Lab 1 Presentation

Group 1:
Athena, Arun, Afreen

Content

- Abstract
- Hybrid Sort Function (Pseudocode)
- Theoretical Analysis of Time Complexity for Hybrid Sort
- Random Data Generation
- Analysis of Fixed S and Varied n (theory and plots)
- Analysis of different values of S with fixed n (theory and plots)
- Optimal value of S

Abstract

- In Mergesort, recursive calls on small subarrays can become inefficient due to the overhead of multiple recursive calls

- To improve performance, A hybrid of merge and insertion sort is used where it switches to insertion sort when the size of the subarray is lesser than or equal to a certain threshold

- This leverages on the efficiency of insertion sort when dealing with smaller subarrays

Hybrid Sort Function

```
HybridSort(arr, S):
    # Check if array is null or empty
    If array is null or size of array is 0:
        End program

    # Check if array size is less than or equal to threshold S
    If size of array is less than or equal to S:
        Call InsertionSort(array)
        End program

    # If array size is greater than S, perform MergeSort
    Else:
        mid = (start of array + end of array) / 2

        # Recursively apply HybridSort to left half until subarray size is <= S
        Call HybridSort(left half of array, S)

        # Recursively apply HybridSort to right half until subarray size is <= S
        Call HybridSort(right half of array, S)

        # Once subarray size <= S, apply InsertionSort
        If size of subarray is less than or equal to S:
            Call InsertionSort(left half of array)
            Call InsertionSort(right half of array)

     # After sorting subarrays, merge them
        Call Merge(left half of array, right half of array)
```

Theoretical Analysis of Time Complexity for Merge and Insertion Sort

Merge Sort Time Complexity:

Best: $O(n \log n)$

Worst: $O(n \log n)$

Average: $O(n \log n)$

Insertion Sort Time Complexity:

Best: $O(n)$

Worst: $O(n^2)$

Average: $O(n^2)$

Theoretical Analysis of Time Complexity for Hybrid Sort

Best: $O(nlog(\frac{n}{S}))$

Worst: $O(nlog(\frac{n}{S})) + O(nS)$

Average: $O(nlog(\frac{n}{S})) + O(nS)$

Random data Generation

```
sizes = list(range(0,100))
sizes = [2500, 5000, 10000]
max_value = 10000  # Maximum value in the random arrays
```

Passing the parameters

```
datasets = generate_datasets(sizes, max_value)
```

Calling the function to generate

```
def generate_datasets(sizes, max_value):
    datasets = []
    for size in sizes:
        datasets.append(np.random.randint(1, max_value + 1, size=size))
    return datasets
```

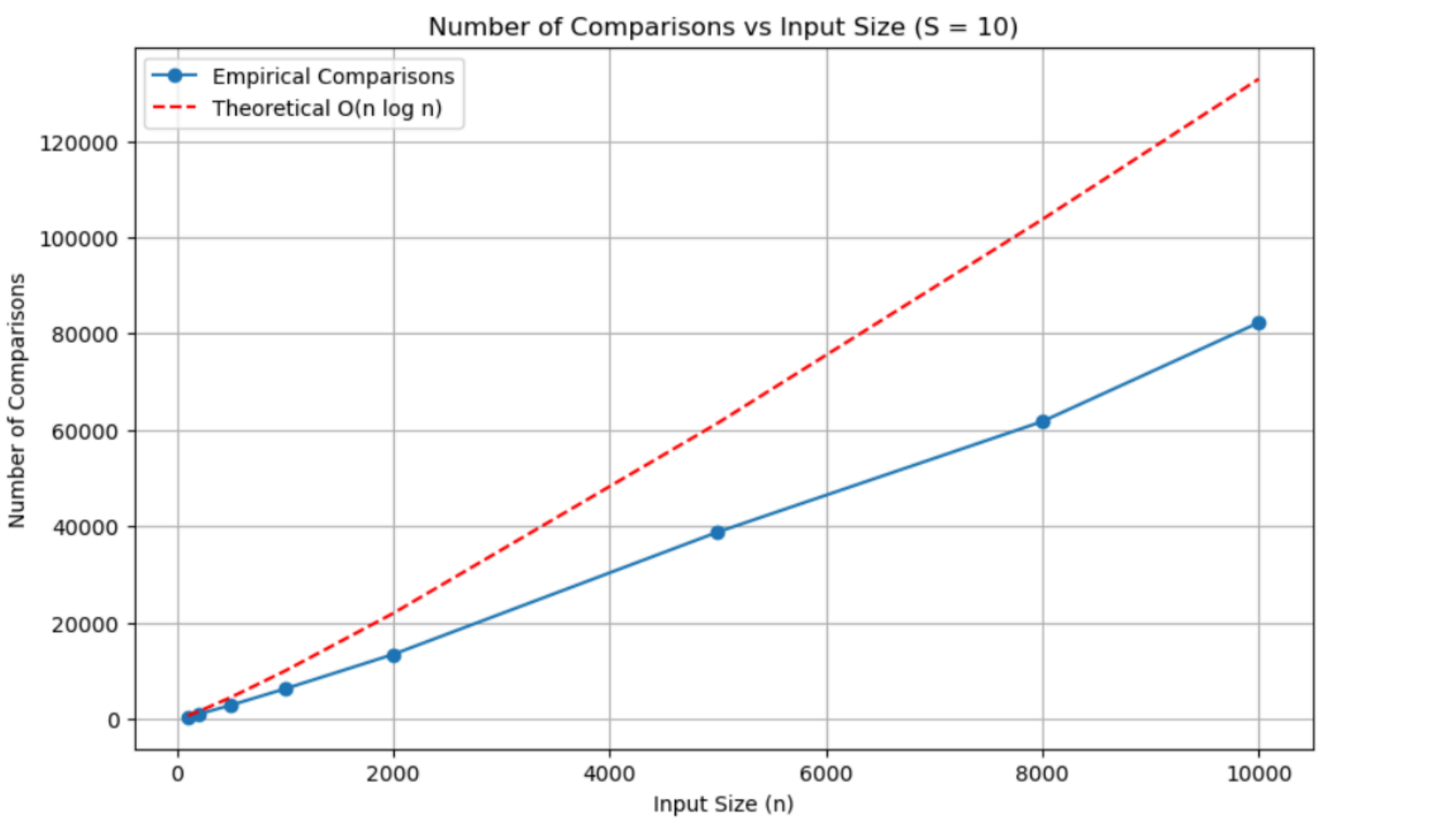
Returns the generated array to be used

Theoretical Analysis of Fixed S and Varied n

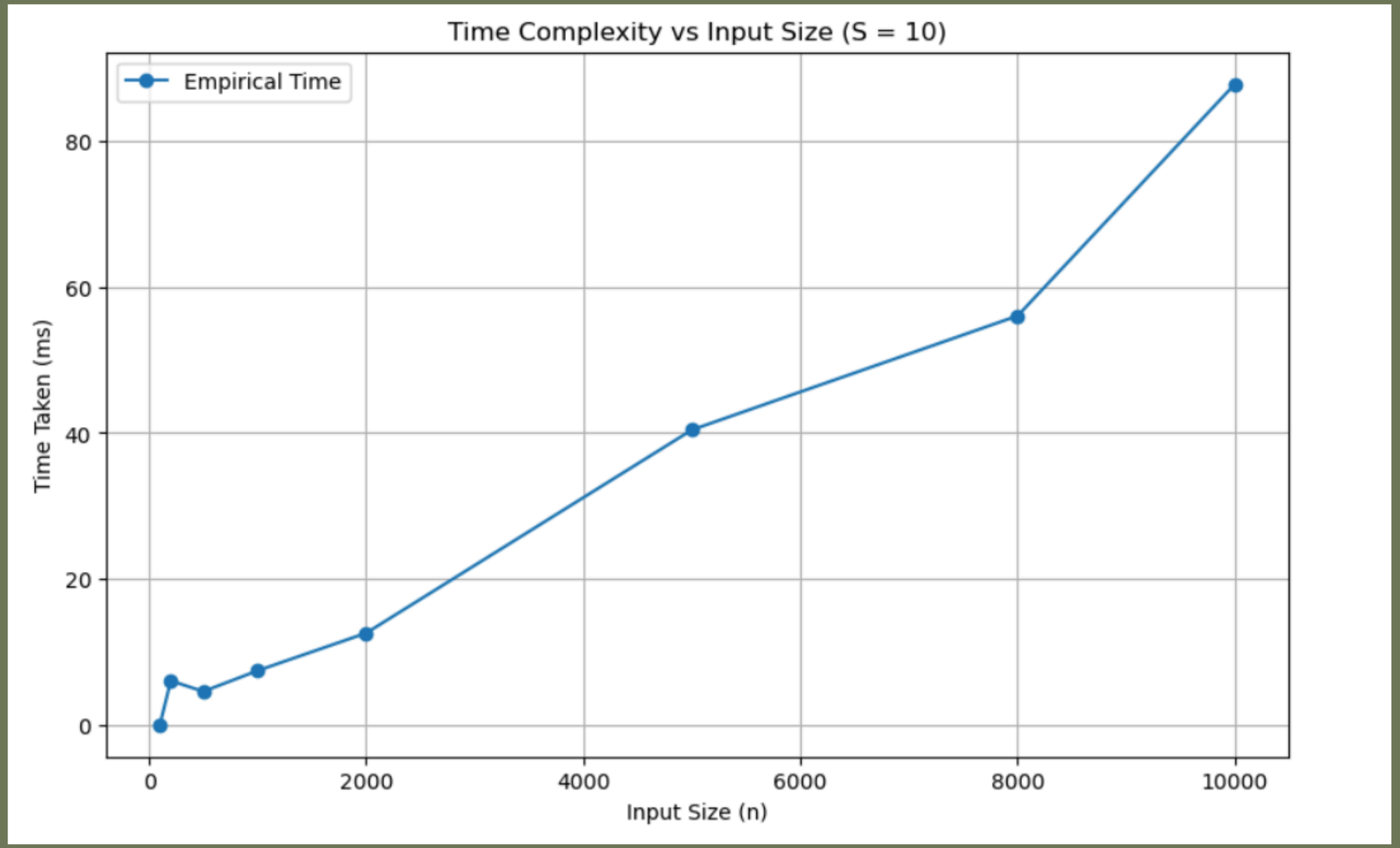
With the value of S fixed, plot the number of key comparisons over different sizes of the input list n. Compare your empirical results with your theoretical analysis of the time complexity.

Due to S being defined as a relatively small number, we decided on S = 10 as our value.

Input Size N against Comparisons



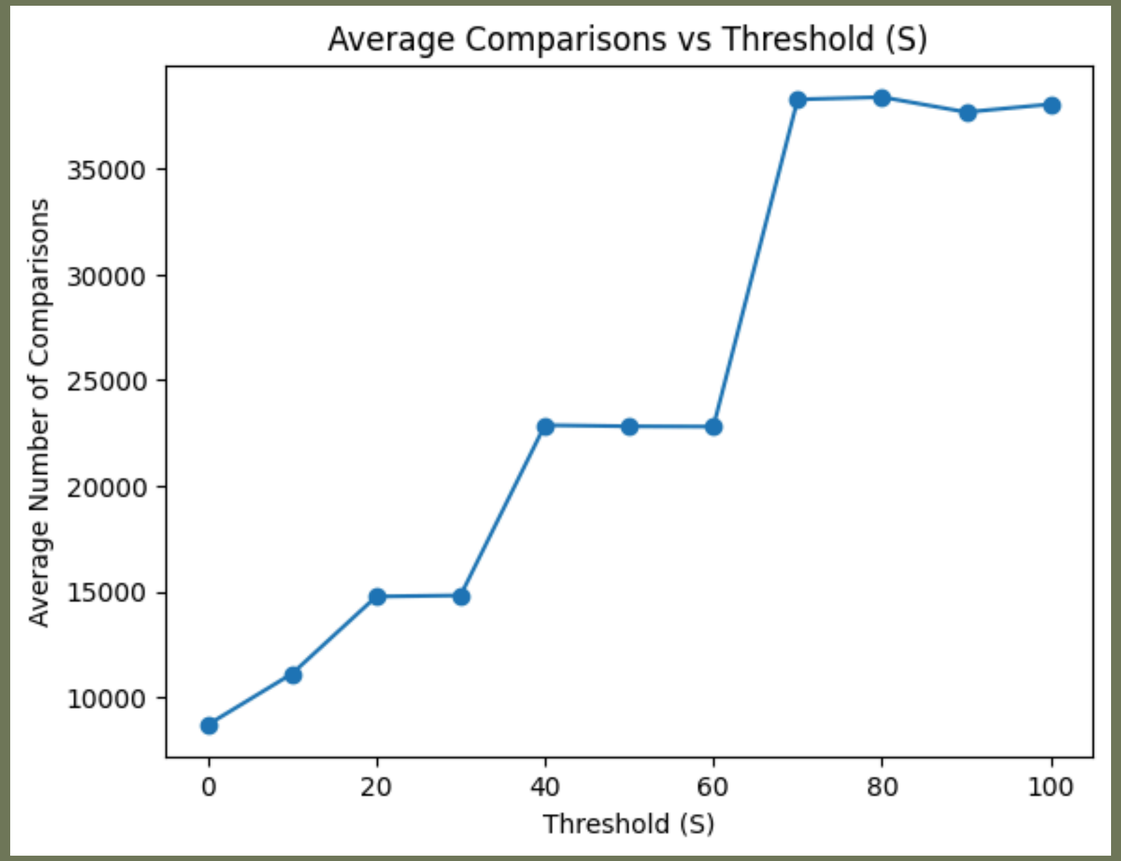
Time complexity against input size



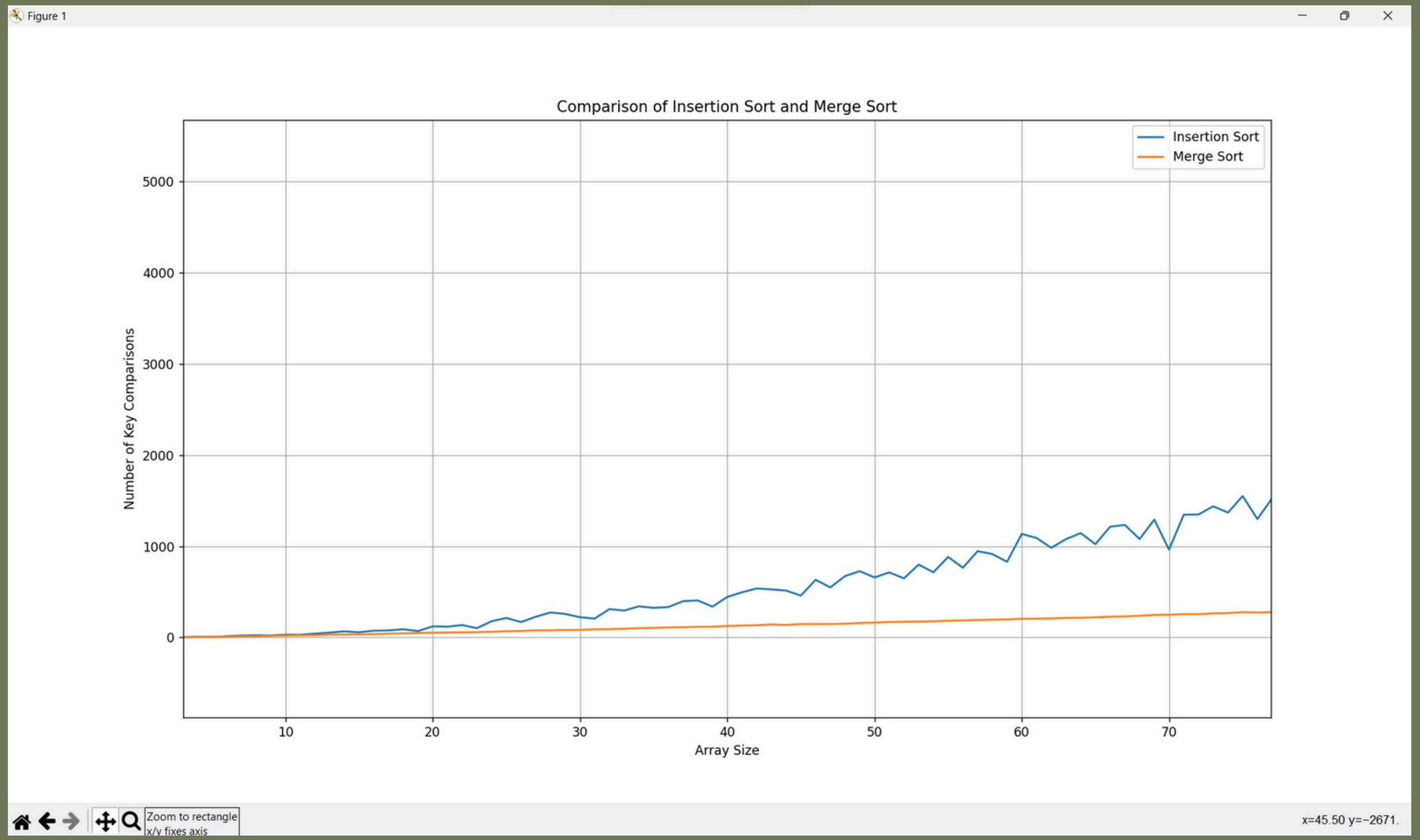
Theoretical Analysis of different values of S with fixed n

Empirical Analysis of different values of S with fixed n

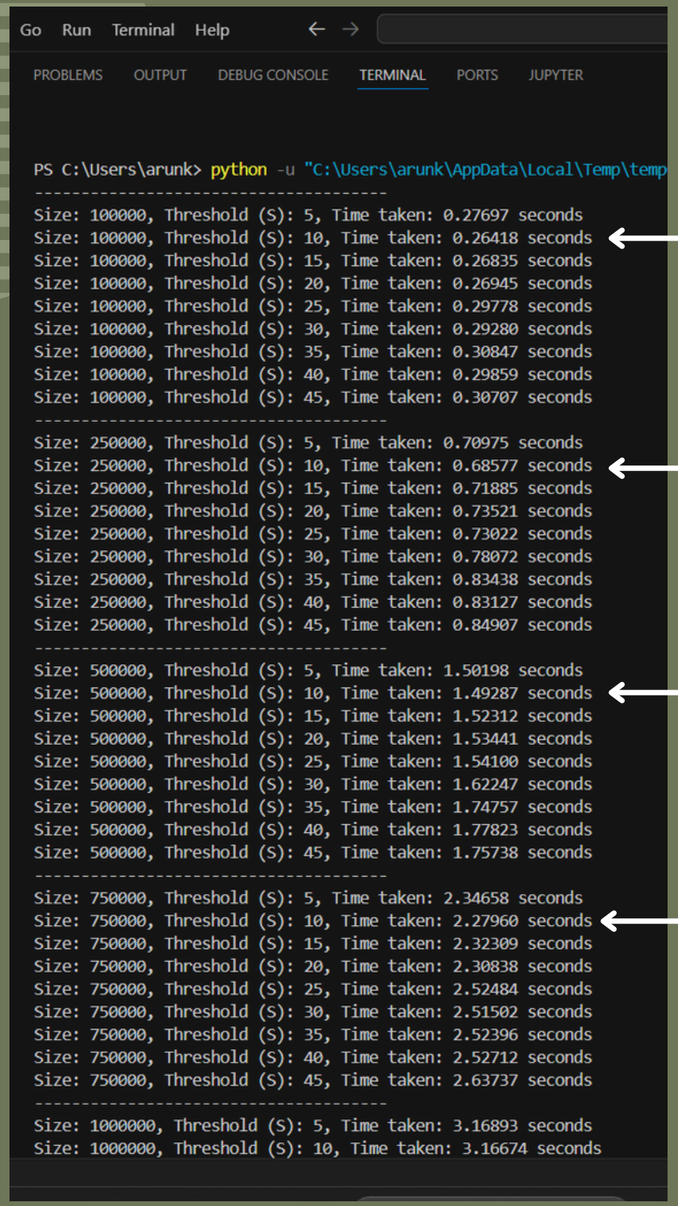
Took the average of 5 runs with fixed size of n = 10000

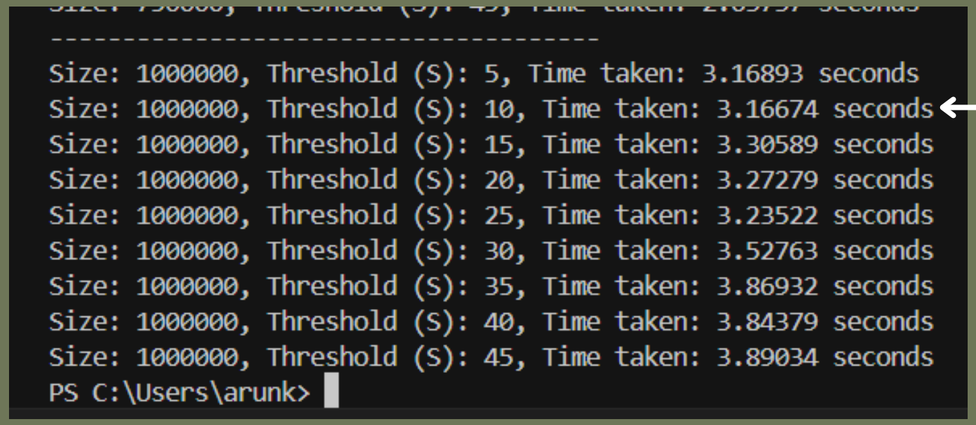


Optimal S value



Optimal S value(CPU time)





For array sizes = 10 000, 25 000, 50 000, 100 000

Optimal S value(CPU time)

```
PS C:\Users\arunk> python -u "c:\Users\arunk\Documents\Arun works\
-------------------------------------
Size: 1000000, Threshold (S): 5, Time taken: 3.22343 seconds
Size: 1000000, Threshold (S): 10, Time taken: 3.15326 seconds
Size: 1000000, Threshold (S): 15, Time taken: 3.22450 seconds
Size: 1000000, Threshold (S): 20, Time taken: 3.24125 seconds
Size: 1000000, Threshold (S): 25, Time taken: 3.25920 seconds
Size: 1000000, Threshold (S): 30, Time taken: 3.48813 seconds
Size: 1000000, Threshold (S): 35, Time taken: 3.74089 seconds
-------------------------------------
Size: 2000000, Threshold (S): 5, Time taken: 6.73913 seconds
Size: 2000000, Threshold (S): 10, Time taken: 6.63365 seconds
Size: 2000000, Threshold (S): 15, Time taken: 6.79644 seconds
Size: 2000000, Threshold (S): 20, Time taken: 6.88581 seconds
Size: 2000000, Threshold (S): 25, Time taken: 6.90331 seconds
Size: 2000000, Threshold (S): 30, Time taken: 7.31424 seconds
Size: 2000000, Threshold (S): 35, Time taken: 7.79565 seconds
-------------------------------------
Size: 5000000, Threshold (S): 5, Time taken: 17.85274 seconds
Size: 5000000, Threshold (S): 10, Time taken: 17.73651 seconds
Size: 5000000, Threshold (S): 15, Time taken: 17.64739 seconds
Size: 5000000, Threshold (S): 20, Time taken: 18.83582 seconds
Size: 5000000, Threshold (S): 25, Time taken: 18.84407 seconds
Size: 5000000, Threshold (S): 30, Time taken: 18.85041 seconds
Size: 5000000, Threshold (S): 35, Time taken: 18.74230 seconds
-------------------------------------
Size: 7500000, Threshold (S): 5, Time taken: 27.97934 seconds
Size: 7500000, Threshold (S): 10, Time taken: 27.31043 seconds
Size: 7500000, Threshold (S): 15, Time taken: 27.87819 seconds
Size: 7500000, Threshold (S): 20, Time taken: 27.76891 seconds
Size: 7500000, Threshold (S): 25, Time taken: 27.86364 seconds
Size: 7500000, Threshold (S): 30, Time taken: 30.88601 seconds
Size: 7500000, Threshold (S): 35, Time taken: 30.98453 seconds
-------------------------------------
Size: 10000000, Threshold (S): 5, Time taken: 37.07623 seconds
Size: 10000000, Threshold (S): 10, Time taken: 37.06110 seconds
Size: 10000000, Threshold (S): 15, Time taken: 37.11428 seconds
Size: 10000000, Threshold (S): 20, Time taken: 38.93371 seconds
Size: 10000000, Threshold (S): 25, Time taken: 39.04322 seconds
Size: 10000000, Threshold (S): 30, Time taken: 39.02599 seconds
Size: 10000000, Threshold (S): 35, Time taken: 39.33857 seconds
PS C:\Users\arunk>
```

For array sizes = 1000 000, 2000 000, 5000 000, 7500 000 10 000 000

2nd fastest for array size 5000 000

The optimal threshold value(S) = 10

Optimal S value(CPU time)

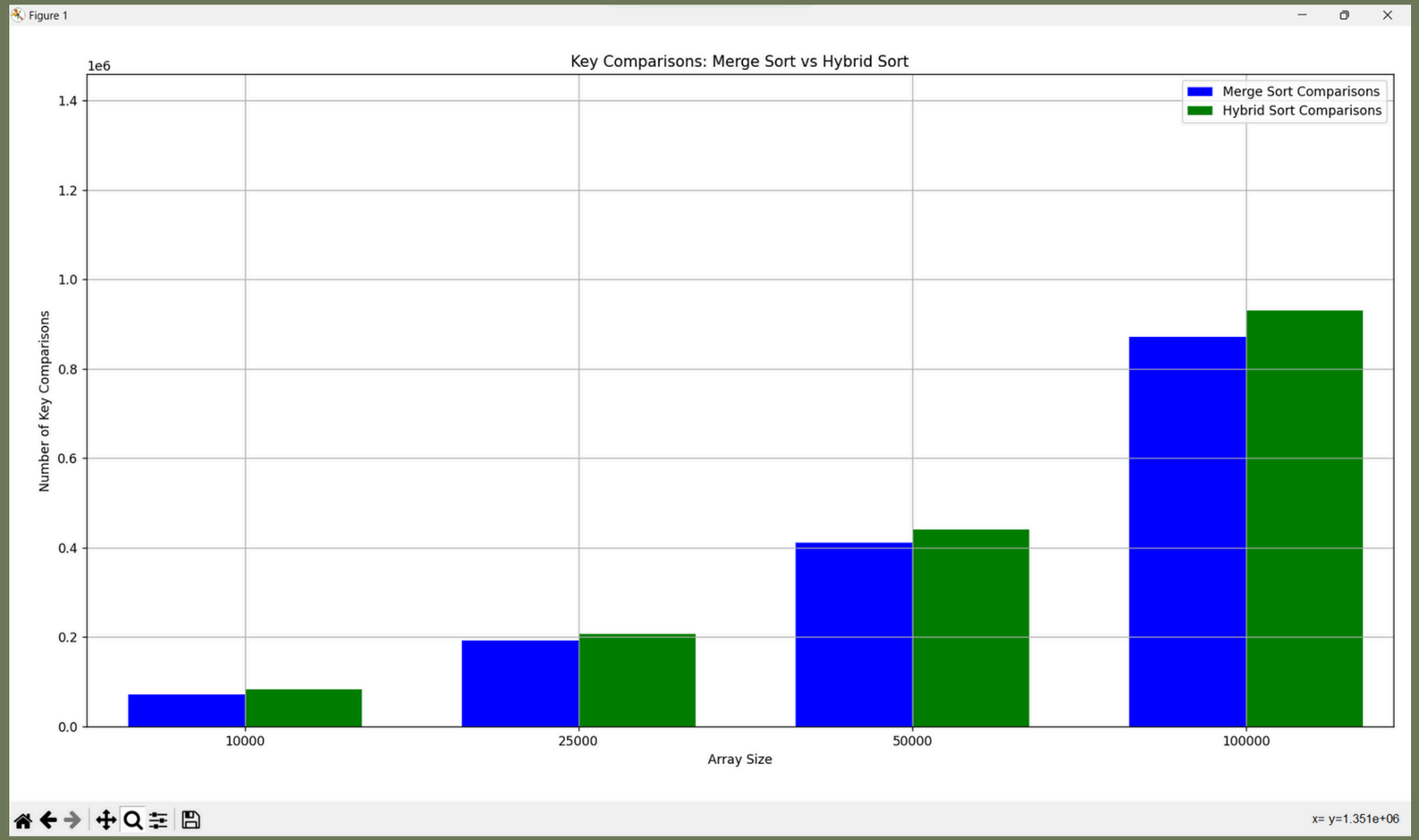
```
PS C:\Users\arunk> python -u "c:\Users\arunk\Documents\Arun works\UNI WORKS\Y2S1\SC2001\
-------------------------------------
Size: 10000, Threshold (S): 5, Time taken: 0.05777 seconds, Key comparisons: 74904
Size: 10000, Threshold (S): 10, Time taken: 0.04423 seconds, Key comparisons: 81908
-------------------------------------
Size: 50000, Threshold (S): 5, Time taken: 0.25616 seconds, Key comparisons: 427359
Size: 50000, Threshold (S): 10, Time taken: 0.24605 seconds, Key comparisons: 440289
-------------------------------------
Size: 100000, Threshold (S): 5, Time taken: 0.54282 seconds, Key comparisons: 904687
Size: 100000, Threshold (S): 10, Time taken: 0.53761 seconds, Key comparisons: 930456
PS C:\Users\arunk>
```

Hybrid sort vs Merge sort

CPU time

```
116    merge_comparisons, hybrid_comparisons, merge_times, hyb
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\arunk> python -u "C:\Users\arunk\AppData\Local\Temp\t
Array Size: 10000
  Merge Sort - Comparisons: 71593, Time: 0.03211 seconds
  Hybrid Sort - Comparisons: 82231, Time: 0.01510 seconds

Array Size: 25000
  Merge Sort - Comparisons: 192989, Time: 0.07101 seconds
  Hybrid Sort - Comparisons: 207813, Time: 0.07126 seconds

Array Size: 50000
  Merge Sort - Comparisons: 410964, Time: 0.14415 seconds
  Hybrid Sort - Comparisons: 440435, Time: 0.12192 seconds

Array Size: 100000
  Merge Sort - Comparisons: 871654, Time: 0.32452 seconds
  Hybrid Sort - Comparisons: 930442, Time: 0.27263 seconds

Array Size: 1000000
  Merge Sort - Comparisons: 10415223, Time: 3.64175 seconds
  Hybrid Sort - Comparisons: 11200871, Time: 3.23687 seconds

PS C:\Users\arunk>
```

Key comparisons



THANK YOU!