ALGORITHM DESIGN AND ANALYSIS PRESENTATION

Lab section: T13L
Lab Lecturer Name: Angeline Pang
Group No: AngelineT13L E

Group Member:

Group Member's ID	Name
1221102540	CHIN ZHEN HO
1221102007	ERIC TEOH WEI XIANG
1221101777	BERNARD RYAN SIM KANG XUAN
1221103819	JORDAN LIM WEI ZHI

Contribution:

Name	Contribution
CHIN ZHEN HO	merge_sort_step.py merge_sort_step.cpp merge_sort.py merge_sort.cpp
ERIC TEOH WEI XIANG	dataset_generator.py quick_sort_step.py quick_sort.py
BERNARD RYAN SIM KANG XUAN	binary_search_step.cpp binary_search_step.py binary_search.cpp binary_search.py
JORDAN LIM WEI ZHI	quick_sort_step.cpp quick_sort.cpp

Introduction

This project focuses on evaluating sorting and searching algorithms in the context of building an AVL Tree using arrays. We compare Merge Sort and Quick Sort (last-element pivot), and analyze Binary Search in best, average, and worst-case scenarios. The goal is to identify the most efficient approach through both theoretical and experimental analysis using datasets of various sizes and two programming languages.

Assignment Overview

Objective: Evaluate sorting and searching algorithms for AVL Tree implementation using arrays

Algorithms Studied

- Sorting: Merge Sort vs. Quick Sort (last-element pivot)
- Searching: Binary Search (best, average, worst-case)

Implementation Details

- Developed in two programming languages
- No built-in sorting or searching functions used
- Custom dataset generator created for testing

Analysis Focus

- Time and space complexities (theoretical & empirical)
- Language-based performance differences
- Hardware impact on algorithm efficiency

Chosen Programming Languages & Data Structures

- **Languages Used:**
  - **C++:** for high performance and precise control
  - **Python:** for faster development and clear syntax
- **Data Structures:**
  - **Array/List:** used for sorting and searching
  - **CSV Files:** store (integer, string) pairs as input/output
- **Rules Followed:**
  - No built-in sort/search functions
  - No advanced data structures (e.g., TreeMap, PriorityQueue)

DATASET GENERATION: ALGORITHM & REQUIREMENTS

Our dataset_generator Algorithm

- Developed in Python to create randomized input datasets for sorting and searching experiments.
- Generates unique positive integers paired with random 5-letter lowercase strings.
- Output saved in .csv format, matching the structure of dataset_sample_1000.csv.

Key Requirements

- Random Order: All elements are unsorted prior to testing.
- Uniqueness: Integers are non-repeating within a dataset.
- Scalability: Dataset sizes vary (10+ sizes), with the largest large enough to show a ≥ 60s runtime gap between Merge Sort and Quick Sort.
- Range: Integers generated between 1 and 2,000,000,000.

Dataset Format & Example

- Format: (integer, string) per row, separated by a comma
- Example: 1606128894,sziod

Output File

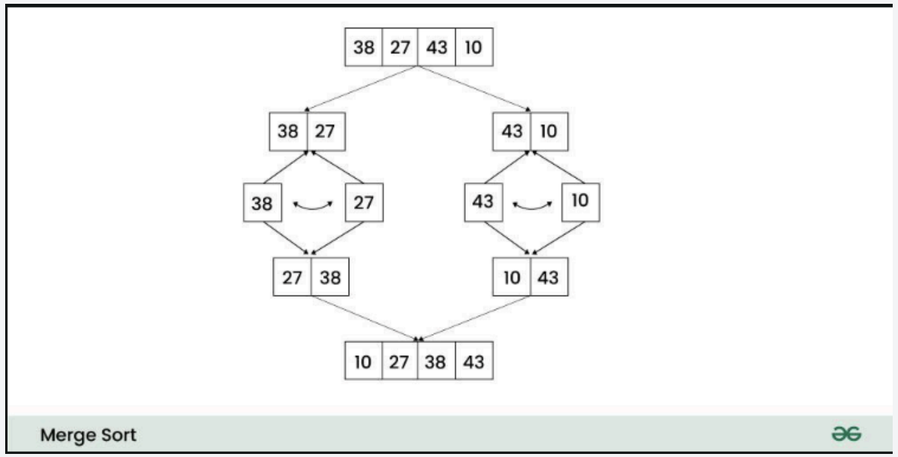
- File name format: dataset_<n>.csv
- Example: dataset_1000000.csv for 1 million records

THEORETICAL ANALYSIS: MERGE SORT

Merge Sort is a highly efficient, comparison-based sorting algorithm that operates on the Divide and Conquer paradigm.

Three Steps:

- **Divide:** The unsorted list of n elements is divided into two sub-lists of about n/2 elements each. This step takes O(1) time.
- **Recur:** Each sub-list is recursively sorted using Merge Sort.
- **Conquer (Merge):** The two sorted sub-lists are merged back into one sorted list. This merging process is the key to Merge Sort's efficiency and takes O(n) time, as each element is processed once.



Algorithm	Best Case	Average Case	Worst Case	Space Complexity
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)

Time Complexity:

- **Running Time:** Merge Sort consistently has a time complexity of O(n log n) in all cases (best, average, and worst).
  - This is derived from the recurrence relation T(n) = 2T(n/2) + O(n), which accounts for two recursive calls on half the size and O(n) for the merging step.

THEORETICAL ANALYSIS: MERGE SORT

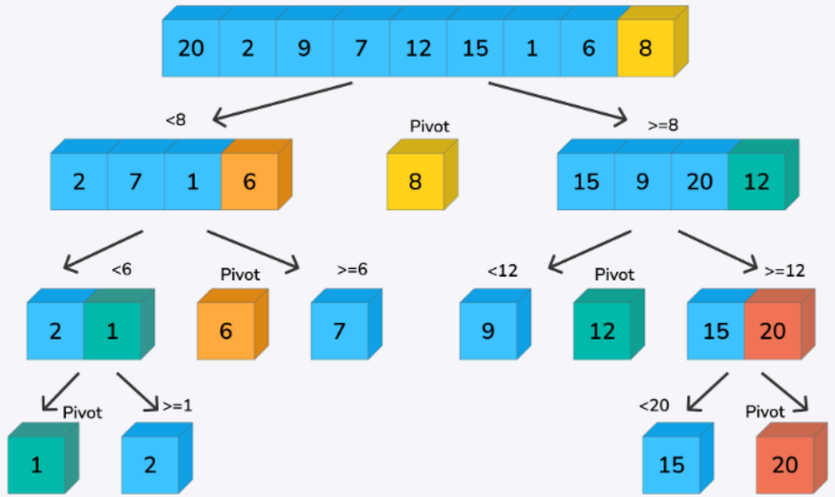
Key Characteristics & Properties:

- **Stability:** Merge sort is a stable sorting algorithm, which means it maintains the relative order of equal elements in the input array.
- **Not-In-Place:** It is generally not an in-place algorithm. It requires additional temporary storage space (typically O(n) auxiliary space) for the merging step.
- **Data Access Pattern:** It accesses data in a sequential manner, making it particularly suitable for sorting data stored on slow-access media like disks.

THEORETICAL ANALYSIS: QUICK SORT

How Quick Sort Works

- Quick Sort is also a Divide and Conquer Algorithm
- Select a pivot (last element as pivot).
- Partition the array into two subarrays:
  - ➤ Elements less than pivot
  - ➤ Elements greater than or equal to pivot
- Recursively apply Quick Sort to both subarrays.
- No additional arrays are used — sorting is done in-place.



Time Complexity

- Best Case: O(n log n) → Pivot divides array evenly.
- Average Case: O(n log n) → Typically balanced partitioning.
- Worst Case: O(n^2) → Pivot causes highly unbalanced partitions

Case	Time Complexity	Space Complexity
Best	O(n log n)	O(log n)
Average	O(n log n)	O(log n)
Worst	O(n^2)	O(n)

Space Complexity

- Auxiliary Space:
  - ➤ O(log n) for best/average case (recursive calls)
  - ➤ O(n) in worst case (unbalanced recursion)

THEORETICAL ANALYSIS: BINARY SEARCH

- Binary Search is an efficient algorithm to find a target in a sorted list.
- It starts by checking the middle element of the list.
- If the target equals the middle element, the search ends successfully.
- If the target is less than the middle element, it repeats the process on the left half of the list.
- If the target is greater, it searches the right half.
- It keeps halving the search range until the target is found or the range becomes empty (target not found).
- Binary Search reduces the problem size by half each time, making it fast with time complexity O(log n).

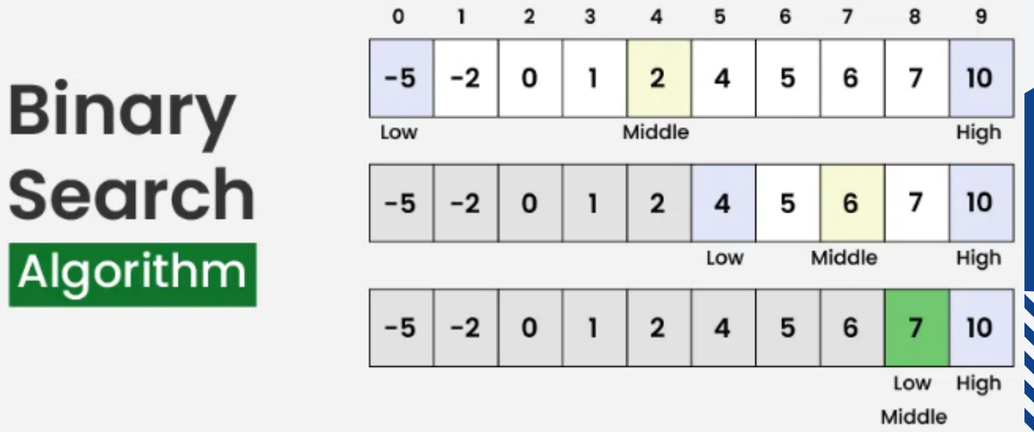
Case	Time Complexity
Best	O(1)
Average	O(log n)
Worst	O(log n)

Key Requirement:
Binary Search only works correctly if the list is already sorted.
Time Complexity:

- Best case: O(1) → target is found at the middle immediately.
- Average/Worst case: O(log n) → each step halves the search space, so it takes at most log n steps. Worst case happens if the target is at the edges or missing.

Space Complexity:

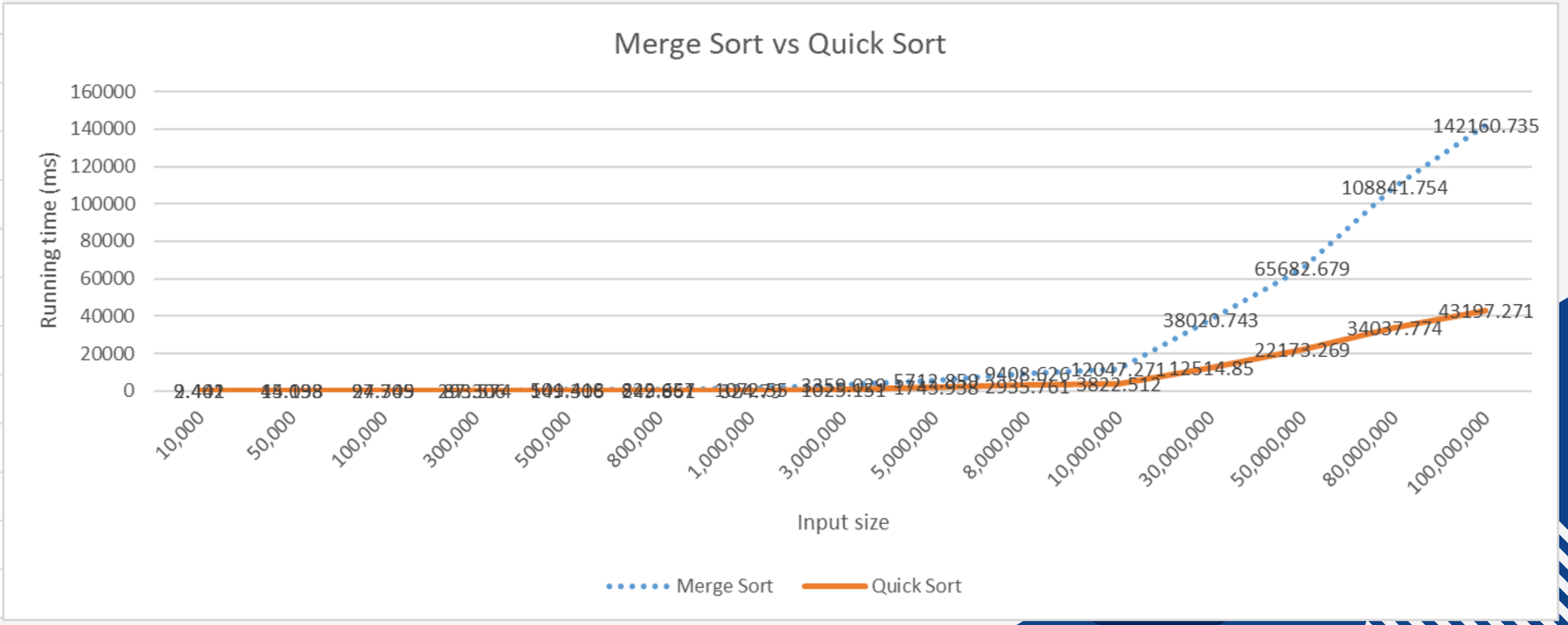
- For an iterative version, the extra space is mainly the array itself → O(n).
- The algorithm itself uses constant extra memory (excluding the input).



EXPERIMENT STUDY : SORTING ALGORITHMS (C++)

Tester : BERNARD RYAN SIM KANG XUAN

	Running time (ms)	
Input size	Merge Sort	Quick Sort
10,000	9.441	2.402
50,000	45.193	14.038
100,000	94.749	27.305
300,000	293.574	87.306
500,000	501.418	149.306
800,000	822.657	249.861
1,000,000	1072.55	324.79
3,000,000	3359.029	1025.151
5,000,000	5712.859	1743.938
8,000,000	9408.626	2935.761
10,000,000	12047.271	3822.512
30,000,000	38020.743	12514.85
50,000,000	65682.679	22173.269
80,000,000	108841.754	34037.774
100,000,000	142160.735	43197.271



PROOF OF EXPERIMENT STUDY : SORTING ALGORITHMS (C++)

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_10000.csv
Running time: 9.441 ms
Sorted data written to merge_sort_10000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_50000.csv
Running time: 45.193 ms
Sorted data written to merge_sort_50000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_100000.csv
Running time: 94.749 ms
Sorted data written to merge_sort_100000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_300000.csv
Running time: 293.574 ms
Sorted data written to merge_sort_300000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_500000.csv
Running time: 501.418 ms
Sorted data written to merge_sort_500000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_800000.csv
Running time: 822.657 ms
Sorted data written to merge_sort_800000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_1000000.csv
Running time: 1072.550 ms
Sorted data written to merge_sort_1000000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_3000000.csv
Running time: 3359.029 ms
Sorted data written to merge_sort_3000000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_5000000.csv
Running time: 5712.859 ms
Sorted data written to merge_sort_5000000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_8000000.csv
Running time: 9408.626 ms
Sorted data written to merge_sort_8000000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_10000000.csv
Running time: 12047.271 ms
Sorted data written to merge_sort_10000000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_30000000.csv
Running time: 38020.743 ms
Sorted data written to merge_sort_30000000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_50000000.csv
Running time: 65682.679 ms
Sorted data written to merge_sort_50000000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_80000000.csv
Running time: 108841.754 ms
Sorted data written to merge_sort_80000000.csv
```

```
PS D:\Algo_Assignment\C++> ./a
Enter CSV file name: dataset_100000000.csv
Running time: 142160.735 ms
Sorted data written to merge_sort_100000000.csv
PS D:\Algo_Assignment\C++>
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_8000000.csv
Running time: 2935.761 ms
Sorted data has been saved to file: quick_sort_8000000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_10000000.csv
Running time: 3822.512 ms
Sorted data has been saved to file: quick_sort_10000000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_30000000.csv
Running time: 12514.850 ms
Sorted data has been saved to file: quick_sort_30000000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_50000000.csv
Running time: 22173.269 ms
Sorted data has been saved to file: quick_sort_50000000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_80000000.csv
Running time: 34037.774 ms
Sorted data has been saved to file: quick_sort_80000000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_100000000.csv
Running time: 43197.271 ms
Sorted data has been saved to file: quick_sort_100000000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_10000.csv
Running time: 2.402 ms
Sorted data has been saved to file: quick_sort_10000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_50000.csv
Running time: 14.038 ms
Sorted data has been saved to file: quick_sort_50000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_100000.csv
Running time: 27.305 ms
Sorted data has been saved to file: quick_sort_100000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_300000.csv
Running time: 87.306 ms
Sorted data has been saved to file: quick_sort_300000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_500000.csv
Running time: 149.306 ms
Sorted data has been saved to file: quick_sort_500000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_800000.csv
Running time: 249.861 ms
Sorted data has been saved to file: quick_sort_800000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_1000000.csv
Running time: 324.790 ms
Sorted data has been saved to file: quick_sort_1000000.csv
```

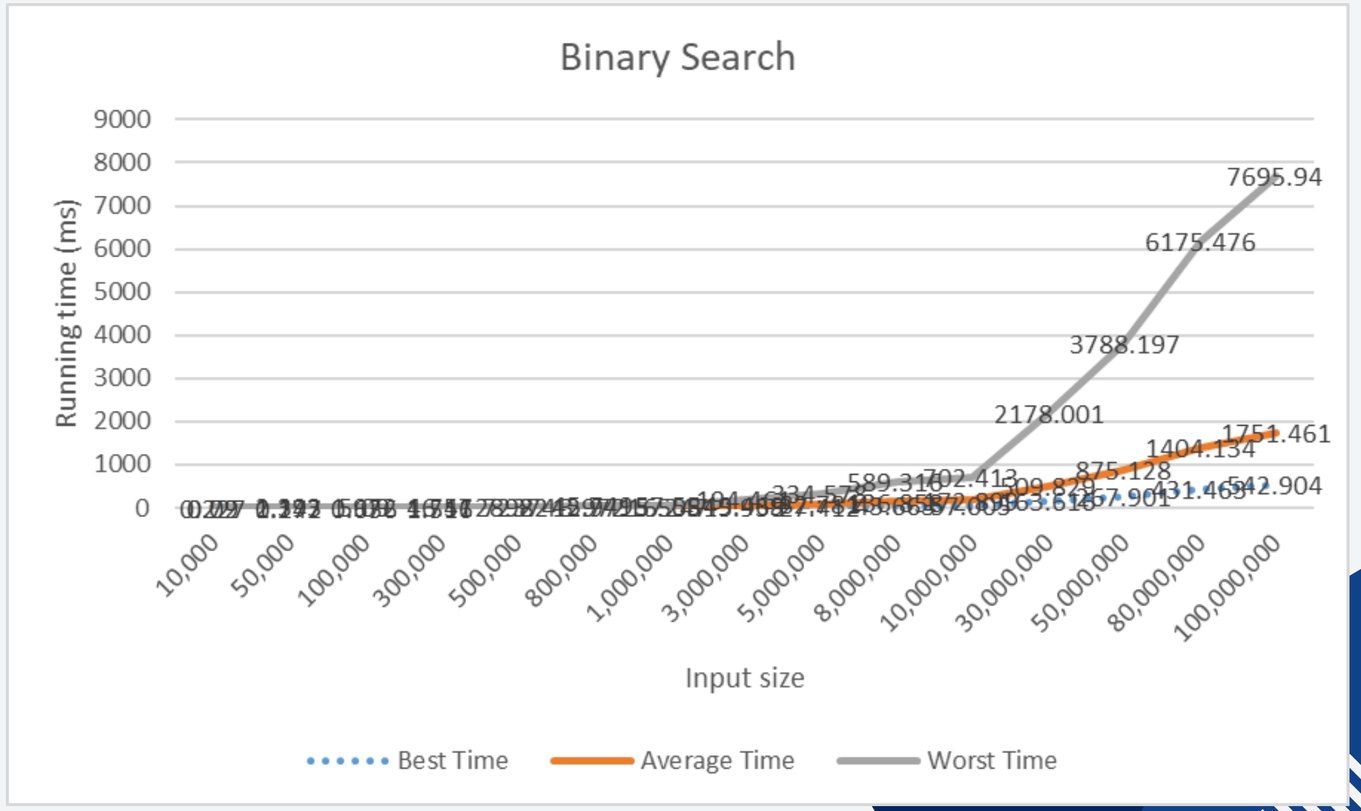
```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_3000000.csv
Running time: 1025.151 ms
Sorted data has been saved to file: quick_sort_3000000.csv
```

```
PS D:\Algo_Assignment\C++> ./b
Enter CSV file name: dataset_5000000.csv
Running time: 1743.938 ms
Sorted data has been saved to file: quick_sort_5000000.csv
```

EXPERIMENT STUDY : BINARY SEARCH (C++)

Tester : BERNARD RYAN SIM KANG XUAN

	Running time (ms)		
Input size	Best Time	Average Time	Worst Time
10,000	0.09	0.227	0.72
50,000	0.272	1.147	2.393
100,000	0.636	1.972	5.09
300,000	1.516	4.741	16.57
500,000	2.8	7.922	28.374
800,000	3.972	12.971	45.749
1,000,000	5.508	16.247	57.55
3,000,000	15.938	49.469	194.461
5,000,000	27.412	82.781	334.578
8,000,000	43.663	136.858	589.316
10,000,000	57.005	172.899	702.413
30,000,000	163.616	509.829	2178.001
50,000,000	267.901	875.128	3788.197
80,000,000	431.463	1404.134	6175.476
100,000,000	542.904	1751.461	7695.94



PROOF OF EXPERIMENT STUDY : BINARY SEARCH (C++)

binary_search_10000.txt

```
Best case target: 998263956
Best case time: 0.090 ms

Average case time (10% random targets): 0.227 ms

Worst case target: -1
Worst case time: 0.720 ms
```

binary_search_50000.txt

```
Best case target: 992524046
Best case time: 0.272 ms

Average case time (10% random targets): 1.147 ms

Worst case target: -1
Worst case time: 2.393 ms
```

binary_search_100000.txt

```
Best case target: 992856939
Best case time: 0.636 ms

Average case time (10% random targets): 1.972 ms

Worst case target: -1
Worst case time: 5.090 ms
```

binary_search_300000.txt

```
Best case target: 1001558385
Best case time: 1.516 ms

Average case time (10% random targets): 4.741 ms

Worst case target: -1
Worst case time: 16.570 ms
```

binary_search_500000.txt

```
Best case target: 1001128920
Best case time: 2.800 ms

Average case time (10% random targets): 7.922 ms

Worst case target: -1
Worst case time: 28.374 ms
```

binary_search_800000.txt

```
Best case target: 1002267200
Best case time: 3.972 ms

Average case time (10% random targets): 12.971 ms

Worst case target: -1
Worst case time: 45.749 ms
```

binary_search_1000000.txt

```
Best case target: 999851588
Best case time: 5.508 ms

Average case time (10% random targets): 16.247 ms

Worst case target: -1
Worst case time: 57.550 ms
```

binary_search_3000000.txt

```
Best case target: 999518623
Best case time: 15.938 ms

Average case time (10% random targets): 49.469 ms

Worst case target: -1
Worst case time: 194.461 ms
```

binary_search_5000000.txt

```
Best case target: 1000270565
Best case time: 27.412 ms

Average case time (10% random targets): 82.781 ms

Worst case target: -1
Worst case time: 334.578 ms
```

binary_search_8000000.txt

```
Best case target: 999304139
Best case time: 43.663 ms

Average case time (10% random targets): 136.858 ms

Worst case target: -1
Worst case time: 589.316 ms
```

binary_search_10000000.txt

```
Best case target: 999544202
Best case time: 57.005 ms

Average case time (10% random targets): 172.899 ms

Worst case target: -1
Worst case time: 702.413 ms
```

binary_search_30000000.txt

```
Best case target: 999859108
Best case time: 163.616 ms

Average case time (10% random targets): 509.829 ms

Worst case target: -1
Worst case time: 2178.001 ms
```

binary_search_50000000.txt

```
Best case target: 999827423
Best case time: 267.901 ms

Average case time (10% random targets): 875.128 ms

Worst case target: -1
Worst case time: 3788.197 ms
```

binary_search_80000000.txt

```
Best case target: 1000137663
Best case time: 431.463 ms

Average case time (10% random targets): 1404.134 ms

Worst case target: -1
Worst case time: 6175.476 ms
```

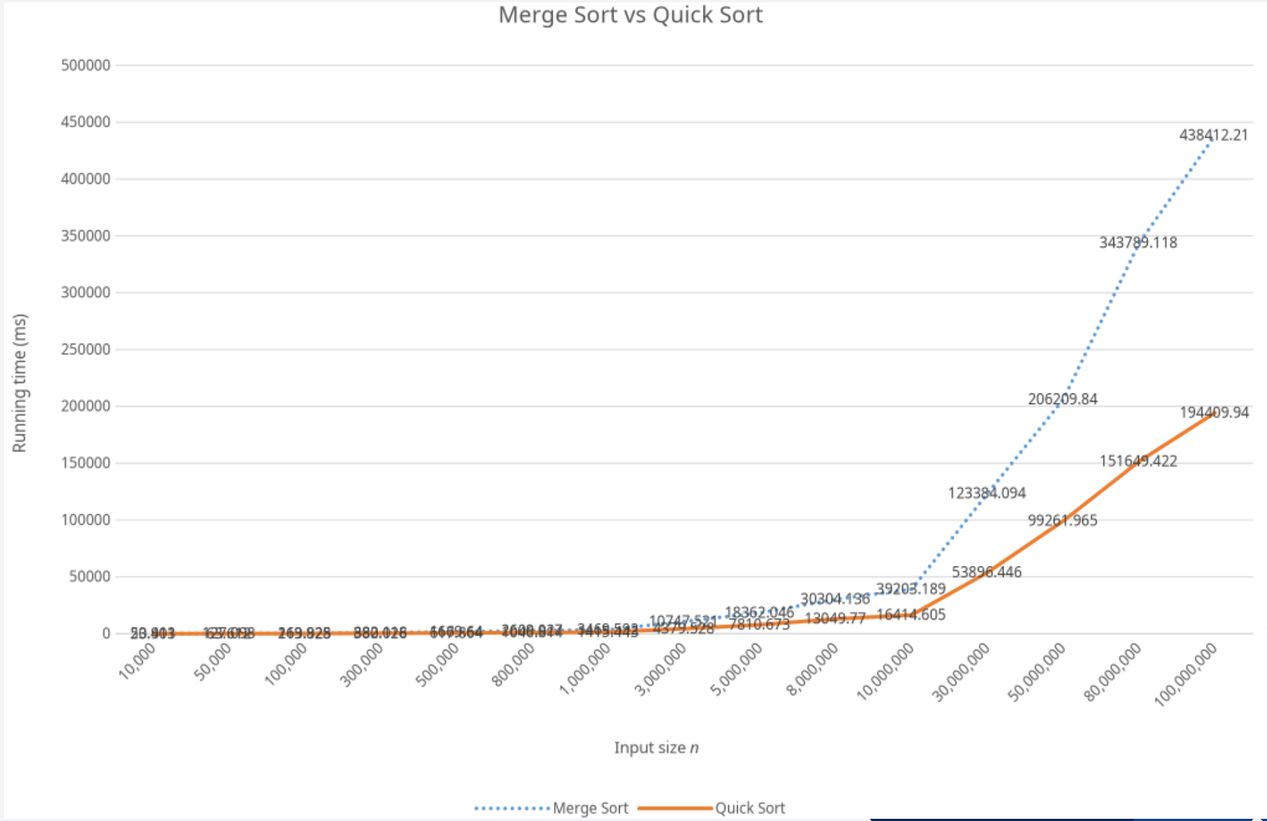
binary_search_100000000.txt

```
Best case target: 1000007845
Best case time: 542.904 ms

Average case time (10% random targets): 1751.461 ms

Worst case target: -1
Worst case time: 7695.940 ms
```

EXPERIMENT STUDY : SORTING ALGORITHMS (C++)

Tester : JORDAN LIM WEI ZHI

	Running time (ms)	
Input size n	Merge Sort	Quick Sort
10,000	50.901	23.413
50,000	127.098	63.612
100,000	269.925	113.828
300,000	882.118	380.026
500,000	1669.64	617.864
800,000	2609.027	1046.814
1,000,000	3469.592	1415.443
3,000,000	10747.521	4379.528
5,000,000	18362.046	7810.673
8,000,000	30304.136	13049.77
10,000,000	39203.189	16414.605
30,000,000	123384.094	53896.446
50,000,000	206209.84	99261.965
80,000,000	343789.118	151649.422
100,000,000	438412.21	194409.94



PROOF OF EXPERIMENT STUDY : SORTING ALGORITHMS (C++)

```
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_10000.csv
Running time: 23.413 ms
Sorted data has been saved to file: quick_sort_10000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_50000.csv
Running time: 63.612 ms
Sorted data has been saved to file: quick_sort_50000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_100000.csv
Running time: 113.828 ms
Sorted data has been saved to file: quick_sort_100000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_300000.csv
Running time: 380.026 ms
Sorted data has been saved to file: quick_sort_300000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_500000.csv
Running time: 617.864 ms
Sorted data has been saved to file: quick_sort_500000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_800000.csv
Running time: 1046.814 ms
Sorted data has been saved to file: quick_sort_800000.csv
[user@unknown C++]$
```

```
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_1000000.csv
Running time: 1415.443 ms
Sorted data has been saved to file: quick_sort_1000000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_3000000.csv
Running time: 4379.528 ms
Sorted data has been saved to file: quick_sort_3000000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_5000000.csv
Running time: 7810.673 ms
Sorted data has been saved to file: quick_sort_5000000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_8000000.csv
Running time: 13049.770 ms
Sorted data has been saved to file: quick_sort_8000000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_10000000.csv
Running time: 16414.605 ms
Sorted data has been saved to file: quick_sort_10000000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_50000000.csv
Running time: 99261.965 ms
Sorted data has been saved to file: quick_sort_50000000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_30000000.csv
Running time: 53896.446 ms
Sorted data has been saved to file: quick_sort_30000000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_80000000.csv
Running time: 151649.422 ms
Sorted data has been saved to file: quick_sort_80000000.csv
[user@unknown C++]$ ./quick_sort.out
Enter CSV file name: dataset_100000000.csv
Running time: 194409.940 ms
Sorted data has been saved to file: quick_sort_100000000.csv
[user@unknown C++]$
```

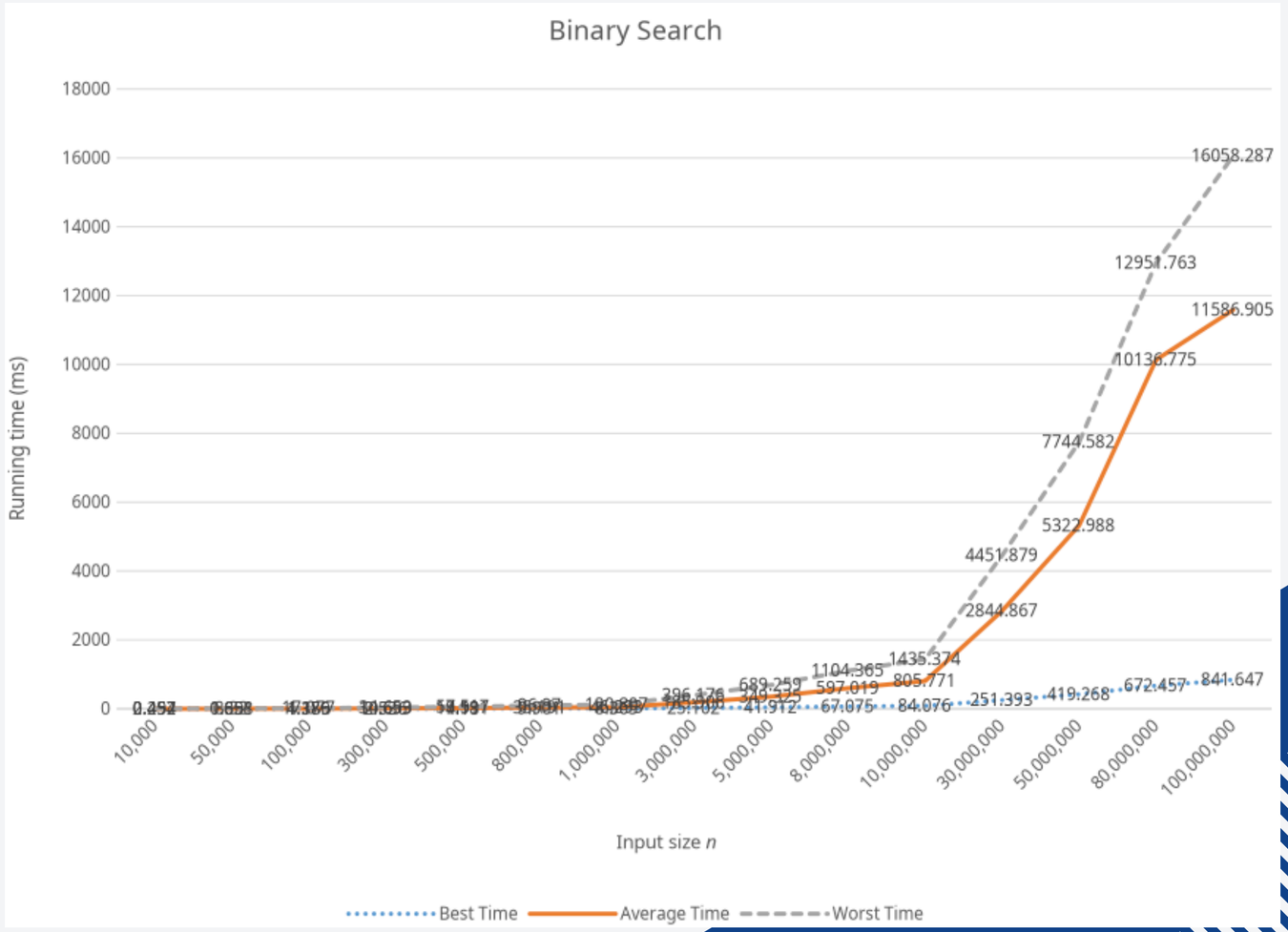
```
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_100000000.csv
Running time: 438412.210 ms
Sorted data written to merge_sort_100000000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_80000000.csv
Running time: 343789.118 ms
Sorted data written to merge_sort_80000000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_50000000.csv
Running time: 206209.840 ms
Sorted data written to merge_sort_50000000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_30000000.csv
Running time: 123384.094 ms
Sorted data written to merge_sort_30000000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_10000000.csv
Running time: 39203.189 ms
Sorted data written to merge_sort_10000000.csv
```

```
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_8000000.csv
Running time: 30304.136 ms
Sorted data written to merge_sort_8000000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_5000000.csv
Running time: 18362.046 ms
Sorted data written to merge_sort_5000000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_3000000.csv
Running time: 10747.521 ms
Sorted data written to merge_sort_3000000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_1000000.csv
Running time: 3469.592 ms
Sorted data written to merge_sort_1000000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_800000.csv
Running time: 2609.027 ms
Sorted data written to merge_sort_800000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_500000.csv
Running time: 1669.640 ms
Sorted data written to merge_sort_500000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_300000.csv
Running time: 882.118 ms
Sorted data written to merge_sort_300000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_100000.csv
Running time: 269.925 ms
Sorted data written to merge_sort_100000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_50000.csv
Running time: 127.098 ms
Sorted data written to merge_sort_50000.csv
[user@unknown C++]$ ./merge_sort.out
Enter CSV file name: dataset_10000.csv
Running time: 50.901 ms
Sorted data written to merge_sort_10000.csv
```

EXPERIMENT STUDY : BINARY SEARCH (C++)

Tester : JORDAN LIM WEI ZHI

	Running time (ms)		
Input size n	Best Time	Average Time	Worst Time
10,000	0.252	0.454	2.497
50,000	0.691	1.858	8.32
100,000	1.376	4.189	17.077
300,000	2.536	10.699	34.653
500,000	4.18	19.991	57.517
800,000	9.901	35.191	96.97
1,000,000	8.365	46.293	120.807
3,000,000	25.102	185.906	396.176
5,000,000	41.912	349.325	689.259
8,000,000	67.075	597.019	1104.365
10,000,000	84.076	805.771	1435.374
30,000,000	251.393	2844.867	4451.879
50,000,000	419.268	5322.988	7744.582
80,000,000	672.457	10136.775	12951.763
100,000,000	841.647	11586.905	16058.287



PROOF OF EXPERIMENT STUDY : BINARY SEARCH (C++)

```
[user@unknown Algo_Assignment]$ cat binary_search_10000.txt
Best case target: 998263956
Best case time: 0.252 ms

Average case time (10% random targets): 0.454 ms

Worst case target: -1
Worst case time: 2.497 ms
[user@unknown Algo_Assignment]$ cat binary_search_50000.txt
Best case target: 992524046
Best case time: 0.691 ms

Average case time (10% random targets): 1.858 ms

Worst case target: -1
Worst case time: 8.320 ms
[user@unknown Algo_Assignment]$ cat binary_search_100000.txt
Best case target: 992856939
Best case time: 1.376 ms

Average case time (10% random targets): 4.189 ms

Worst case target: -1
Worst case time: 17.077 ms
[user@unknown Algo_Assignment]$ cat binary_search_300000.txt
Best case target: 1001558385
Best case time: 2.536 ms

Average case time (10% random targets): 10.699 ms

Worst case target: -1
Worst case time: 34.653 ms
[user@unknown Algo_Assignment]$ cat binary_search_500000.txt
Best case target: 1001128920
Best case time: 4.180 ms

Average case time (10% random targets): 19.991 ms

Worst case target: -1
Worst case time: 57.517 ms
[user@unknown Algo_Assignment]$ cat binary_search_800000.txt
Best case target: 1002267200
Best case time: 9.901 ms

Average case time (10% random targets): 35.191 ms

Worst case target: -1
Worst case time: 96.970 ms
```

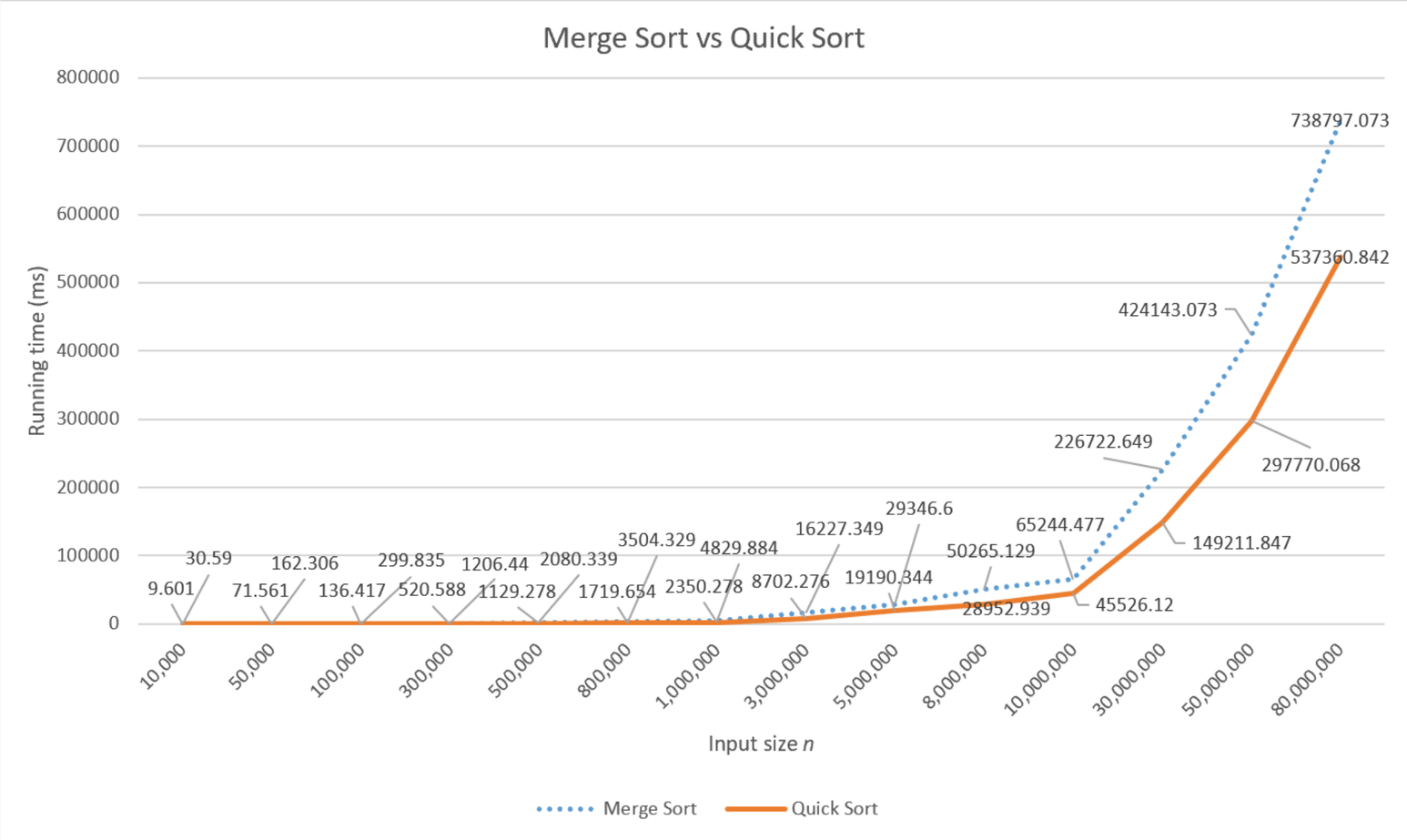
```
[user@unknown Algo_Assignment]$ cat binary_search_1000000.txt
Best case target: 999851588
Best case time: 8.365 ms

Average case time (10% random targets): 46.293 ms

Worst case target: -1
Worst case time: 120.807 ms
[user@unknown Algo_Assignment]$ cat binary_search_3000000.txt
Best case target: 999518623
Best case time: 25.102 ms

Average case time (10% random targets): 185.906 ms

Worst case target: -1
Worst case time: 396.176 ms
[user@unknown Algo_Assignment]$ cat binary_search_5000000.txt
Best case target: 1000270565
Best case time: 41.912 ms

Average case time (10% random targets): 349.325 ms

Worst case target: -1
Worst case time: 689.259 ms
[user@unknown Algo_Assignment]$ cat binary_search_8000000.txt
Best case target: 999304139
Best case time: 67.075 ms

Average case time (10% random targets): 597.019 ms

Worst case target: -1
Worst case time: 1104.365 ms
[user@unknown Algo_Assignment]$ cat binary_search_10000000.txt
Best case target: 999544202
Best case time: 84.076 ms

Average case time (10% random targets): 805.771 ms

Worst case target: -1
Worst case time: 1435.374 ms
[user@unknown Algo_Assignment]$ cat binary_search_30000000.txt
Best case target: 999859108
Best case time: 251.393 ms

Average case time (10% random targets): 2844.867 ms

Worst case target: -1
Worst case time: 4451.879 ms
```

```
[user@unknown Algo_Assignment]$ cat binary_search_50000000.txt
Best case target: 999827423
Best case time: 419.268 ms

Average case time (10% random targets): 5322.988 ms

Worst case target: -1
Worst case time: 7744.582 ms
[user@unknown Algo_Assignment]$ cat binary_search_80000000.txt
Best case target: 1000137663
Best case time: 672.457 ms

Average case time (10% random targets): 10136.775 ms

Worst case target: -1
Worst case time: 12951.763 ms
[user@unknown Algo_Assignment]$ cat binary_search_100000000.txt
Best case target: 1000007845
Best case time: 841.647 ms

Average case time (10% random targets): 11586.905 ms

Worst case target: -1
Worst case time: 16058.287 ms
```

EXPERIMENT STUDY : SORTING ALGORITHMS (PYTHON)

Tester : ERIC TEOH WEI XIANG

	Running time (ms)	
Input size n	Merge Sort	Quick Sort
10,000	30.59	9.601
50,000	162.306	71.561
100,000	299.835	136.417
300,000	1206.44	520.588
500,000	2080.339	1129.278
800,000	3504.329	1719.654
1,000,000	4829.884	2350.278
3,000,000	16227.349	8702.276
5,000,000	29346.6	19190.344
8,000,000	50265.129	28952.939
10,000,000	65244.477	45526.12
30,000,000	226722.649	149211.847
50,000,000	424143.073	297770.068
80,000,000	738797.073	537360.842



PROOF OF EXPERIMENT STUDY : SORTING ALGORITHMS (PYTHON)

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Use
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_10000.csv
Running time: 30.590 ms
Sorted data written to merge_sort_10000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_10000.csv
Running time: 30.590 ms
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Use
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_50000.csv
Running time: 162.306 ms
Sorted data written to merge_sort_50000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_100000.csv
Running time: 299.835 ms
Sorted data written to merge_sort_100000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_300000.csv
Running time: 1206.440 ms
Sorted data written to merge_sort_300000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_500000.csv
Running time: 2080.339 ms
Sorted data written to merge_sort_500000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_800000.csv
Running time: 3504.329 ms
Sorted data written to merge_sort_800000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_1000000.csv
Running time: 4829.884 ms
Sorted data written to merge_sort_1000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_3000000.csv
Running time: 16227.349 ms
Sorted data written to merge_sort_3000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_5000000.csv
Running time: 29346.600 ms
Sorted data written to merge_sort_5000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Use
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_8000000.csv
Running time: 50265.129 ms
Sorted data written to merge_sort_8000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_10000000.csv
Running time: 65244.477 ms
Sorted data written to merge_sort_10000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Use
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_30000000.csv
Running time: 226722.649 ms
Sorted data written to merge_sort_30000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/merge_sort.py"
Enter CSV file name : dataset_50000000.csv
Running time: 424143.012 ms
Sorted data written to merge_sort_50000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_10000.csv
Running time: 9.601 ms
Sorted data written to quick_sort_10000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_50000.csv
Running time: 71.561 ms
Sorted data written to quick_sort_50000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/U
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_100000.csv
Running time: 136.417 ms
Sorted data written to quick_sort_100000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Use
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_300000.csv
Running time: 520.588 ms
Sorted data written to quick_sort_300000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/U
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_500000.csv
Running time: 1129.278 ms
Sorted data written to quick_sort_500000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_800000.csv
Running time: 1719.654 ms
Sorted data written to quick_sort_800000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/U
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_1000000.csv
Running time: 2350.278 ms
Sorted data written to quick_sort_1000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_3000000.csv
Running time: 8702.276 ms
Sorted data written to quick_sort_3000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Use
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_5000000.csv
Running time: 19190.344 ms
Sorted data written to quick_sort_5000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_8000000.csv
Running time: 28952.939 ms
Sorted data written to quick_sort_8000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Use
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_10000000.csv
Running time: 45526.120 ms
Sorted data written to quick_sort_10000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Use
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_30000000.csv
Running time: 149211.847 ms
Sorted data written to quick_sort_30000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

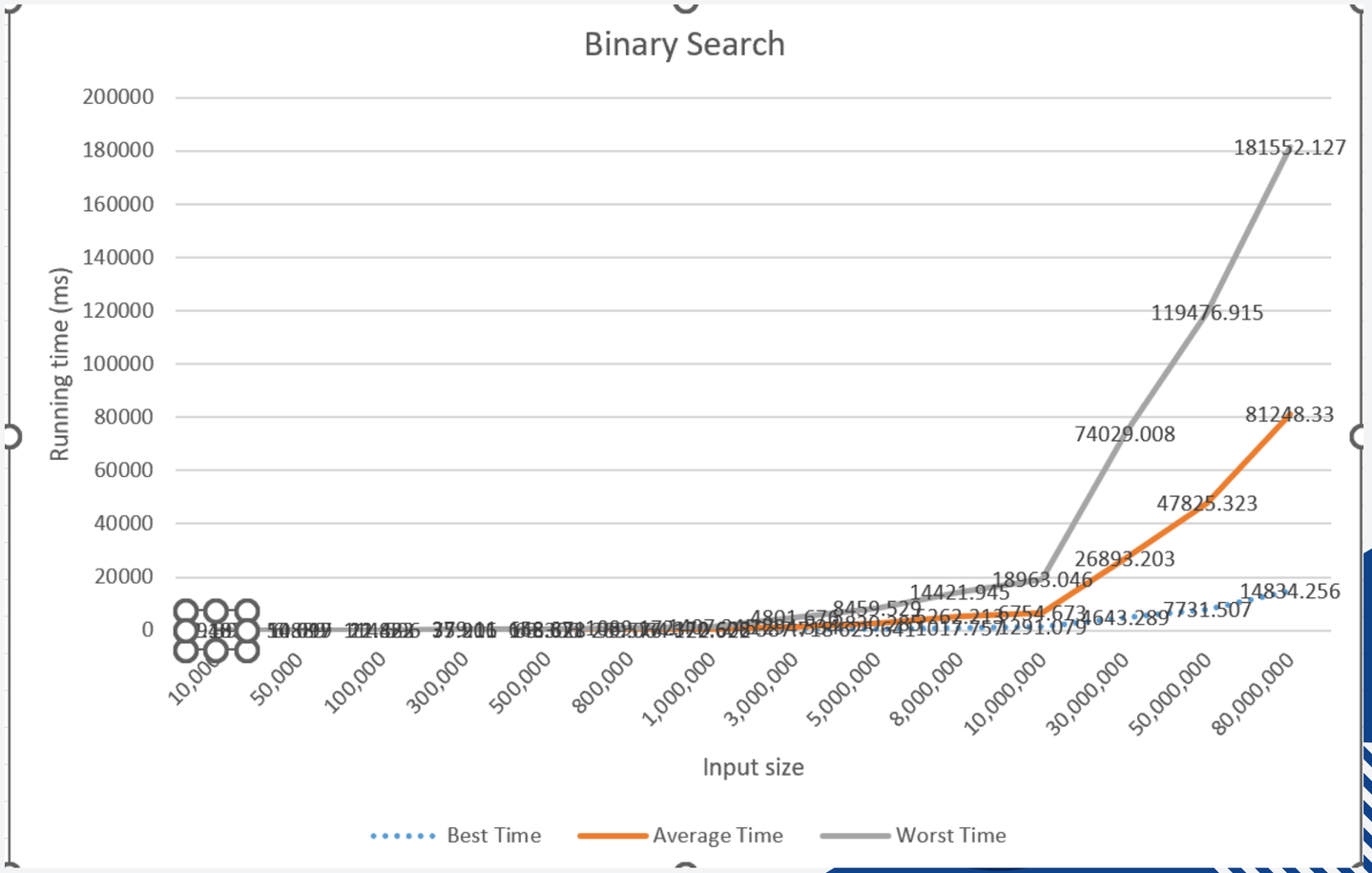
```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/Us
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_50000000.csv
Running time: 297770.068 ms
Sorted data written to quick_sort_50000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

```
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python> & C:/U
MU/Y2S2/Algorithm Design And Analysis/Assignment 2 language/Python/quick_sort.py"
Enter CSV file name : dataset_80000000.csv
Running time: 537360.842 ms
Sorted data written to quick_sort_80000000.csv
PS D:\MMU\Y2S2\Algorithm Design And Analysis\Assignment 2 language\Python>
```

EXPERIMENT STUDY : BINARY SEARCH (PYTHON)

Tester : ERIC TEOH WEI XIANG

	Running time (ms)		
Input size	Best Time	Average Time	Worst Time
10,000	1.487	2.107	9.39
50,000	6.889	10.617	54.099
100,000	11.482	22.823	114.596
300,000	35.206	77.911	379.11
500,000	63.306	148.628	658.871
800,000	99.576	269.064	1089.172
1,000,000	122.622	370.608	1407.245
3,000,000	387.718	1294.534	4801.676
5,000,000	625.641	2830.285	8459.529
8,000,000	1017.757	5262.213	14421.945
10,000,000	1291.079	6754.673	18963.046
30,000,000	4643.289	26893.203	74029.008
50,000,000	7731.507	47825.323	119476.915
80,000,000	14834.256	81248.33	181552.127



PROOF OF EXPERIMENT STUDY : BINARY SEARCH (PYTHON)

binary_search_10000.txt

```
Best case target: 998263956
Best case time: 1.487 ms

Average case time (10% random targets): 2.107 ms

Worst case target: -1
Worst case time: 9.390 ms
```

binary_search_50000.txt

```
Best case target: 992524046
Best case time: 6.889 ms

Average case time (10% random targets): 10.617 ms

Worst case target: -1
Worst case time: 54.099 ms
```

binary_search_100000.txt

```
Best case target: 992856939
Best case time: 11.482 ms

Average case time (10% random targets): 22.823 ms

Worst case target: -1
Worst case time: 114.596 ms
```

binary_search_300000.txt

```
Best case target: 1001558385
Best case time: 35.206 ms

Average case time (10% random targets): 77.911 ms

Worst case target: -1
Worst case time: 379.110 ms
```

binary_search_500000.txt

```
Best case target: 1001128920
Best case time: 63.306 ms

Average case time (10% random targets): 148.628 ms

Worst case target: -1
Worst case time: 658.871 ms
```

binary_search_800000.txt

```
Best case target: 1002267200
Best case time: 99.576 ms

Average case time (10% random targets): 269.064 ms

Worst case target: -1
Worst case time: 1089.172 ms
```

binary_search_1000000.txt

```
Best case target: 999851588
Best case time: 122.662 ms

Average case time (10% random targets): 370.608 ms

Worst case target: -1
Worst case time: 1407.245 ms
```

binary_search_3000000.txt

```
Best case target: 999518623
Best case time: 387.718 ms

Average case time (10% random targets): 1294.534 ms

Worst case target: -1
Worst case time: 4801.676 ms
```

binary_search_5000000.txt

```
Best case target: 1000270565
Best case time: 625.641 ms

Average case time (10% random targets): 2830.285 ms

Worst case target: -1
Worst case time: 8459.529 ms
```

binary_search_8000000.txt

```
Best case target: 999304139
Best case time: 1017.757 ms

Average case time (10% random targets): 5262.213 ms

Worst case target: -1
Worst case time: 14421.945 ms
```

binary_search_10000000.txt

```
Best case target: 999544202
Best case time: 1291.079 ms

Average case time (10% random targets): 6754.673 ms

Worst case target: -1
Worst case time: 18963.046 ms
```

binary_search_30000000.txt

```
Best case target: 999859108
Best case time: 4643.289 ms

Average case time (10% random targets): 26893.203 ms

Worst case target: -1
Worst case time: 74029.008 ms
```

binary_search_50000000.txt

```
Best case target: 999827423
Best case time: 7731.507 ms

Average case time (10% random targets): 47825.323 ms

Worst case target: -1
Worst case time: 119476.915 ms
```

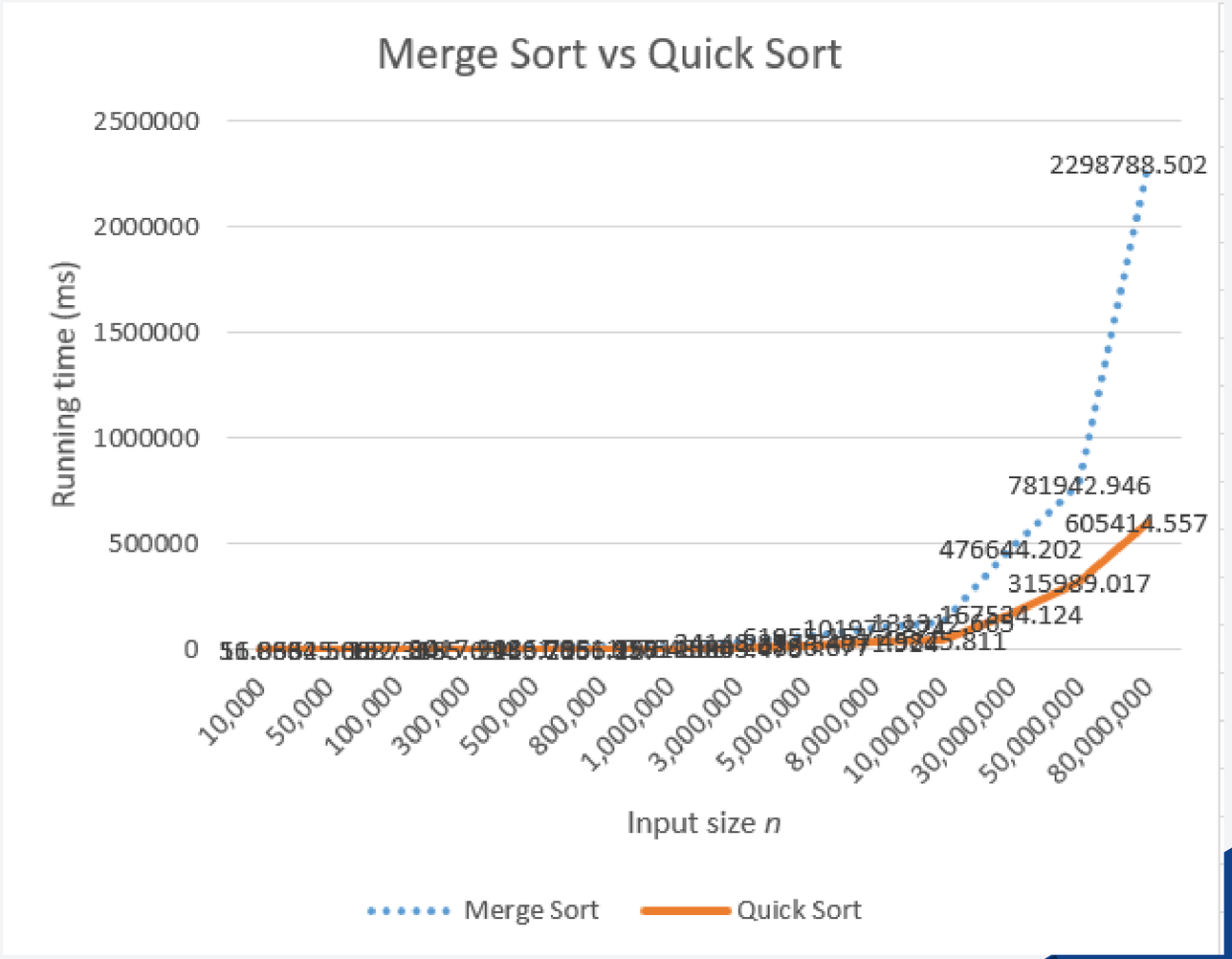
binary_search_80000000.txt

```
Best case target: 1000137663
Best case time: 14834.256 ms

Average case time (10% random targets): 81248.330 ms

Worst case target: -1
Worst case time: 181552.127 ms
```

EXPERIMENT STUDY : SORTING ALGORITHMS (PYTHON)

Tester : CHIN ZHEN HO

	Running time (ms)	
Input size n	Merge Sort	Quick Sort
10,000	51.036	16.863
50,000	325.06	64.569
100,000	887.84	152.318
300,000	3017.903	655.654
500,000	4861.8	1113.706
800,000	7951.959	2156.117
1,000,000	10764.504	2981.806
3,000,000	34148.897	10803.476
5,000,000	61955.151	20896.677
8,000,000	101973.874	34071.924
10,000,000	131312.665	45825.811
30,000,000	476644.202	167534.124
50,000,000	781942.946	315989.017
80,000,000	2298788.502	605414.557



PROOF OF EXPERIMENT STUDY : SORTING ALGORITHMS (PYTHON)

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_10000.csv
Running time: 51.036 ms
Sorted data written to merge_sort_10000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_50000.csv
Running time: 325.060 ms
Sorted data written to merge_sort_50000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_100000.csv
Running time: 887.840 ms
Sorted data written to merge_sort_100000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_300000.csv
Running time: 3017.903 ms
Sorted data written to merge_sort_300000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_500000.csv
Running time: 4861.800 ms
Sorted data written to merge_sort_500000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_800000.csv
Running time: 7951.959 ms
Sorted data written to merge_sort_800000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_1000000.csv
Running time: 10764.504 ms
Sorted data written to merge_sort_1000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_3000000.csv
Running time: 34148.897 ms
Sorted data written to merge_sort_3000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_5000000.csv
Running time: 61955.151 ms
Sorted data written to merge_sort_5000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_8000000.csv
Running time: 101973.874 ms
Sorted data written to merge_sort_8000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_10000000.csv
Running time: 131312.665 ms
Sorted data written to merge_sort_10000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_30000000.csv
Running time: 476644.202 ms
Sorted data written to merge_sort_30000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_50000000.csv
Running time: 781942.946 ms
Sorted data written to merge_sort_50000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\merge_sort.py"
Enter CSV file name : dataset_80000000.csv
Running time: 2298788.502 ms
Sorted data written to merge_sort_80000000.csv
PS C:\Python>
```

```
PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_10000.csv
Running time: 16.863 ms
Sorted data written to quick_sort_10000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_50000.csv
Running time: 64.569 ms
Sorted data written to quick_sort_50000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_100000.csv
Running time: 152.318 ms
Sorted data written to quick_sort_100000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_300000.csv
Running time: 655.654 ms
Sorted data written to quick_sort_300000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_500000.csv
Running time: 1113.706 ms
Sorted data written to quick_sort_500000.csv
PS C:\Python>
```

```
PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_800000.csv
Running time: 2156.117 ms
Sorted data written to quick_sort_800000.csv
PS C:\Python>
```

```
PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_1000000.csv
Running time: 2981.806 ms
Sorted data written to quick_sort_1000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_3000000.csv
Running time: 10803.476 ms
Sorted data written to quick_sort_3000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_5000000.csv
Running time: 20896.677 ms
Sorted data written to quick_sort_5000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_8000000.csv
Running time: 34071.924 ms
Sorted data written to quick_sort_8000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_10000000.csv
Running time: 45825.811 ms
Sorted data written to quick_sort_10000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_30000000.csv
Running time: 167534.124 ms
Sorted data written to quick_sort_30000000.csv
PS C:\Python>
```

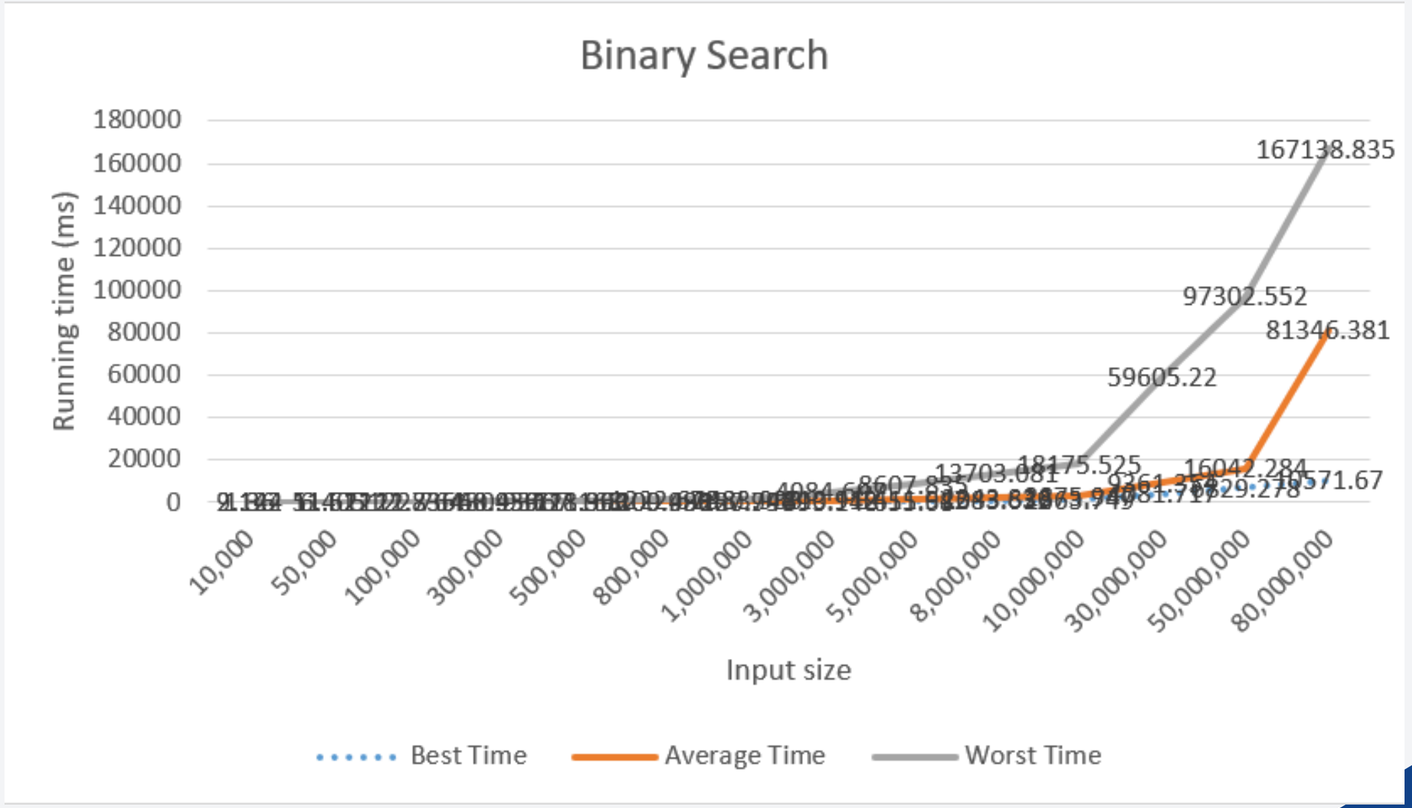
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_50000000.csv
Running time: 315989.017 ms
Sorted data written to quick_sort_50000000.csv
PS C:\Python>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Python> python -u "c:\Python\quick_sort.py"
Enter CSV file name : dataset_80000000.csv
Running time: 605414.557 ms
Sorted data written to quick_sort_80000000.csv
```

EXPERIMENT STUDY : BINARY SEARCH(PYTHON)

Tester : CHIN ZHEN HO

	Running time (ms)		
Input size	Best Time	Average Time	Worst Time
10,000	1.36	2.124	9.142
50,000	6.405	11.521	51.612
100,000	12.836	21.735	122.764
300,000	40.936	68.459	430.936
500,000	67.168	118.393	721.963
800,000	109.498	200.986	1222.673
1,000,000	127.744	250.997	1483.933
3,000,000	393.148	810.912	4984.697
5,000,000	655.68	1411.942	8607.835
8,000,000	1083.632	2343.836	13703.081
10,000,000	1363.749	2875.947	18175.525
30,000,000	4081.717	9361.354	59605.22
50,000,000	6829.278	16042.284	97302.552
80,000,000	10571.67	81346.381	167138.835



PROOF OF EXPERIMENT STUDY : BINARY SEARCH (PYTHON)

binary_search_10000.txt

```
Best case target: 998263956
Best case time: 1.363 ms

Average case time (10% random targets): 2.124 ms

Worst case target: -1
Worst case time: 9.142 ms
```

binary_search_50000.txt

```
Best case target: 992524046
Best case time: 6.405 ms

Average case time (10% random targets): 11.521 ms

Worst case target: -1
Worst case time: 51.612 ms
```

binary_search_100000.txt

```
Best case target: 49054043
Best case time: 12.836 ms

Average case time (10% random targets): 21.735 ms

Worst case target: -1
Worst case time: 122.764 ms
```

binary_search_300000.txt

```
Best case target: 1627308013
Best case time: 40.936 ms

Average case time (10% random targets): 68.459 ms

Worst case target: -1
Worst case time: 430.936 ms
```

binary_search_500000.txt

```
Best case target: 1305421145
Best case time: 67.168 ms

Average case time (10% random targets): 118.393 ms

Worst case target: -1
Worst case time: 721.963 ms
```

binary_search_800000.txt

```
Best case target: 1640848180
Best case time: 109.498 ms

Average case time (10% random targets): 200.986 ms

Worst case target: -1
Worst case time: 1222.673 ms
```

binary_search_1000000.txt

```
Best case target: 27558717
Best case time: 127.744 ms

Average case time (10% random targets): 250.997 ms

Worst case target: -1
Worst case time: 1483.933 ms
```

binary_search_3000000.txt

```
Best case target: 819590
Best case time: 393.148 ms

Average case time (10% random targets): 810.912 ms

Worst case target: -1
Worst case time: 4984.697 ms
```

binary_search_5000000.txt

```
Best case target: 454652959
Best case time: 655.680 ms

Average case time (10% random targets): 1411.942 ms

Worst case target: -1
Worst case time: 8607.835 ms
```

binary_search_8000000.txt

```
Best case target: 1607495810
Best case time: 1083.632 ms

Average case time (10% random targets): 2343.836 ms

Worst case target: -1
Worst case time: 13703.081 ms
```

binary_search_10000000.txt

```
Best case target: 1464745858
Best case time: 1363.749 ms

Average case time (10% random targets): 2875.947 ms

Worst case target: -1
Worst case time: 18175.525 ms
```

binary_search_30000000.txt

```
Best case target: 1439322098
Best case time: 4081.717 ms

Average case time (10% random targets): 9361.354 ms

Worst case target: -1
Worst case time: 59605.220 ms
```

binary_search_50000000.txt

```
Best case target: 1011167965
Best case time: 6829.278 ms

Average case time (10% random targets): 16042.284 ms

Worst case target: -1
Worst case time: 97302.552 ms
```

binary_search_80000000.txt

```
Best case target: 826370703
Best case time: 10571.670 ms

Average case time (10% random targets): 81346.381 ms

Worst case target: -1
Worst case time: 167138.835 ms
```

SCREENSHOT OF DEVICE SPECIFICATIONS

Device specifications

Device name	LAPTOP-S89895HV	
Processor	AMD Ryzen 7 5800H with Radeon Graphics	3.20 GHz
Installed RAM	16.0 GB (13.9 GB usable)	
Device ID	3F427204-FBEE-4E70-8B6C-15483A421914	
Product ID	00327-30000-00000-AAOEM	
System type	64-bit operating system, x64-based processor	
Pen and touch	No pen or touch input is available for this display	

BERNARD RYAN SIM KANG XUAN

Device specifications

Device name	LAPTOP-PCG4MAG7	
Processor	AMD Ryzen 9 5900HX with Radeon Graphics	3.30 GHz
Installed RAM	16.0 GB (15.4 GB usable)	
Device ID	FBD5FC1B-E425-4ECE-A18E-BD885BDE0DCB	
Product ID	00342-42600-10273-AAOEM	
System type	64-bit operating system, x64-based processor	
Pen and touch	No pen or touch input is available for this display	

CHIN ZHEN HO

```
OS: Artix Linux x86_64
Host: 82B5 Lenovo Legion 5 15ARH05
Kernel: 6.14.6-artix1-1
Shell: bash 5.2.37
Resolution: 1920x1080
Terminal: xfce4-terminal
Terminal Font: Monospace 11
CPU: AMD Ryzen 5 4600H with Radeon Graphics (12) @ 3.000GHz
GPU: AMD ATI Radeon Vega Series / Radeon Vega Mobile Series
GPU: NVIDIA GeForce GTX 1650 Mobile / Max-Q
Memory: 6174MiB / 15351MiB
```

JORDAN LIM WEI ZHI

Device name	DESKTOP-O21J09H
Processor	11th Gen Intel(R) Core(TM) i9-11900H @ 2.50GHz (2.50 GHz)
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	289A6FDB-181F-4386-96EE-78A0EDE0DCFB
Product ID	00342-42615-98376-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

ERIC TEOH WEI XIANG

REFERENCES

- VIDELA, L. S. (2020, FEBRUARY 28). QUICK SORT ANIMATION (PIVOT AS LAST ELEMENT) [VIDEO]. YOUTUBE. HTTPS://YOUTU.BE/BIOJCLBDR7Y
- GEEKSFORGEEKS. (2024, NOVEMBER 18). TIME AND SPACE COMPLEXITY ANALYSIS OF QUICK SORT. GEEKSFORGEEKS. HTTPS://WWW.GEEKSFORGEEKS.ORG/DSA/TIME-AND-SPACE-COMPLEXITY-ANALYSIS-OF-QUICK-SORT/

  GEEKSFORGEEKS. (2024B, MARCH 14). TIME AND SPACE COMPLEXITY ANALYSIS OF MERGE SORT. GEEKSFORGEEKS. HTTPS://WWW.GEEKSFORGEEKS.ORG/DSA/TIME-AND-SPACE-COMPLEXITY-ANALYSIS-OF-MERGE-SORT/
- CODECADEMY. (N.D.-D). TIME COMPLEXITY OF MERGE SORT: A DETAILED ANALYSIS. CODECADEMY. HTTPS://WWW.CODECADEMY.COM/ARTICLE/TIME-COMPLEXITY-OF-MERGE-SORT
- GEEKSFORGEEKS. (2025, APRIL 25). MERGE SORT DATA STRUCTURE AND ALGORITHMS TUTORIALS. GEEKSFORGEEKS. HTTPS://WWW.GEEKSFORGEEKS.ORG/DSA/MERGE-SORT/
- GEEKSFORGEEKS. (2025, MAY 12). BINARY SEARCH ALGORITHM ITERATIVE AND RECURSIVE IMPLEMENTATION. GEEKSFORGEEKS. HTTPS://WWW.GEEKSFORGEEKS.ORG/DSA/BINARY-SEARCH/
- GEEKSFORGEEKS. (2024, MARCH 18). TIME AND SPACE COMPLEXITY ANALYSIS OF BINARY SEARCH ALGORITHM. GEEKSFORGEEKS. HTTPS://WWW.GEEKSFORGEEKS.ORG/COMPLEXITY-ANALYSIS-OF-BINARY-SEARCH/
- ABDUL BARI. (2018, JANUARY 29). 2.6.1 BINARY SEARCH ITERATIVE METHOD [VIDEO]. YOUTUBE. HTTPS://WWW.YOUTUBE.COM/WATCH?V=C2APEW9PGTW

DATASET

The link below had provided the basic dataset that generated by dataset generator, quick sort dataset and also merge sort dataset

https://drive.google.com/drive/folders/1R1DJuG79ly-IezHu9He4fmj10uXr_2eG?usp=sharing

CONCLUSION

- Use arrays with Merge Sort for stable O(n log n) sorting.
- Use Binary Search for fast O(log n) lookups in sorted arrays.
- Arrays offer fast index-based access, ideal for AVL-style ordered data.
- Merge Sort gives reliable performance unlike Quick Sort’s worst-case $O(n^2)$.
- Linked lists are better for frequent insert/remove but not for fast searching or sorting.

Final Takeaways:

- Array + Merge Sort + Binary Search is best for AVL-inspired implementations.
- Merge Sort is stable and reliable.
- Binary Search provides the fastest lookups on sorted data.

QNA SESSION

THANK YOU