SC2001 LAB 1

INTEGRATION OF MERGE AND INSERTION SORT

Jesmond, Harish, Kyla

ALGORITHM DESIGN

EG: LIST LENGTH = 7, S =3

[38, 27, 43, 3, 9, 82, 10]

[38, 27, 43, 3]   [9, 82, 10]

[38, 27]   [43, 3]

-DIVIDE UNTIL REACH THRESHOLD, S

-INSERTION SORT

-MERGE SORT

ALGORITHM DESIGN

EG: LIST LENGTH = 7, S =3

[3, 9, 10, 27, 38, 43, 82]

↑ ↑

[3, 27, 38, 43] [9, 10, 82]

↑ ↑

[27, 38] [3, 43]

-DIVIDE UNTIL REACH THRESHOLD, S

-INSERTION SORT

-MERGE SORT

ALGORITHM IMPLEMENTATION

HYBRID SORT

```python
def _hybrid_mergesort_helper(arr, left, right, threshold):
    """
    Recursive helper function for hybrid mergesort

    Args:
        arr: Array to sort (modified in-place)
        left: Starting index of subarray
        right: Ending index of subarray
        threshold: Size threshold for switching to insertion sort
    """
    if left < right:
        size = right - left + 1

        # If subarray size is small, use insertion sort
        if size <= threshold:
            _insertion_sort(arr, left, right)
        else:
            # Otherwise, use mergesort
            mid = left + (right - left) // 2
            _hybrid_mergesort_helper(arr, left, mid, threshold)
            _hybrid_mergesort_helper(arr, mid + 1, right, threshold)
            _merge_sort(arr, left, mid, right)
```

CODE SNIPPETS

```python
def hybrid_mergesort(arr, threshold=10):
    """
    Hybrid sorting algorithm combining Merge Sort and Insertion Sort

    Args:
        arr: List of comparable elements to sort
        threshold: Size threshold below which to use insertion sort (default: 10)

    Returns:
        Sorted array
    """
    if not arr:
        return arr

    # Create a copy to avoid modifying the original array
    arr_copy = arr.copy()
    _hybrid_mergesort_helper(arr_copy, 0, len(arr_copy) - 1, threshold)
    return arr_copy
```

ALGORITHM IMPLEMENTATION

GENERATING INPUT DATA

```
import random
import time
import algorithm
def run(x, n, S, mode="hybrid"):
    """
    Function to run a single experiment

    Arg:
    x: the maximum data in dataset
    n: the number of data in datasets
    S: the threshold value

    Return:
    map of values
    """
    a = [random.randint(1, x) for _ in range(n)]
    counter = [0]
    t0 = time.perf_counter()
    if mode=="hybrid":
        algorithm.hybrid_mergesort(a, counter, S)
    else:
        merge_sort(a, 0, len(a)-1, counter)
    t1 = time.perf_counter()
    return {"comp": counter[0], "time": t1 - t0}

#Test
print(run(10, 1000, 10))
```

CREATES A LIST OF N RANDOM INTEGERS BETWEEN 1 AND X.

ALGORITHM ANALYSIS

TIME COMPLEXITY : THEORETICAL ANALYSIS

- **Best Case**: $O(\frac{n}{s} \cdot s + n\log(\frac{n}{s})) \approx O(n + n\log(\frac{n}{s}))$
- **Average Case**: $O(\frac{n}{s} \cdot s^2 + n\log(\frac{n}{s})) \approx O(n \cdot s + n\log(\frac{n}{s}))$
- **Worst Case**: $O(\frac{n}{s} \cdot s^2 + n\log(\frac{n}{s})) \approx O(n \cdot s + n\log(\frac{n}{s}))$

Where:

- n is the input size
- s is the threshold value for switching between merge sort and insertion sort

ALGORITHM ANALYSIS

TIME COMPLEXITY : THEORETICAL ANALYSIS

BEST CASE

- Insertion sort: $\frac{n}{s} \cdot O(s) = O(n)$
- Merge sort: $O(n \log(n/s))$
- **Total:**

$$O(n) + O(n \log(n/s)) = O(n + n \log(n/s))$$

AVERAGE CASE

- Insertion sort: $\frac{n}{s} \cdot O(s^2) = O(ns)$
- Merge sort: $O(n \log(n/s))$
- **Total:**

$$O(ns) + O(n \log(n/s)) = O(ns + n \log(n/s))$$

WORST CASE

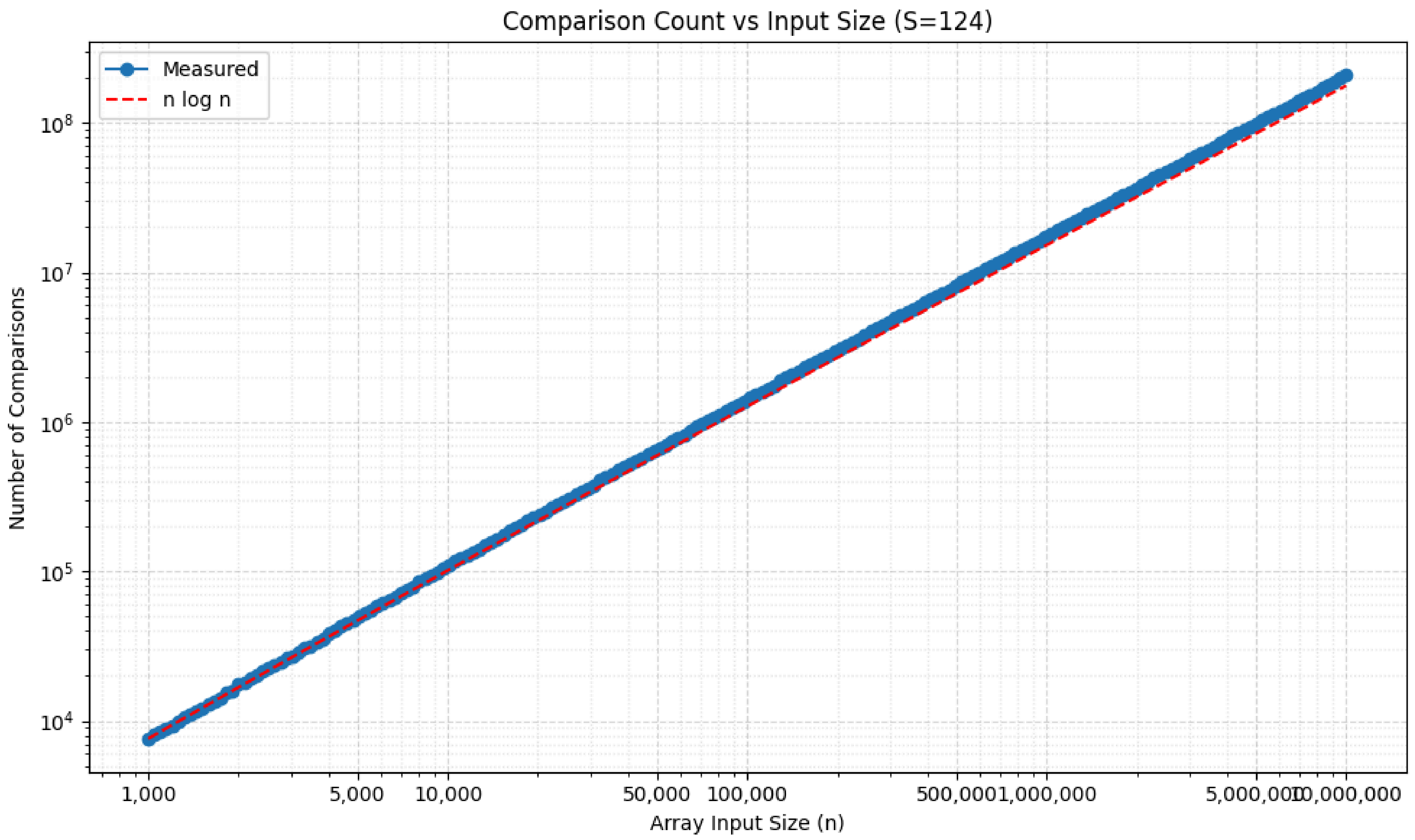
- Insertion sort: $\frac{n}{s} \cdot O(s^2) = O(ns)$
- Merge sort: $O(n \log(n/s))$
- **Total:**

$$O(ns) + O(n \log(n/s)) = O(ns + n \log(n/s))$$

- n is the input size
- s is the threshold value for switching between merge sort and insertion sort

ALGORITHM ANALYSIS

TIME COMPLEXITY : EMPIRICAL ANALYSIS I



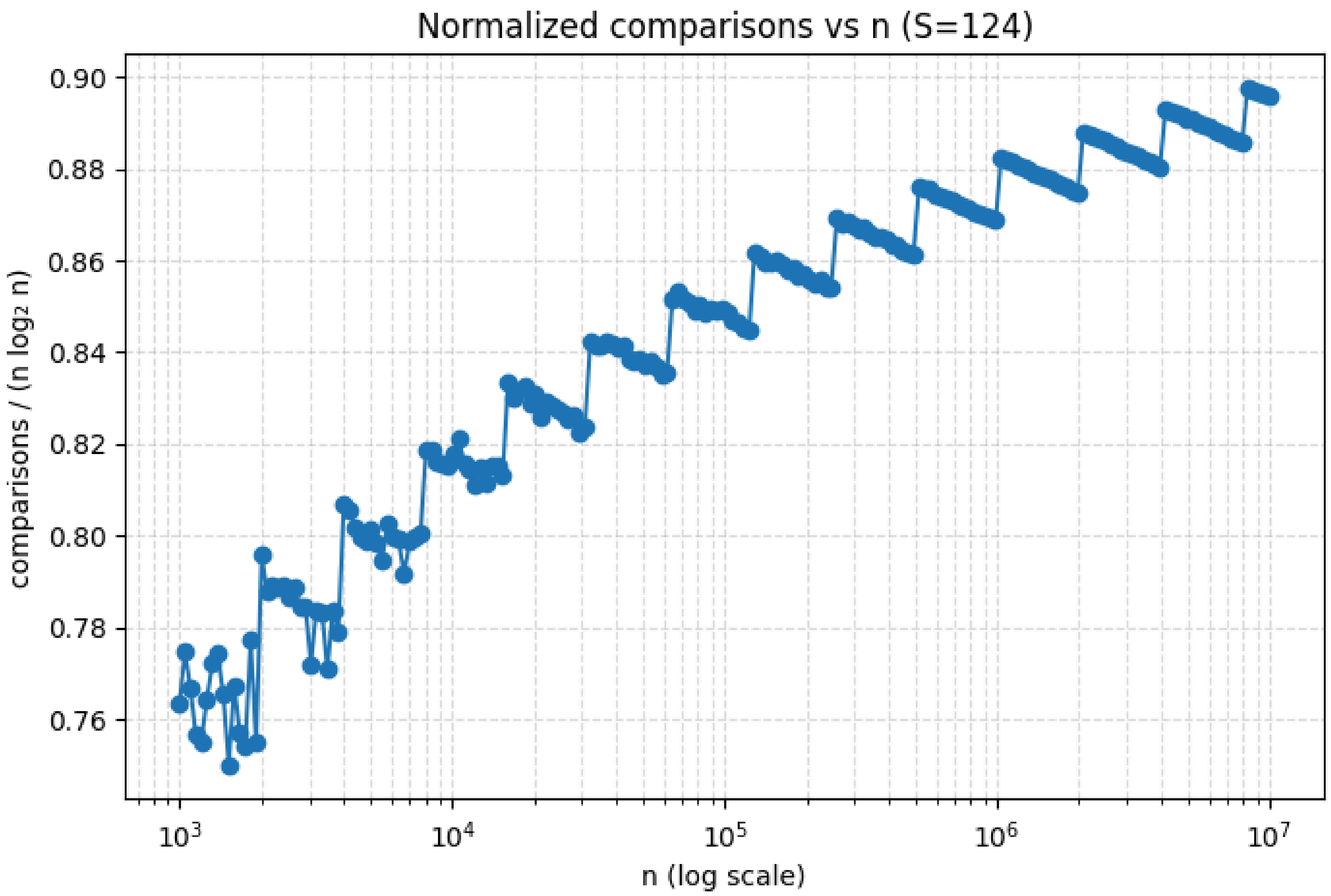
-S FIXED AT 124.

-EACH DATA POINT IS THE MEDIAN OF 3 TRIALS OF THE SAME SET OF DATA.

-STRAIGHT LINE GRAPH(LOG-LOG SCALE) CORRESPONDING TO O(NLOGN)

ALGORITHM ANALYSIS

TIME COMPLEXITY : EMPIRICAL ANALYSIS I



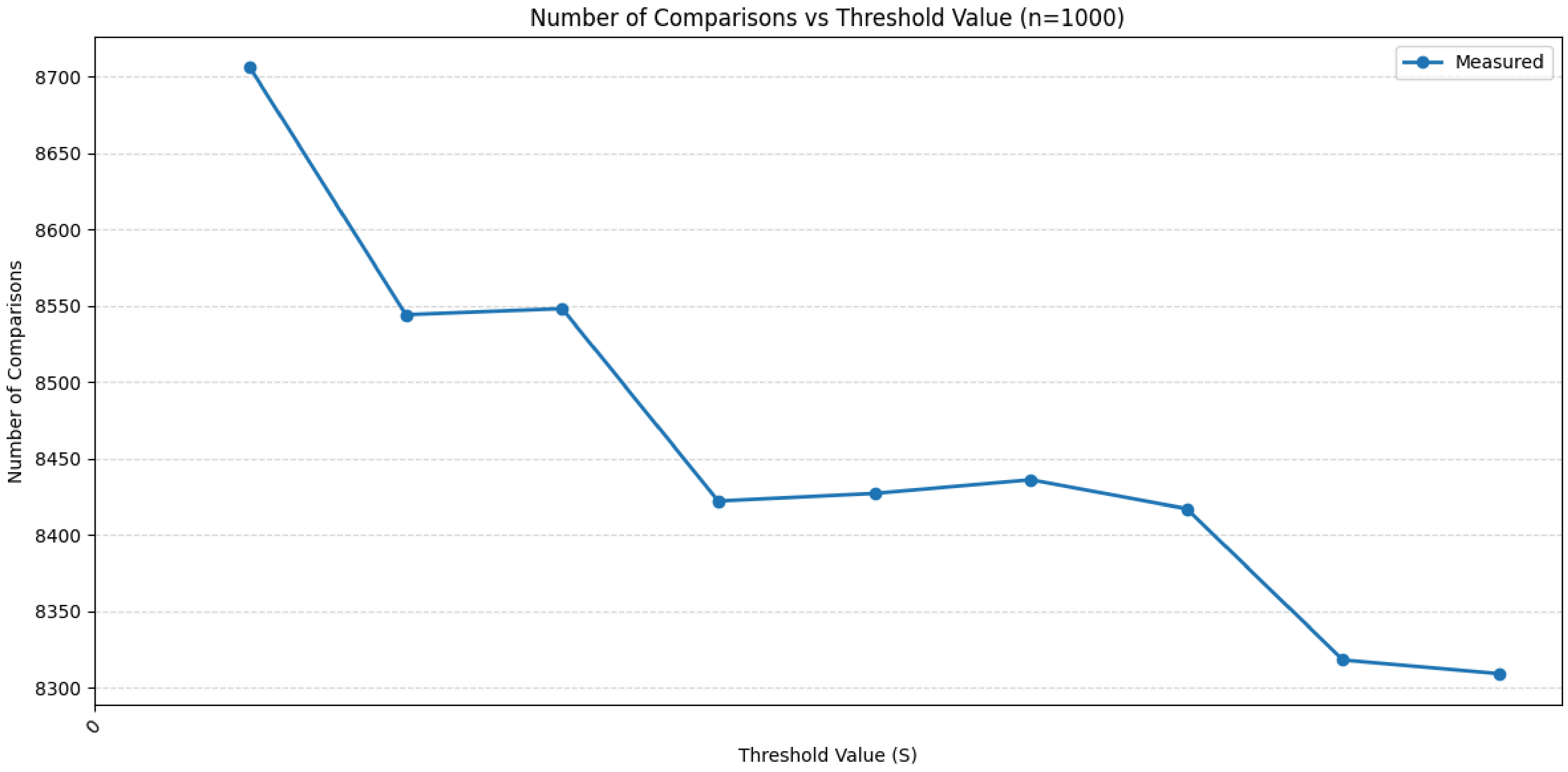
-RATIO IS APPROACHING FLAT LINE

-SEESAW PATTERN APPEARS -> MERGE SORT SPLITS ARRAYS EVENLY INTO HALVES

VERDICT: CONTRIBUTION FROM INSERTION SORT IS NEGLIGIBLE FOR SMALL N.

ALGORITHM ANALYSIS

TIME COMPLEXITY: EMPIRICAL ANALYSIS II



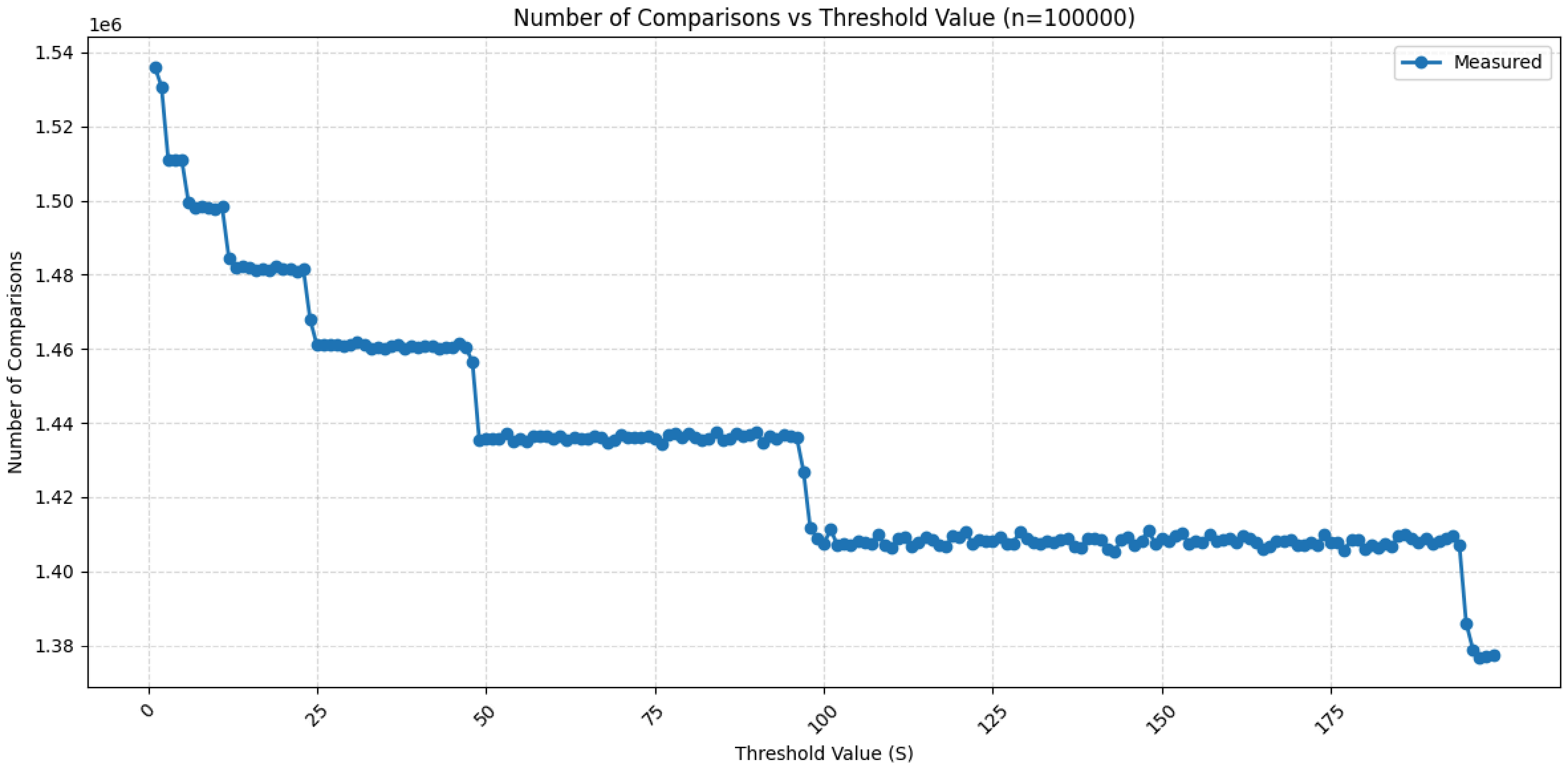
-GRAPH FOR SMALL N AND SMALL S.

-EACH POINT IS A MEDIAN OF THREE TRIALS

-WE TAKE THE MEDIAN OF 3 TRIALS TO ENSURE THAT THE GRAPH DOES NOT SKEW DUE TO ANOMALIES FROM SINGLE RUN TRIALS.

ALGORITHM ANALYSIS

TIME COMPLEXITY: EMPIRICAL ANALYSIS II



-GRAPH FOR LARGE N AND LARGE S.

-EACH POINT IS THE MEDIAN OF THREE DIFFERENT TRIALS

ALGORITHM ANALYSIS

TIME COMPLEXITY: EMPIRICAL ANALYSIS II

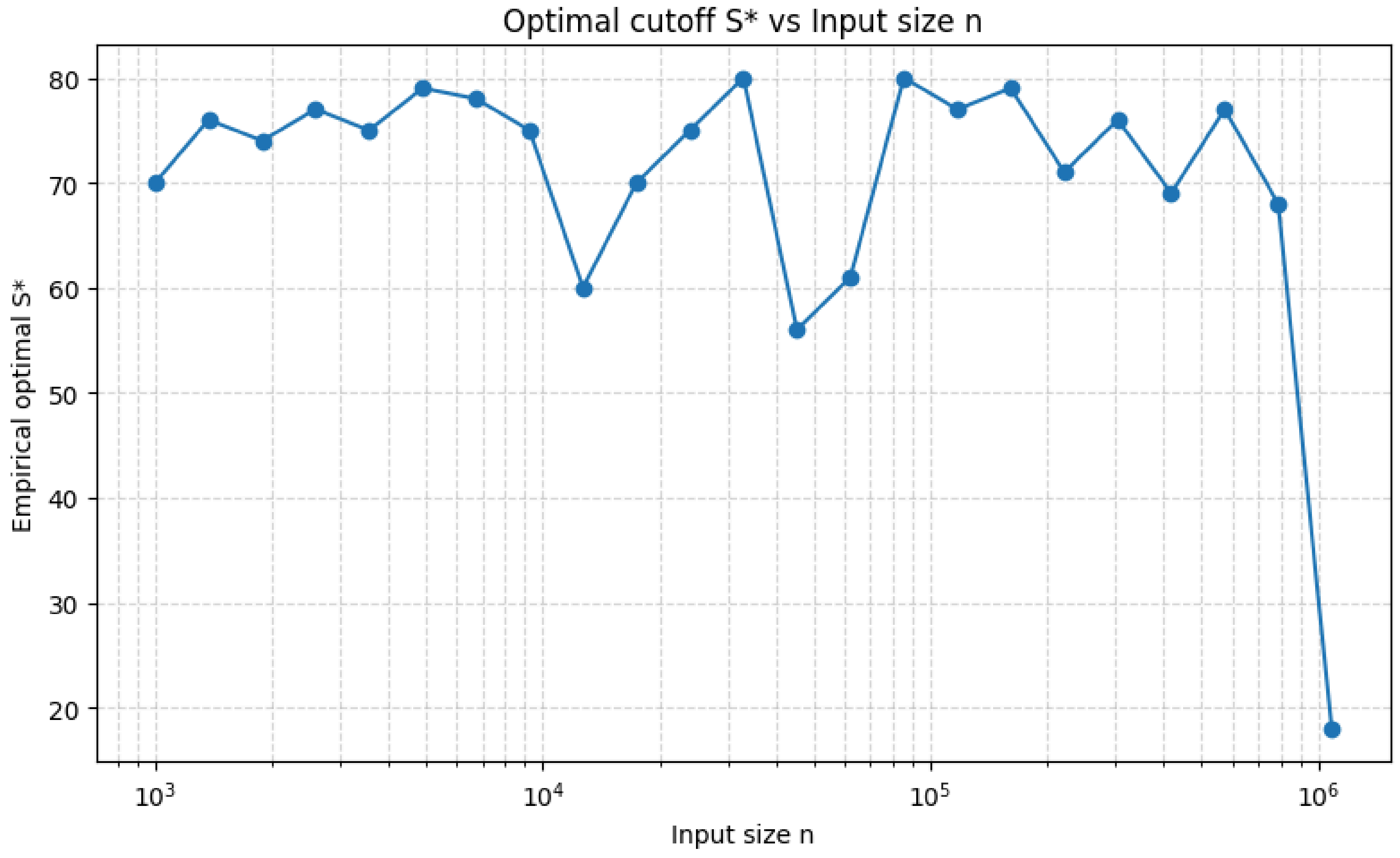
–AS S INCREASES FOR A FIXED VALUE OF N, THE TOTAL NUMBER OF COMPARISONS DECREASES.

–COST OF MERGE SORT DECREASES FOR LARGER S.

–LARGE N DISPLAYS A STAIRCASE PATTERN – WHEN S EXCEEDS THE SIZE OF SUBARRAYS AT A GIVEN RECURSION LEVEL, ONE WHOLE LEVEL OF MERGING IS REMOVED.

ALGORITHM ANALYSIS

DETERMINING OPTIMAL S

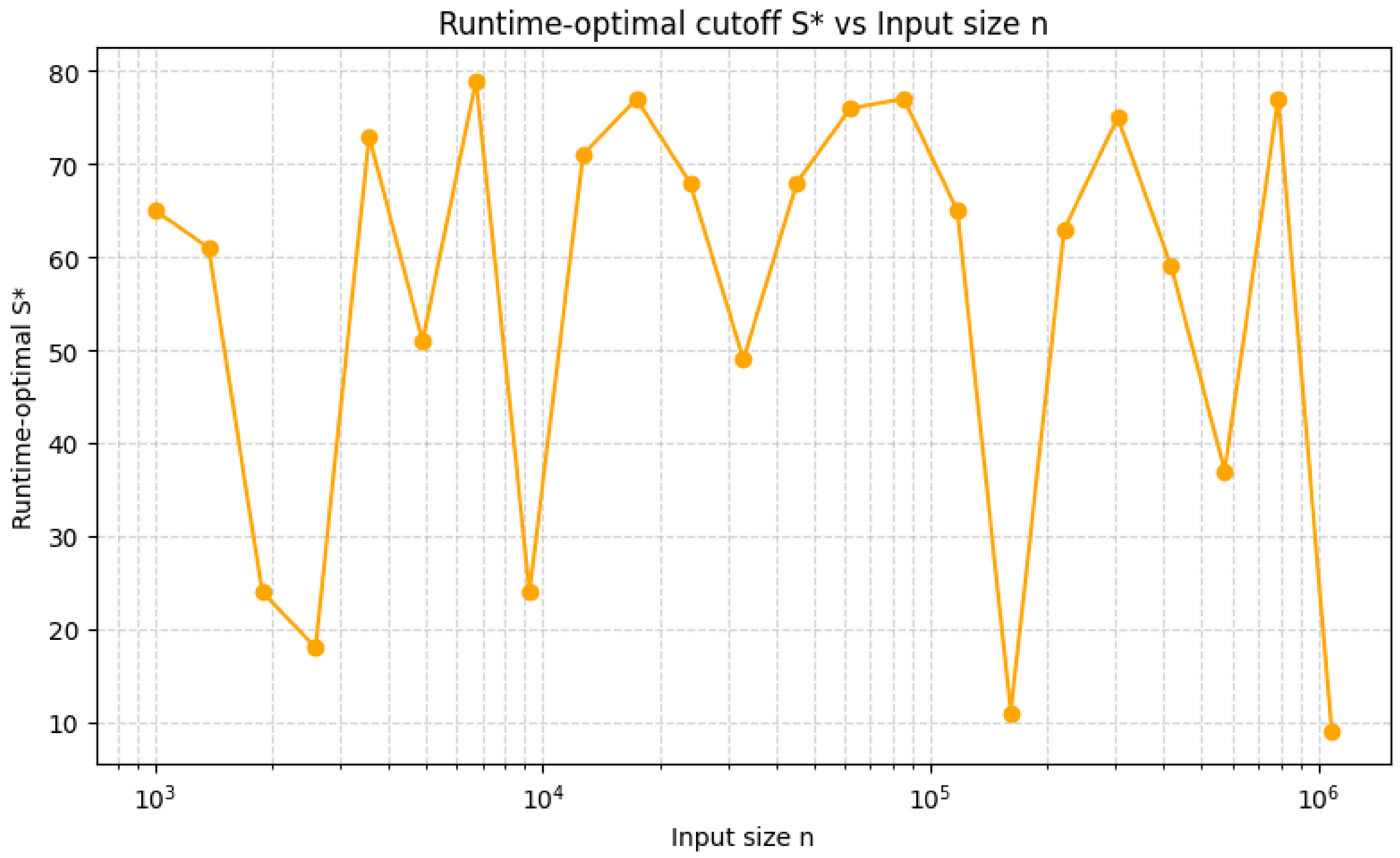


-FOR VERY LARGE DATA SETS, 15-20 IS OPTIMAL.

-OTHERWISE, OPTIMAL RANGE IS BETWEEN 60-80.

ALGORITHM ANALYSIS

DETERMINING OPTIMAL S



-THE OPTIMAL VALUE OF S FLUCTUATES.

-EXTERNAL FACTORS AFFECT OPTIMAL VALUE DURING RUNTIME. E.G. CACHE AND RUNTIME ENVIRONMENT.

-OPTIMAL S VALUE RANGE GENERALLY STILL LIES BETWEEN 60-80 AND THE SUDDEN DECREASE WHEN N APPROACHES 10MIL STILL STANDS. (WITH THE EXCEPTION OF A FEW DIPS).

ALGORITHM ANALYSIS

DETERMINING OPTIMAL S

-FOR SMALLER DATA SETS, AN OPTIMAL VALUE OF 70 FOR S WOULD BE PREFERRED.

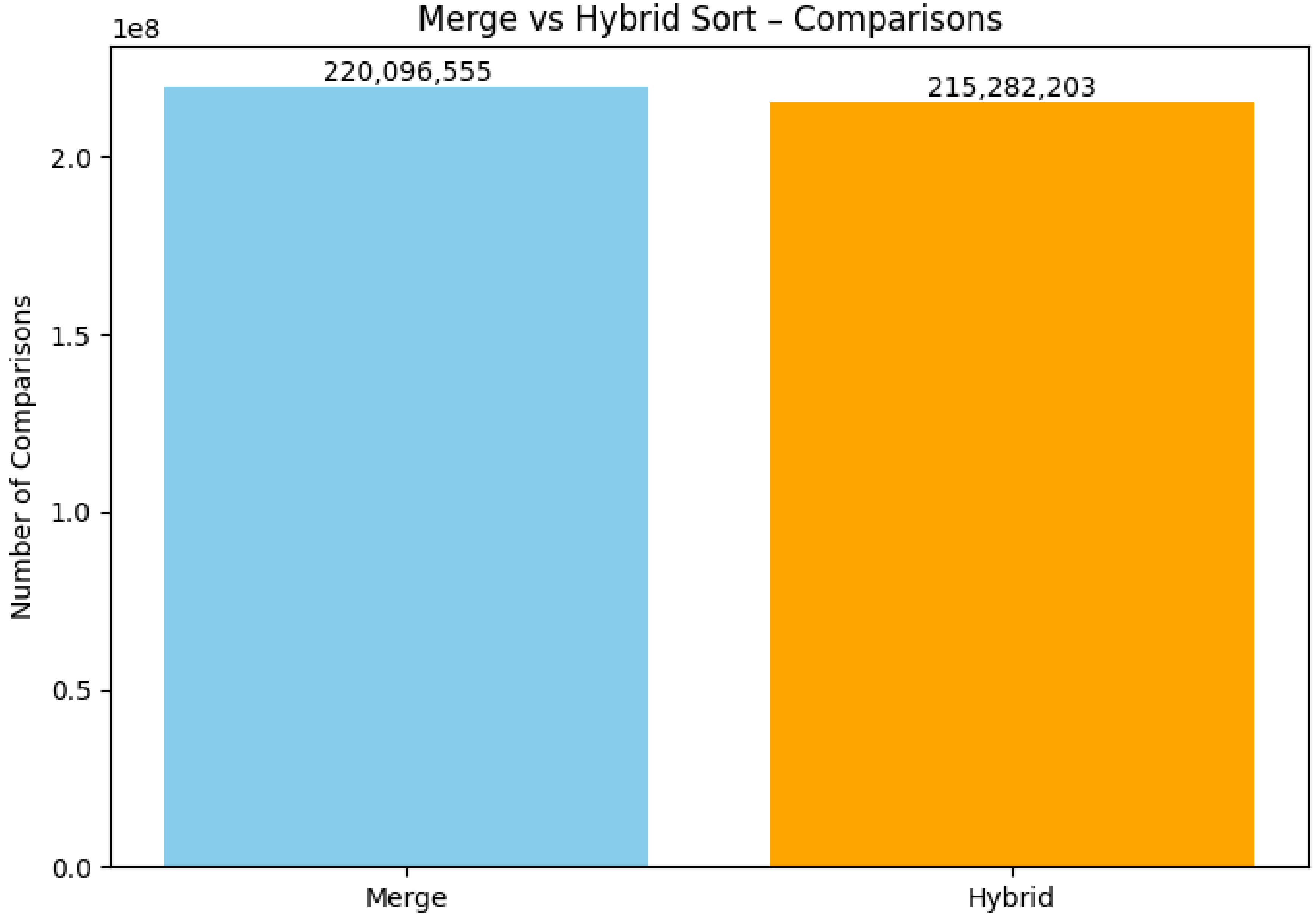
-FOR LARGER DATA SETS, AN OPTIMAL VALUE OF 16 WOULD BE PREFERRED.

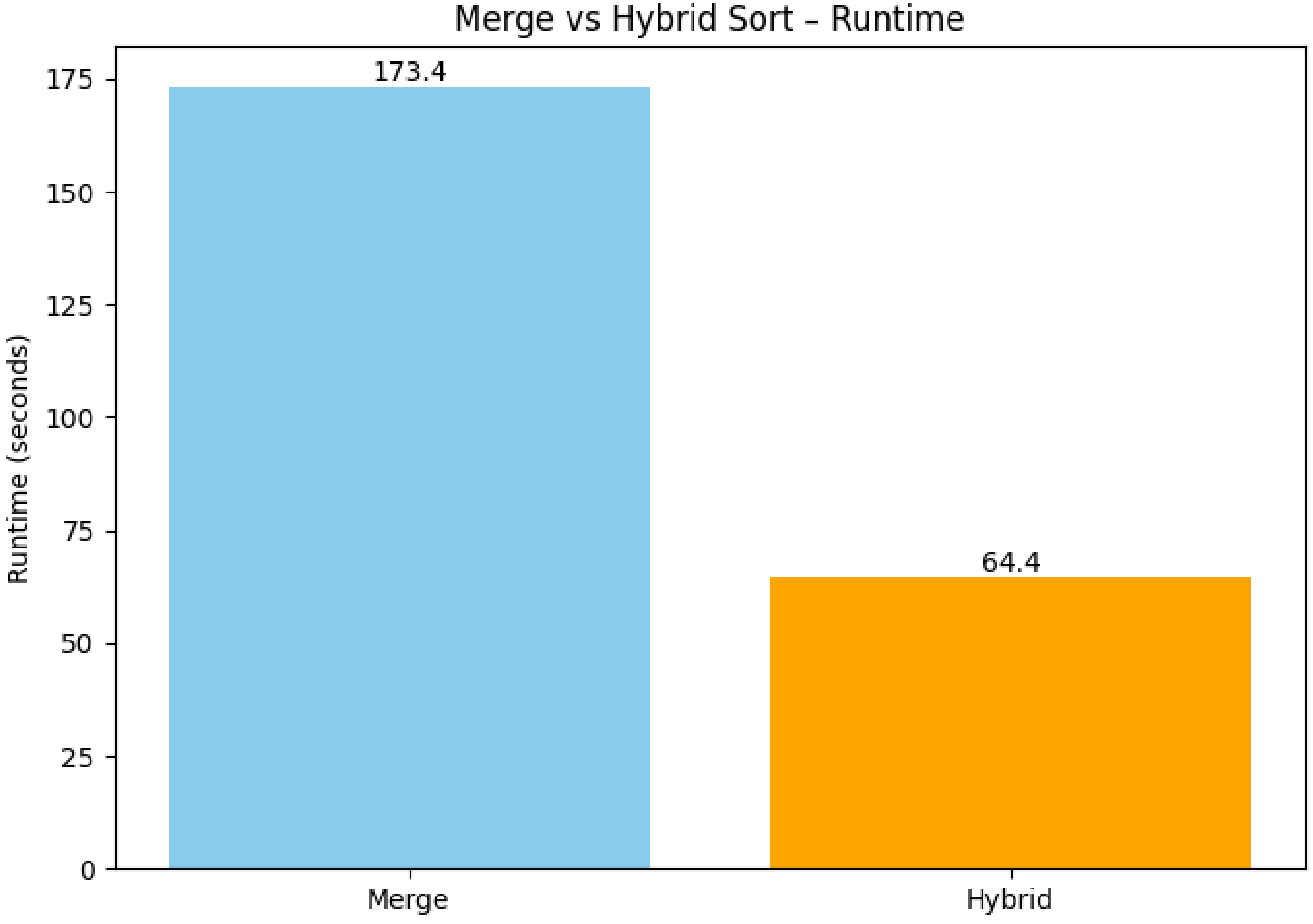
-TAKING THE AVERAGE OF THE 2, IN THE EVENT WE ARE UNSURE OF THE SIZE OF DATA SETS, S SHOULD TAKE A VALUE OF AROUND 43 FOR THE BEST PERFORMANCE.

COMPARING WITH ORIGINAL

HYBRID VS MERGE SORT

10 000 000 INTEGERS
S VALUE = 16





COMPARING WITH ORIGINAL

HYBRID VS MERGE SORT

-SIMILAR IN TERMS OF THE NUMBER OF COMPARISONS, BUT A SIGNIFICANT DIFFERENCE SEEN IN THE RUNTIME.

-RUNTIME IS NOT SOLELY DEPENDENT ON NUMBER OF COMPARISONS, BUT ALSO CONSISTS OF OTHER OPERATIONS LIKE RECURSIVE CALLS AND MEMORY ALLOCATION FOR SUBARRAYS IN MERGESORT.

-HYBRID SORT THEREFORE DRASTICALLY REDUCES THE OVERHEAD ASSOCIATED WITH RECURSIVE CALLS.

THANK YOU