



ALGORITHM ANALYSIS AND DESIGN

FINAL PROJECT

SUPERVISED BY :
DR. AZAA A. ALI





Table of Contant:


01 Introduction

04 Discussion

02 Implementation

05 Team Members

03 Problem Solving and
Analysis





INTRODUCTION

THIS PROJECT AIMS ON IDENTIFYING AND ANALYZING THE PERFORMANCE OF DIFFERENT SORTING ALGORITHMS (MERGE, SELECTION AND QUICK SORT) IN DIFFERENT SORTING CASES, INCREASING, DECREASING AND RANDOM GENERATION OF NUMBERS.

EACH OF THESE CASES WILL BE CALCULATED USING BOTH, THEORETICAL AND EXPERIMENTAL TIME AND THEN ALL OF THE CASES WILL BE CONDUCTED.

DEVICE CHARACTERISTICS

Random Access Memory (RAM)	Central processor unit (CPU)	Operating System (OS)	System Type
8 GB	Core i7	Windows OS	64-bit

INPUT SELECTION

INPUT SIZES :

*(100, 500, 1000, 5000, 10000, 50000,
100000)*

INPUT ORDER:

- *INCREASING*
- *DECREASING*
- *RANDOM*



IMPLEMENTATION

SELECTION SORT ALGORITHM

A SIMPLE SORTING ALGORITHM. THIS SORTING ALGORITHM IS AN IN-PLACE COMPARISON-BASED ALGORITHM IN WHICH THE LIST IS DIVIDED INTO TWO PARTS, THE SORTED PART AT THE LEFT END AND THE UNSORTED PART AT THE RIGHT END. INITIALLY, THE SORTED PART IS EMPTY, AND THE UNSORTED PART IS THE ENTIRE LIST.

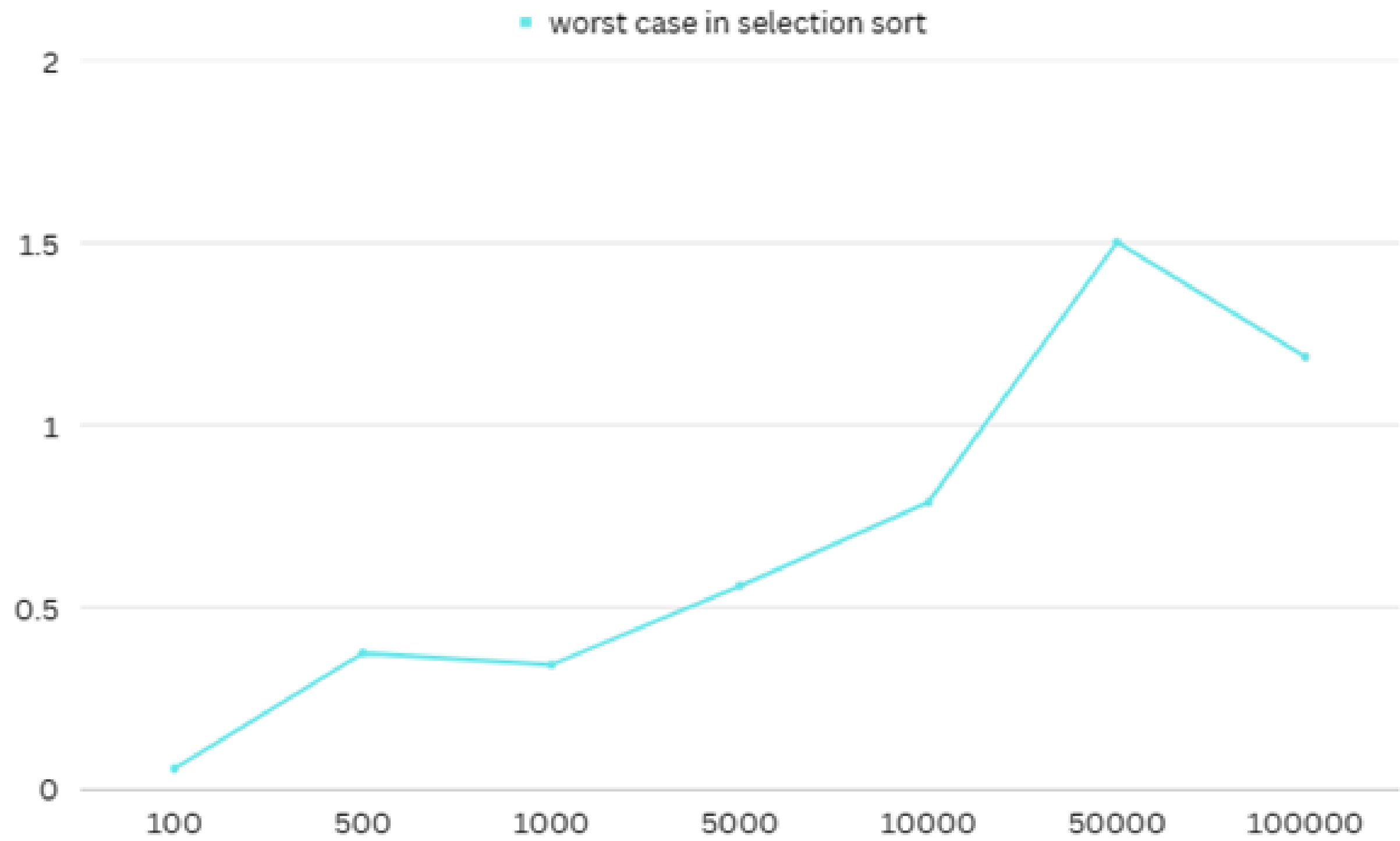
	Best Case	Average Case	Worst Case
Data Order	Data sorted increasingly	Randomly	Data sorted decreasingly
Time Complexity	$T(N) = O(n^2)$	$T(N) = O(n^2)$	$T(N) = O(n^2)$

Cases	Best case		Average case		Worst case	
Input size	Theoretic al Time	Practical Time	Theoretic al Time	Practical Time	Theoretic al Time	Practical Time
100	10000	0.3678	10000	0.3776	10000	0.5835
500	250000	2.3404	250000	11.1503	250000	2.8195
1000	1000000	106.8826	1000000	16.2546	1000000	34.3706
5000	25000000	66.5106	25000000	60.4879	25000000	55.9923
10000	10000000 0	90.8322	10000000 0	287.0152	10000000 0	118.441
50000	25000000 00	563.0444 8	25000000 00	1785.5635 2	25000000 00	1878.7329 3
100000	10000000 000	1921.4448 6	10000000 000	6491.2481	10000000 000	5942.009 9

Cases	Best case	Average case	Worst case
Input size (n)	$P/T \times 100$	$P/T \times 100$	$P/T \times 100$
100	1.226	0.03776	0.05835
500	2.3404	1.48670667	0.375933333 3
1000	0.01068826	0.54182	0.343707
5000	0.664106	0.604879	0.559923
10000	1.816644	0.22961216	0.789606666 7
50000	2.252177922.2 5217792	1.428450816	1.502986344
100000	0.384288972	1.29824962	1.18840198







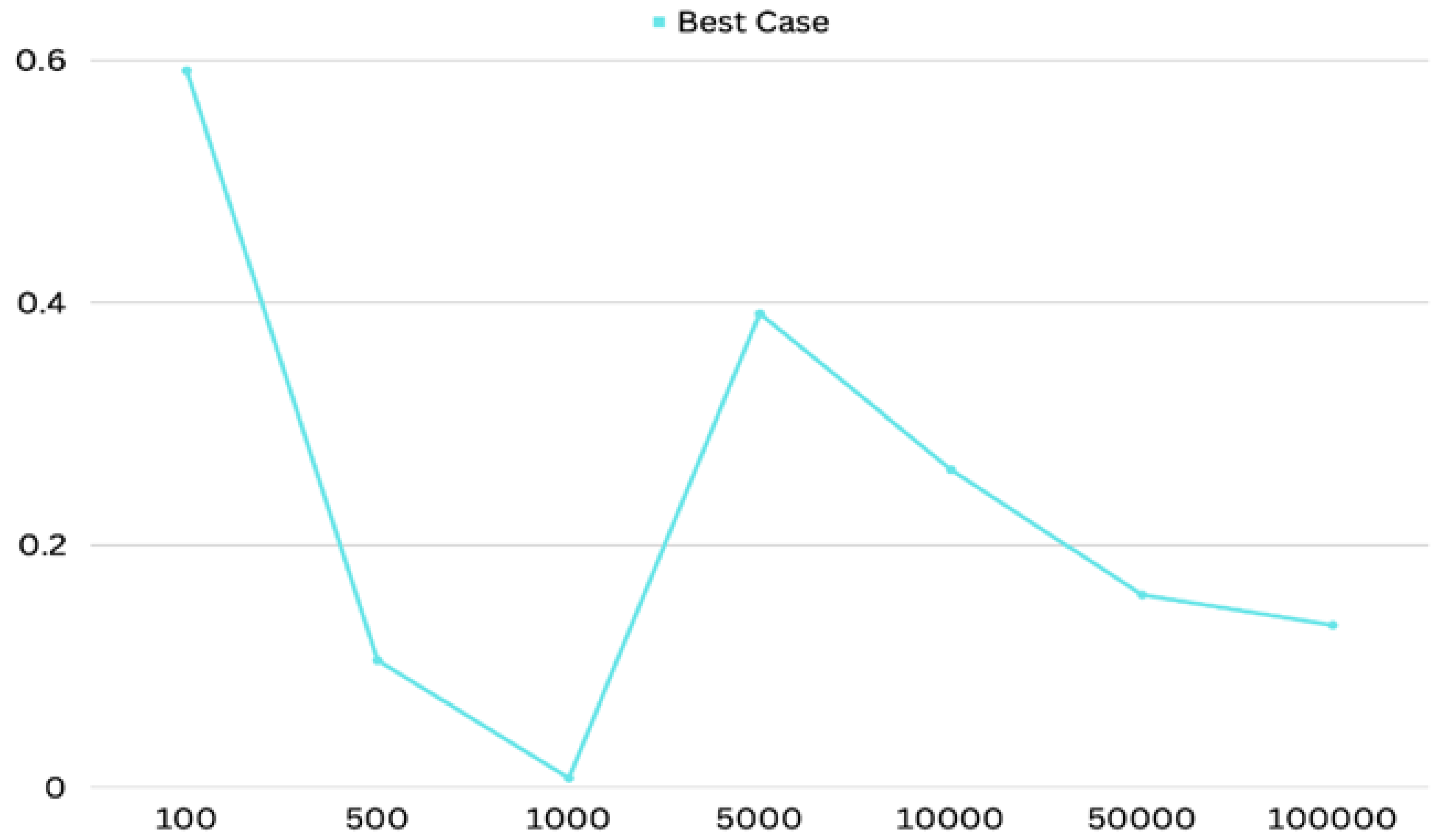
MERGE SORT ALGORITHM

MERGE SORT IS A SORTING ALGORITHM THAT WORKS BY DIVIDING AN ARRAY INTO SMALLER SUBARRAYS, SORTING EACH SUBARRAY, AND THEN MERGING THE SORTED SUBARRAYS BACK TOGETHER TO FORM THE FINAL SORTED ARRAY.

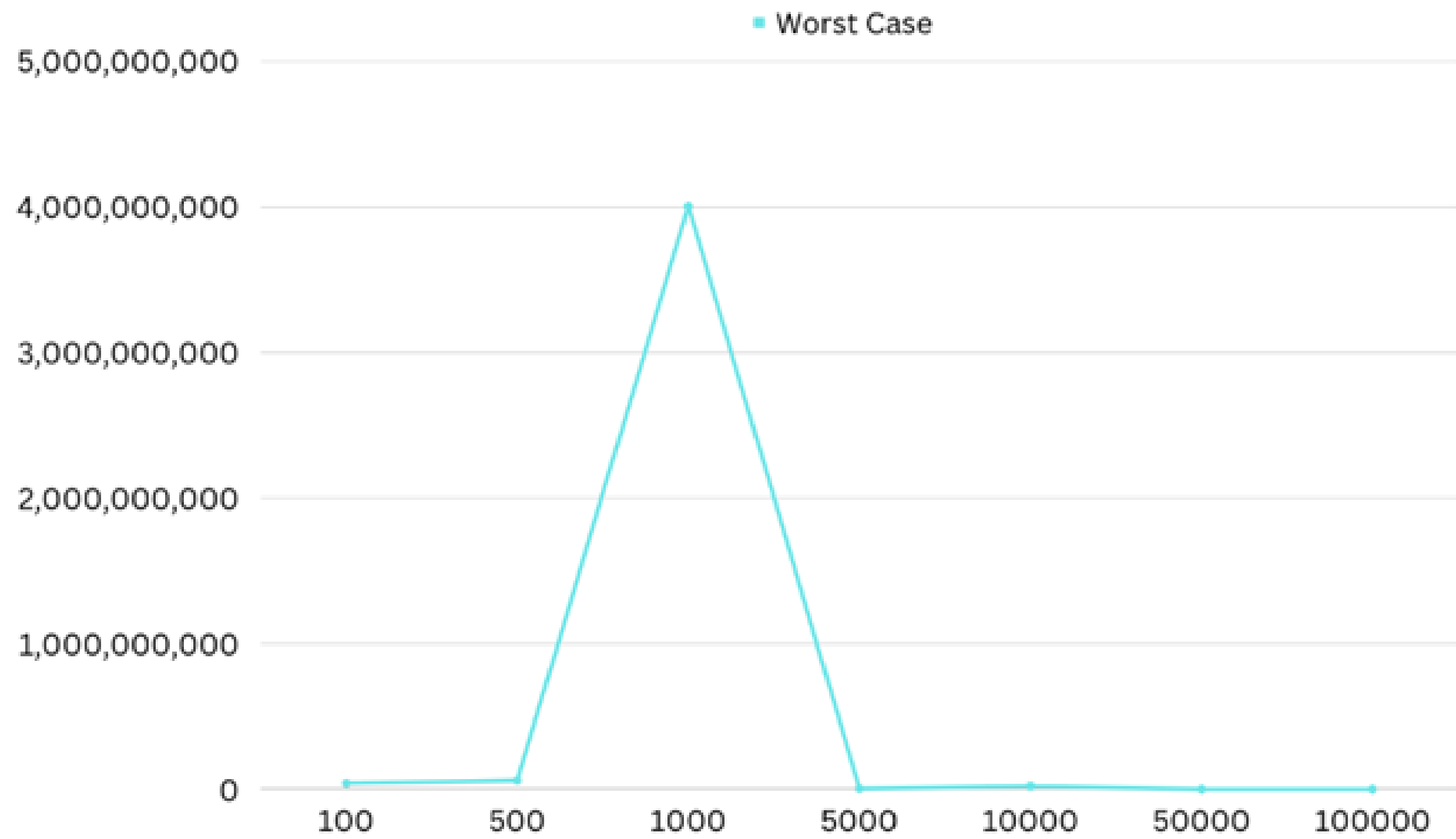
	Best Case	Average Case	Worst Case
Data Order	Data sorted increasingly	Randomly	Data sorted decreasingly
Time Complexity	$T(N) = \Theta(n \log n)$	$T(N) = \Theta(n \log n)$	$T(N) = \Theta(n \log n)$

Cases	Best case		Average case		Worst case	
Input size	Theoretic al Time	Practical Time	Theoretic al Time	Practical Time	Theoretic al Time	Practical Time
100	200	1.1832	200	0.6874	200	812234960
500	1349.485002	1.4155	1349.485002	2.1782	1349.485002	812262940
1000	3000	23.7872	3000	20.2924	3000	812295690
5000	18494.85002	72.284704	18494.85002	48.1811	18494.85002	812329980
10000	40000	104.9732	40000	101.884704	40000	812354340
50000	234948.5002	373.298912	234948.5002	315.849696	234948.5002	812376020
100000	500000	669.42208	500000	667.8953	500000	812402260

Cases	Best case	Average case	Worst case
Input size (n)	$P/T \times 100$	$P/T \times 100$	$P/T \times 100$
100	0.5916	0.3437	40617480
500	0.104891866	0.1614097227	60190586.69
1000	2.643022222	0.6764133333	4002955244
5000	0.39083693	0.2605108987	4392195.552
10000	0.262433	0.25471176	20308858.5
50000	0.15888854203	0.1344335868	345767.6977
100000	0.133884416	0.13357906	162480.452







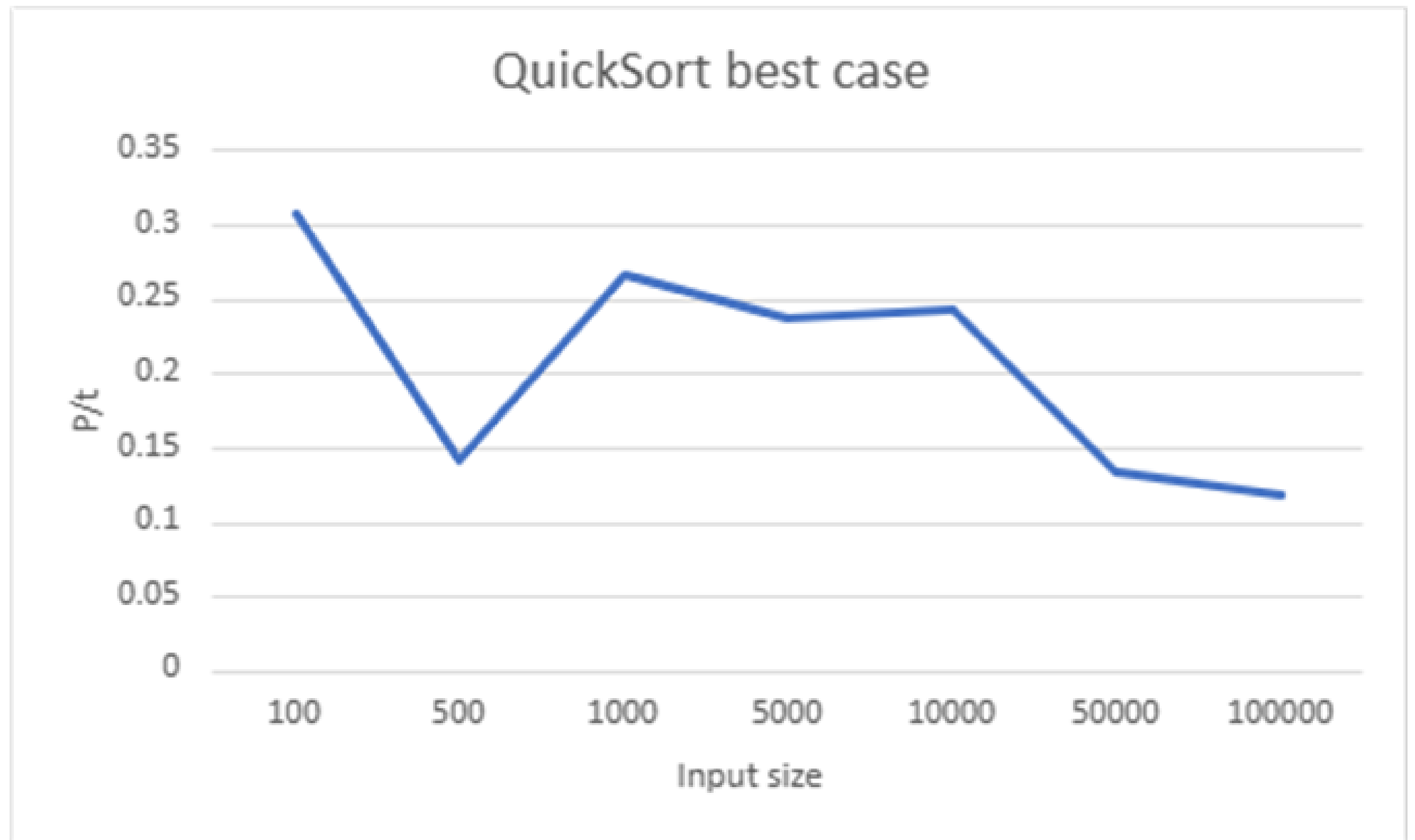
QUICK SORT ALGORITHM

QUICKSORT IS A SORTING ALGORITHM BASED ON THE DIVIDE AND CONQUER APPROACH WHERE AN ARRAY IS DIVIDED INTO SUBARRAYS BY SELECTING A PIVOT ELEMENT (ELEMENT SELECTED FROM THE ARRAY).

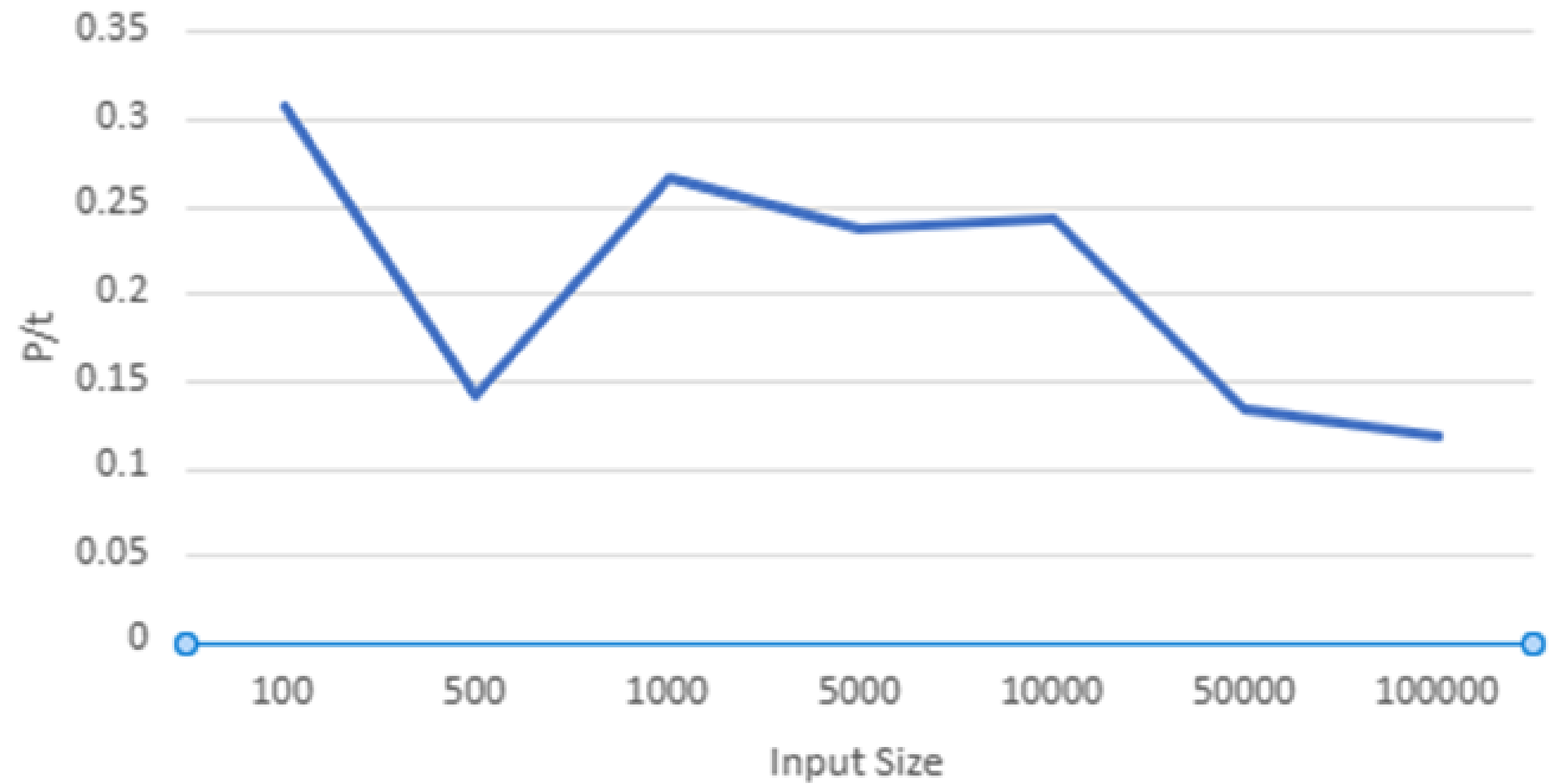
	Best Case	Average Case	Worst Case
Data Order	Data sorted increasingly	Randomly	Data sorted decreasingly
Time Complexity	$T(N) = O(n \log n)$	$T(N) = O(n \log n)$	$T(N) = O(n^2)$

Cases	Best case		Average case		Worst case	
Input size	Theoretic al Time	Practical Time	Theoretic al Time	Practical Time	Theoretic al Time	Practical Time
100	200	0.4655	200	0.6157	10000	80928270
500	1349.485002	1.0675	1349.485002	1.9257	250000	809323170
1000	3000	14.1009	3000	8.0233	1000000	809357730
5000	18494.85002	50.1287	18494.85002	44.0026	25000000	809401020
10000	40000	125.298	40000	97.438096	100000000	809430410
50000	234948.5002	292.411008	234948.5002	313.8768	2500000000	809458060
100000	500000	617.80122	500000	592.97619	100000000000	809498660

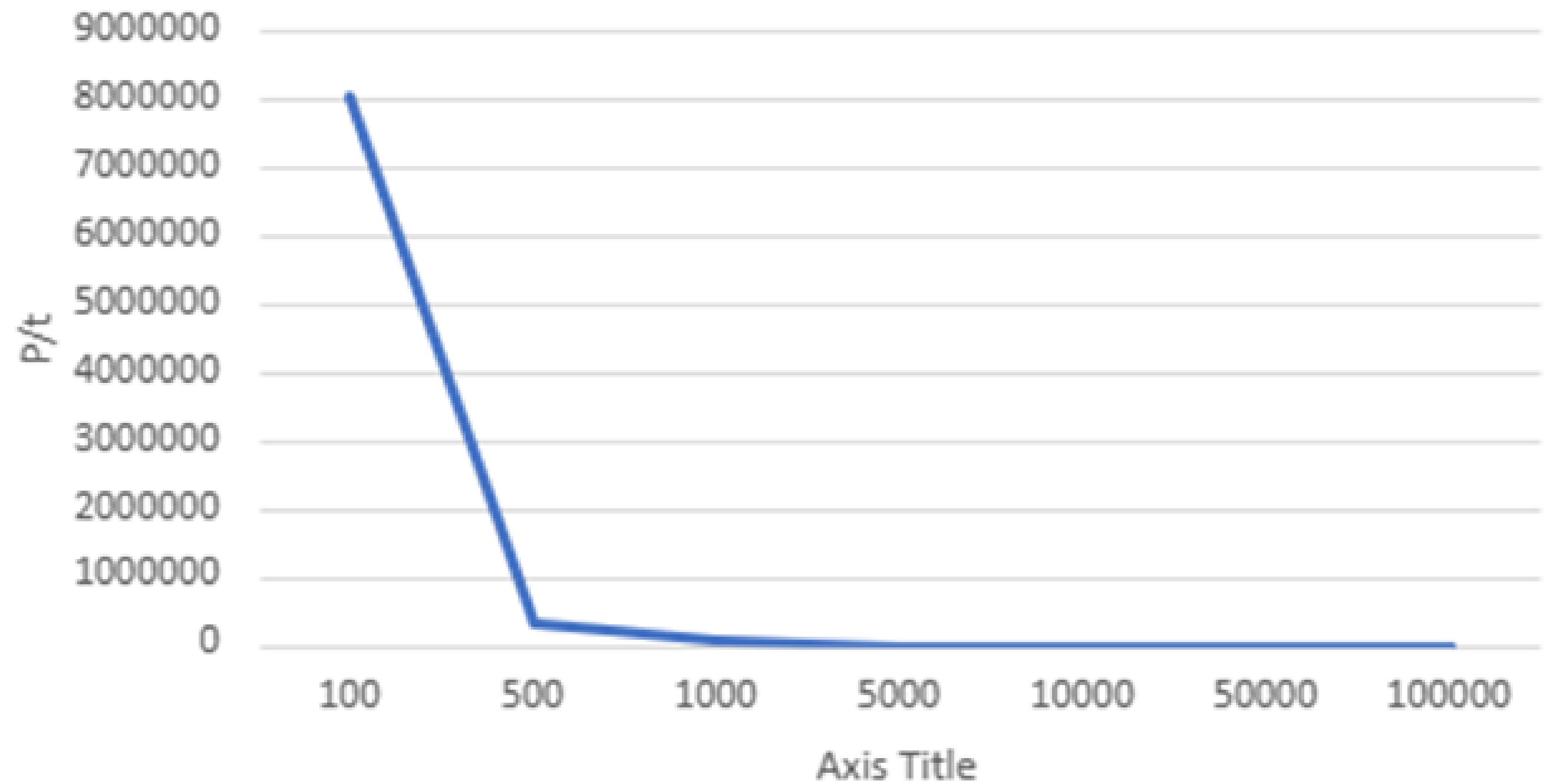
Cases	Best case	Average case	Worst case
Input size (n)	$P/T \times 100$	$P/T \times 100$	$P/T \times 100$
100	0.23275	0.30785	8092827
500	0.07910425076	0.1426988812	323729.268
1000	0.47003	0.267443333	80935.773
5000	0.2710413977	0.2379181229	3237.60408
10000	0.313245	0.24359524	809.43041
50000	0.1244574908	0.1335938726	32.3783224
100000	0.123560244	0.118595238	8.0945806



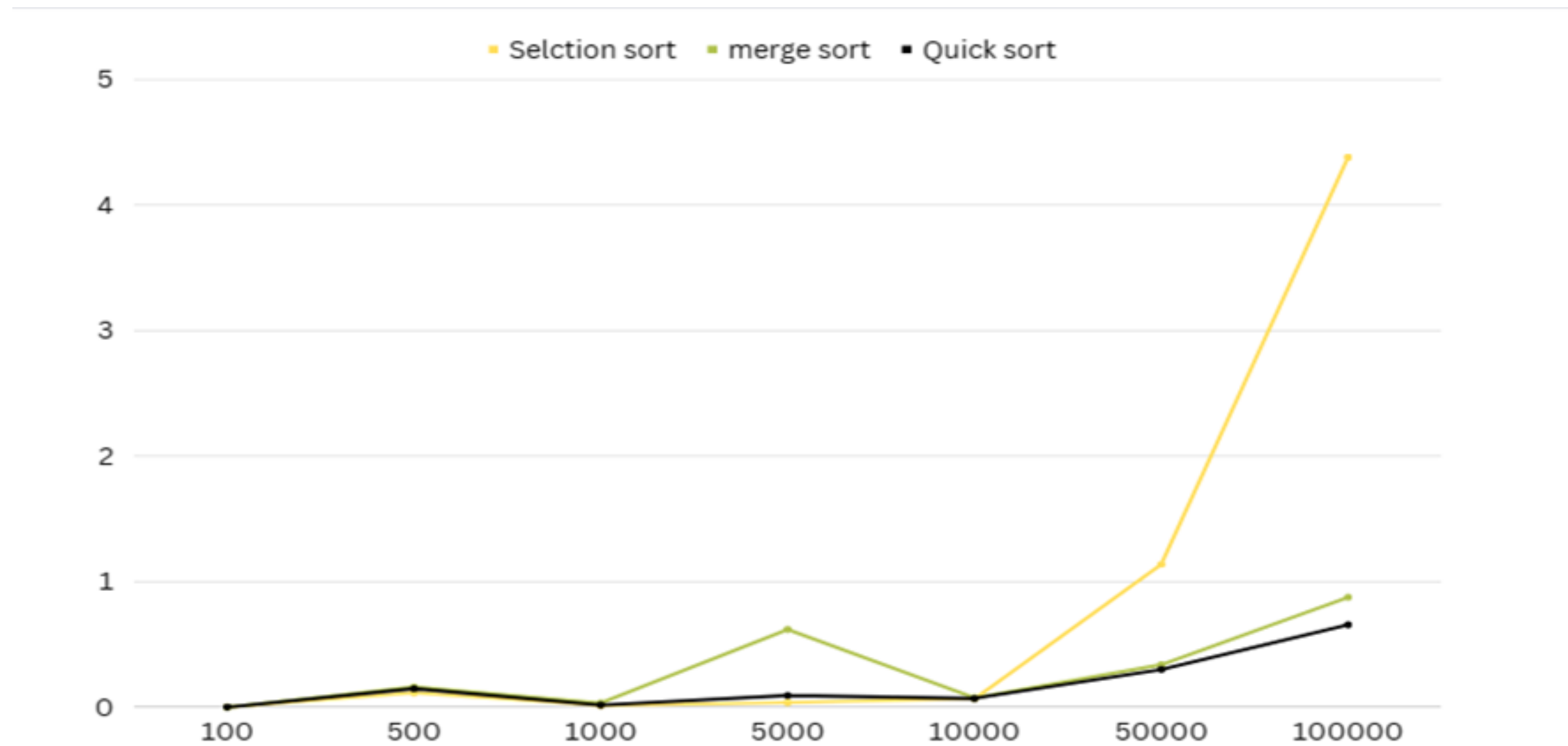
QuickSort Average Case



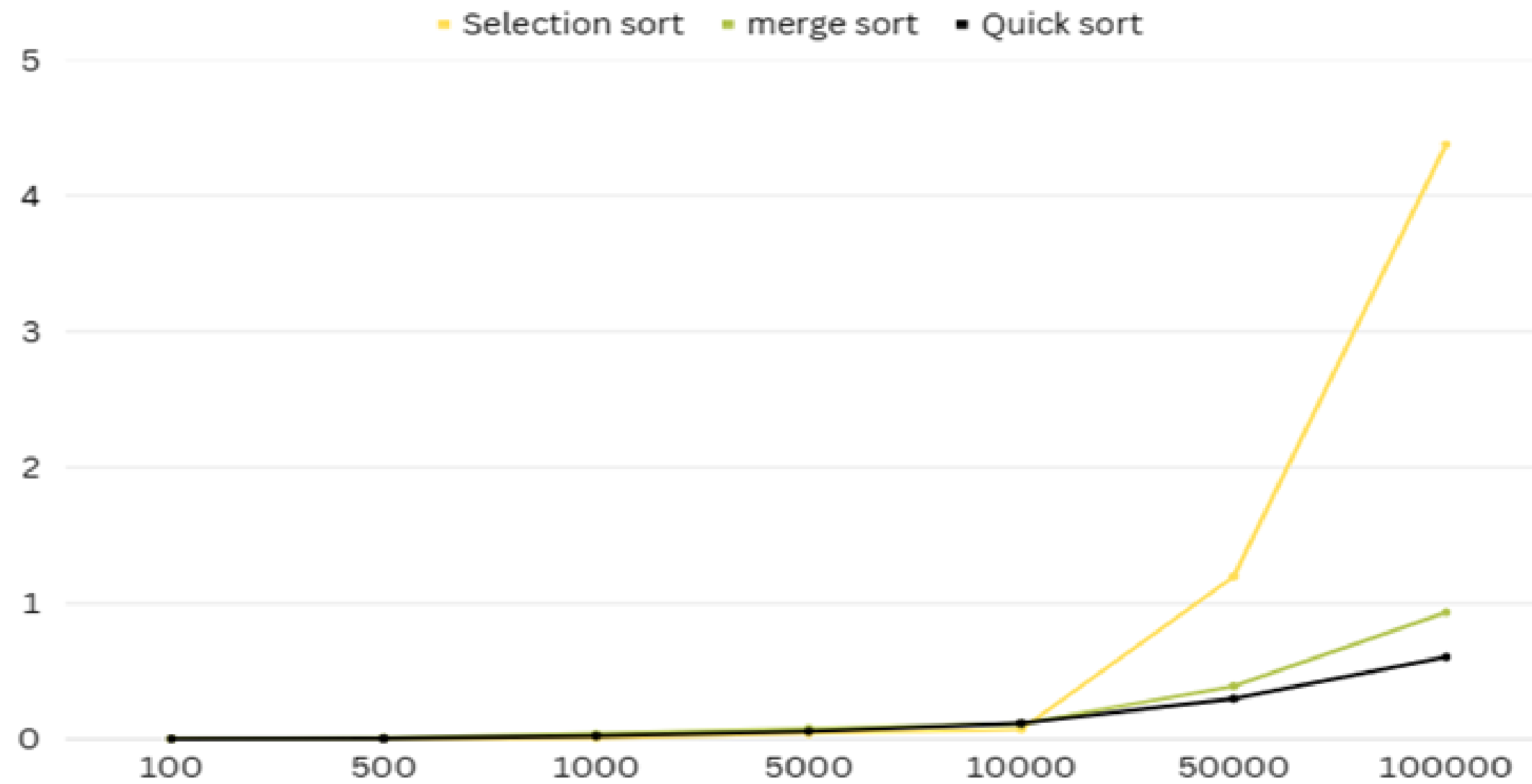
QuickSort Worst Case



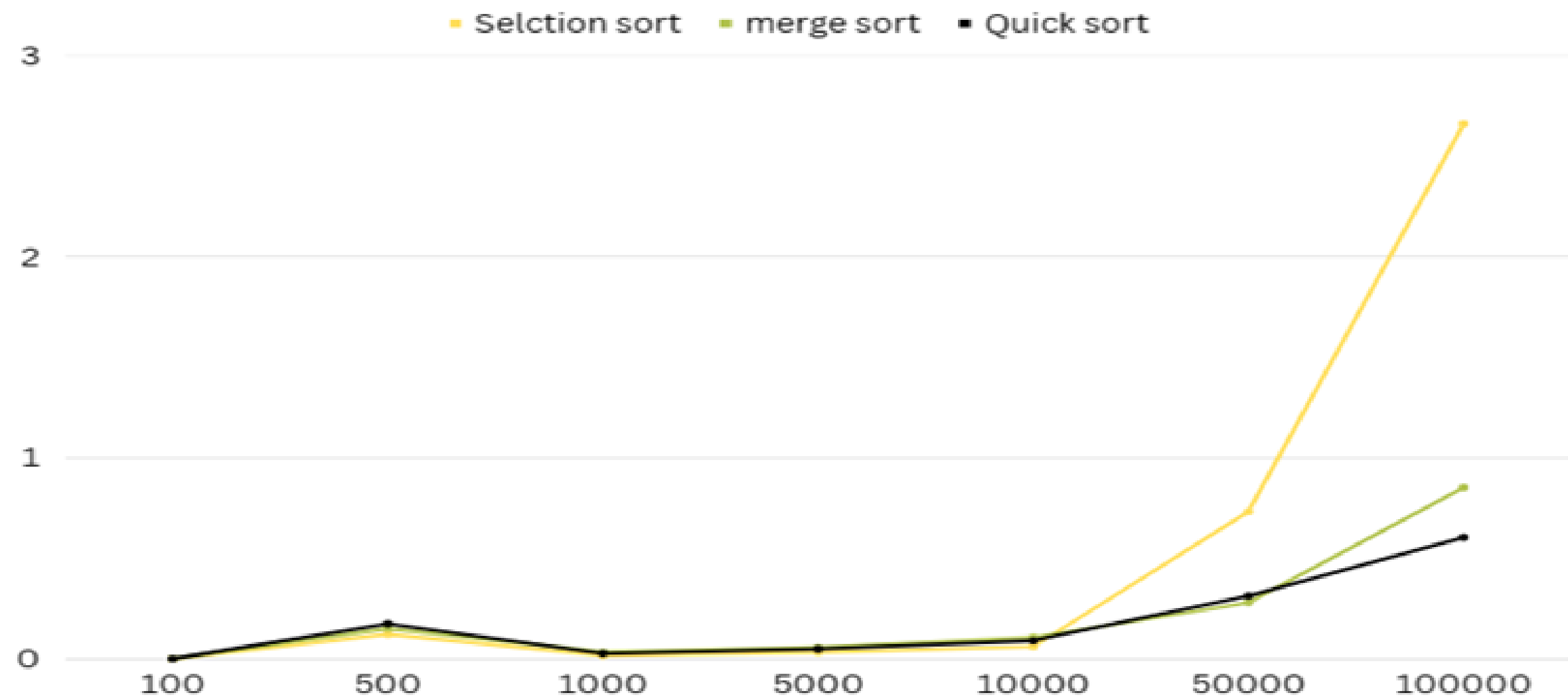
BEST CASE GRAPH



WORST CASE GRAPH



AVERAGE CASE GRAPH





COMPARISON SORT

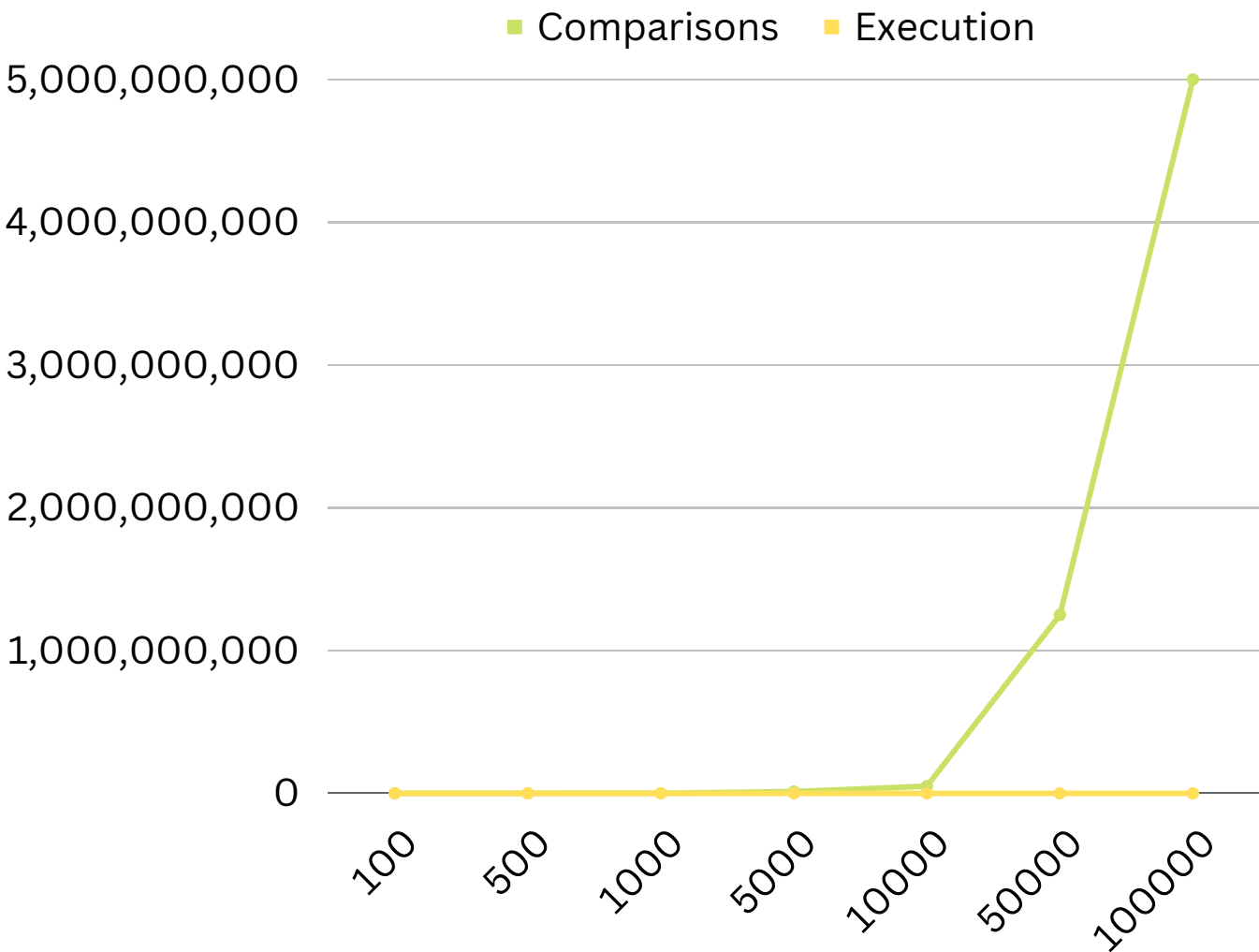
FURMLA FOR COMPUTR EXECUTION TIME

```
long startTime = System.nanoTime();  
long stopTime = System.nanoTime();  
long excutionTime = stopTime - startTime;  
System.out.println("Excution Time In nanosecend:"  
+excutionTime+ "ns");
```

SELECTION SORT ALGORITHM

IS THE NUMBER OF COMPARISONS IN SELECTION SORT REALLY A GOOD PREDICTOR OF THE EXECUTION TIME?

Input Size (n)	Number of comparisons	Execution Time (μs)
100	664.4	2.1636ms
500	4483	5.69ms
1000	9965.8	29.6679ms
5000	61438.6	194.44ms
10000	132877.1	534.72ms
50000	780482	2637.16ms
100000	3521928	5107.87ms

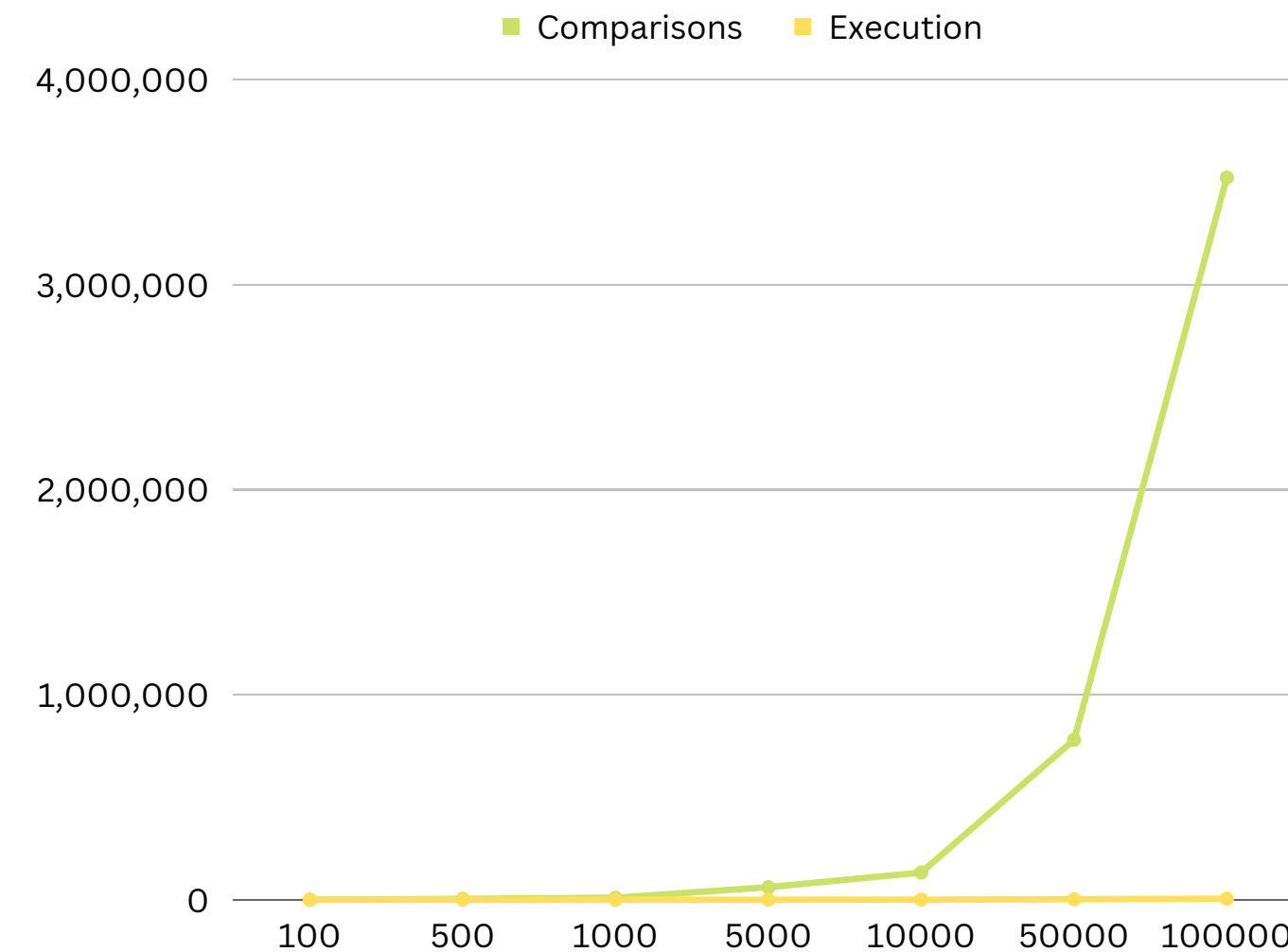


EQUATION ($n^2/2$)

MERGE SORT ALGORITHM

IS THE NUMBER OF COMPARISONS IN MERGE SORT REALLY A GOOD PREDICTOR OF THE EXECUTION TIME?

Input Size (n)	Number of comparisons	Execution Time (μs)
100	5000	329.77ms
500	125000	530 ms
1000	500000	378.46ms
5000	12500000	674.06ms
10000	50000000	1170.88ms
50000	1250000000	7124.7ms
100000	5000000000	17512.28ms

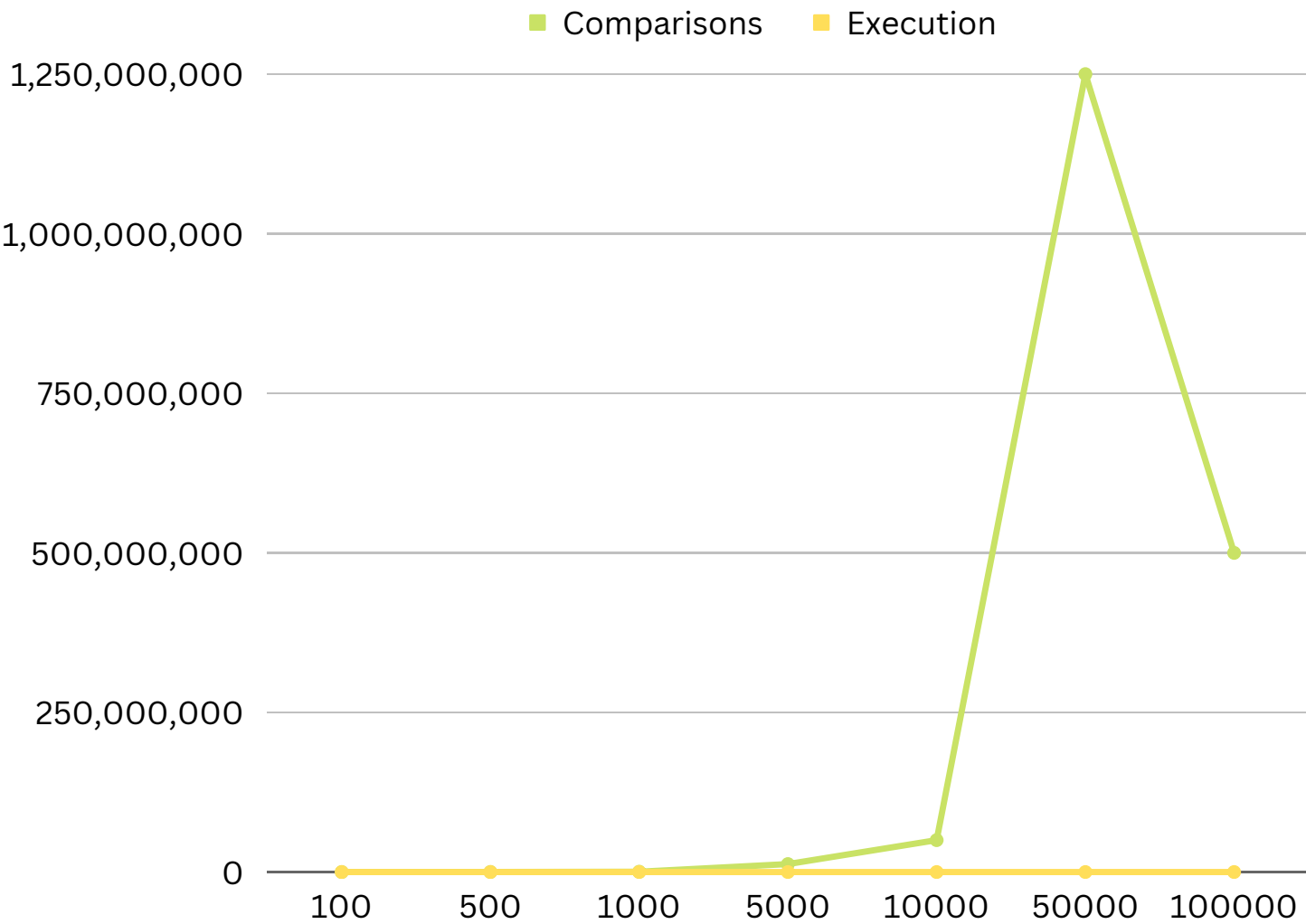


EQUATION ($N \log N$)

QUICK SORT ALGORITHM

IS THE NUMBER OF COMPARISONS IN QUICK SORT REALLY A GOOD PREDICTOR OF THE EXECUTION TIME?

Input Size (n)	Number of comparisons	Execution Time (µs)
100	5000	329.77ms
500	125000	530 ms
1000	500000	378.46ms
5000	12500000	674.06ms
10000	50000000	1170.88ms
50000	1250000000	7124.7ms
100000	5000000000	17512.28ms



EQUATION (N^2)

RESULT

According to the line graphs, it is shown that when the number of comparisons increases, the running time increases in all sorting algorithms.

As a result:

Since the number of comparisons is strongly correlated with the execution time, we computed the relationship between the number of comparisons and the execution time in each line graph to support our findings.



DESIGN AND ANALYSIS OF AN IMPROVED DIVIDE-AND-CONQUER ALGORITHM TO COMPUTE MATRIX MULTIPLICATION

SOLVE THE PROBLEM IN A BRUTE-FORCE MANNER

Brute force algorithm: this kind of algorithm is the most fundamental and straightforward, is a simple solution to a problem, or the first solution that springs to mind when we perceive the problem. Technically speaking, it is equivalent to iterating through all of the options available to resolve that issue.

```
def multiply(A, B, n):  
    C = []  
    for i in range(n):  
        for j in range(n):  
            for k in range(n):  
                C[i][j] += A[i][k]*B[k][j]  
    return C
```


RUNNING TIME ANALYSIS:

Code	Constant	Time
c = []	c1	1
for i in range(n):	c2	(n+1)
for j in range(n):	c3	n(n+1)
for k in range(n):	c4	(n+1)(n(n+1))-1
c[i][j] += A[i][k]*B[k][j]	c5	(n+1)(n(n+1))-1-1

Time complexity:

$$c1 + c2 \times (n+1) + c3 \times n(n+1) + c4 \times (n+1)(n(n+1))-1 + c5 \times (n+1)(n(n+1))-1-1 =$$
$$T(n) = O(n^3).$$

EFFICIENT ALGORITHM AND TIME COMPLEXITY

Practical code in java:

```
static int MatrixMultiply(int arrayOne[][size], int
arrayTwo[][size], int arrayThree[][size])
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            arrayThree [i][j] = 0;
            for (int k = 0; k < size; k++)
            {
                arrayThree [i][j] += arrayOne[i][k]*
arrayTwo[k][j];
            }
        }
    }
}
```

RUNNING TIME ANALYSIS:

Code	Constant	Time
for (int i = 0; i < size; i++)	c1	n+1
for (int j = 0; j < size; j++)	c2	n(n+1)
arrayThree [i][j] = 0;	c3	n(n+1)-1
for (int k = 0; k < size; k++)	c4	(n+1)(n(n+1))-1

Time complexity:

$$c1 \times (n+1) + c2 \times n(n+1) + c3 \times (n(n+1)-1) + c4 \times (n+1)(n(n+1))-1 =$$

$$T(n) = O(n^3).$$

The Team



**Renad
Alshahrani**

2200002949



**Fatima
Alrastem**

2200003453



**Shahed
Aljiaan**

2200003861



**Layan
Alghamdi**

2200002543



**Ghaida
Alotaibi**

2200002466