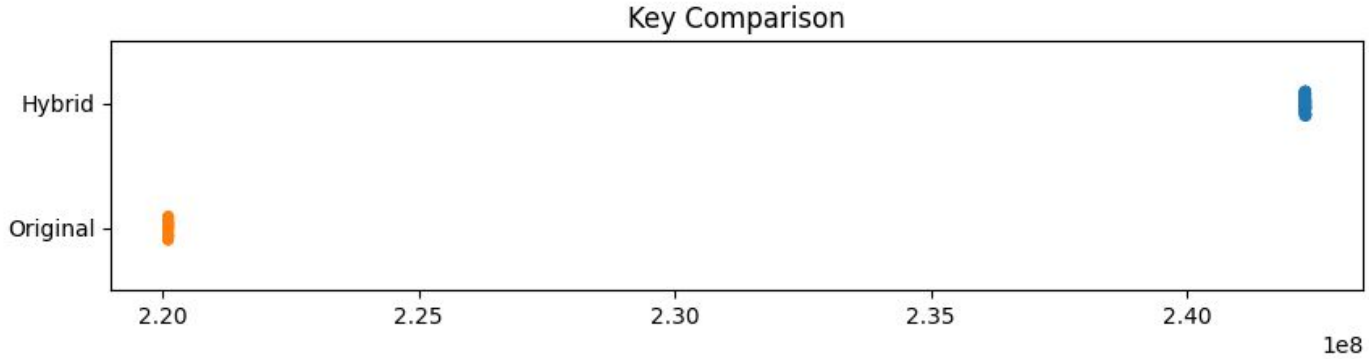


Results

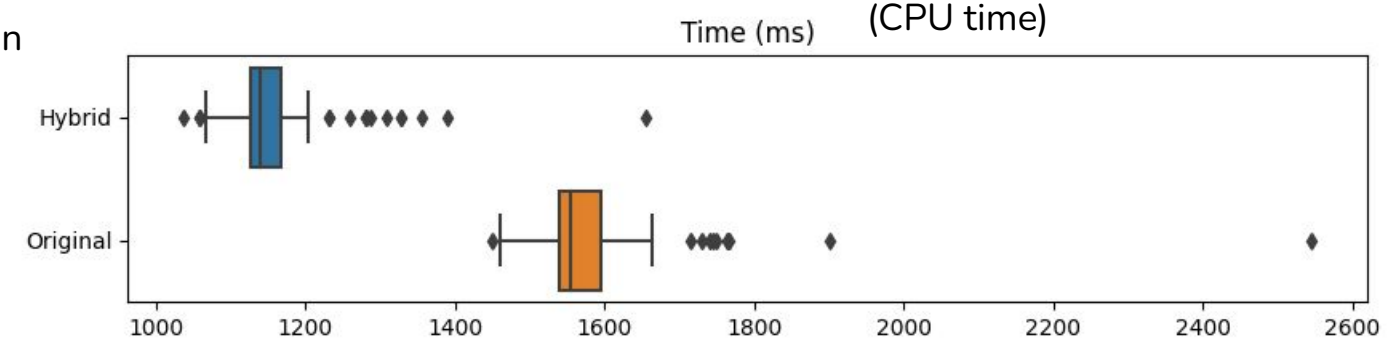


- Data from merge_sort_with_insertion_sort_diff_n_and_s_10000000.csv
- Data from merge_sort_with_insertion_sort_diff_n_and_s_20000000.csv
- Data from merge_sort_with_insertion_sort_diff_n_and_s_30000000.csv
- Data from merge_sort_with_insertion_sort_diff_n_and_s_40000000.csv
- Data from merge_sort_with_insertion_sort_diff_n_and_s_50000000.csv

Evaluation Hybrid vs Original with S = 37



Approximate Percentage gain = 26.72%



(CPU time)