Name: Abu Nowshed Sakib

ID: 1705107

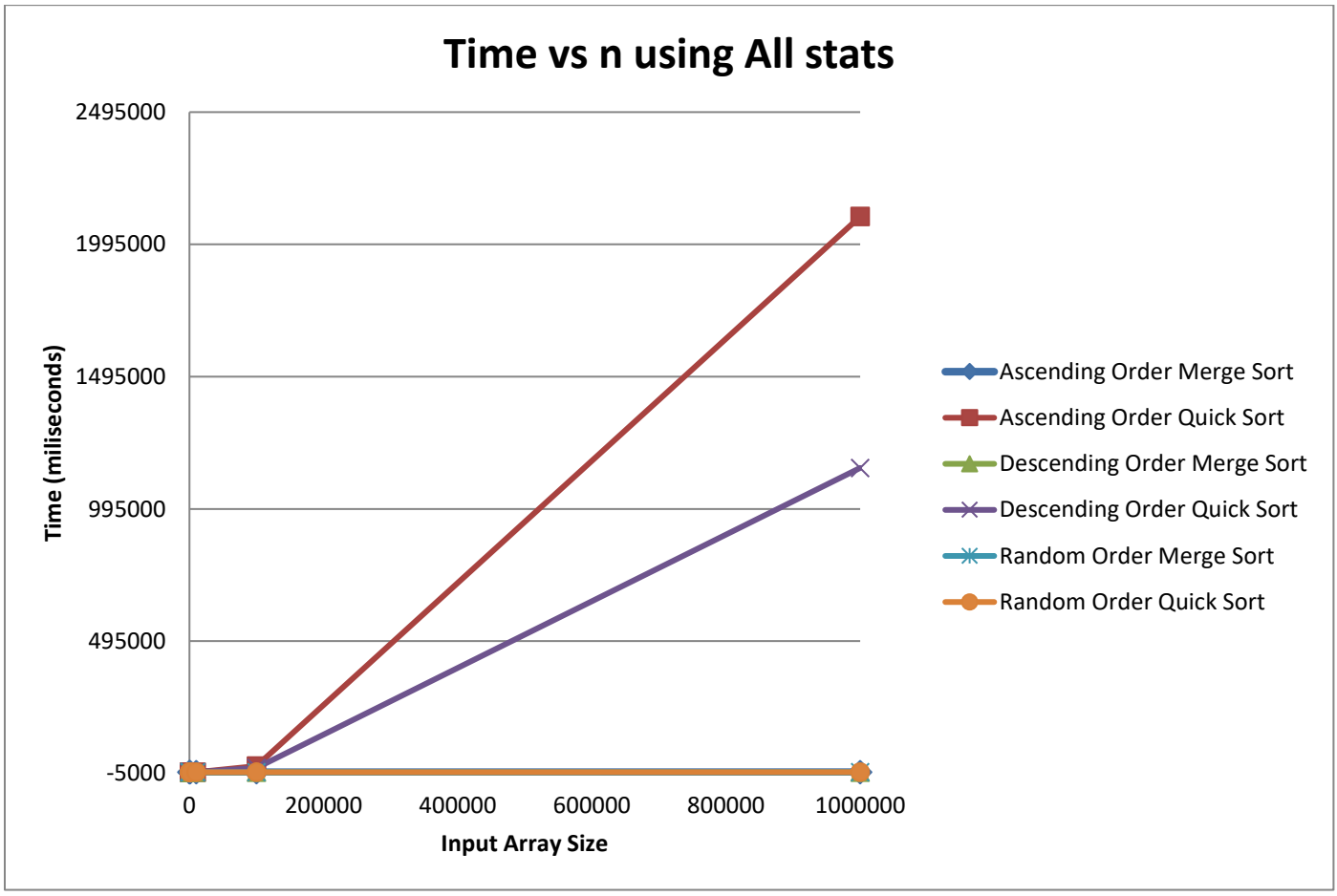
NB: ***I have measure the time in miliseconds(ms).

Data Table

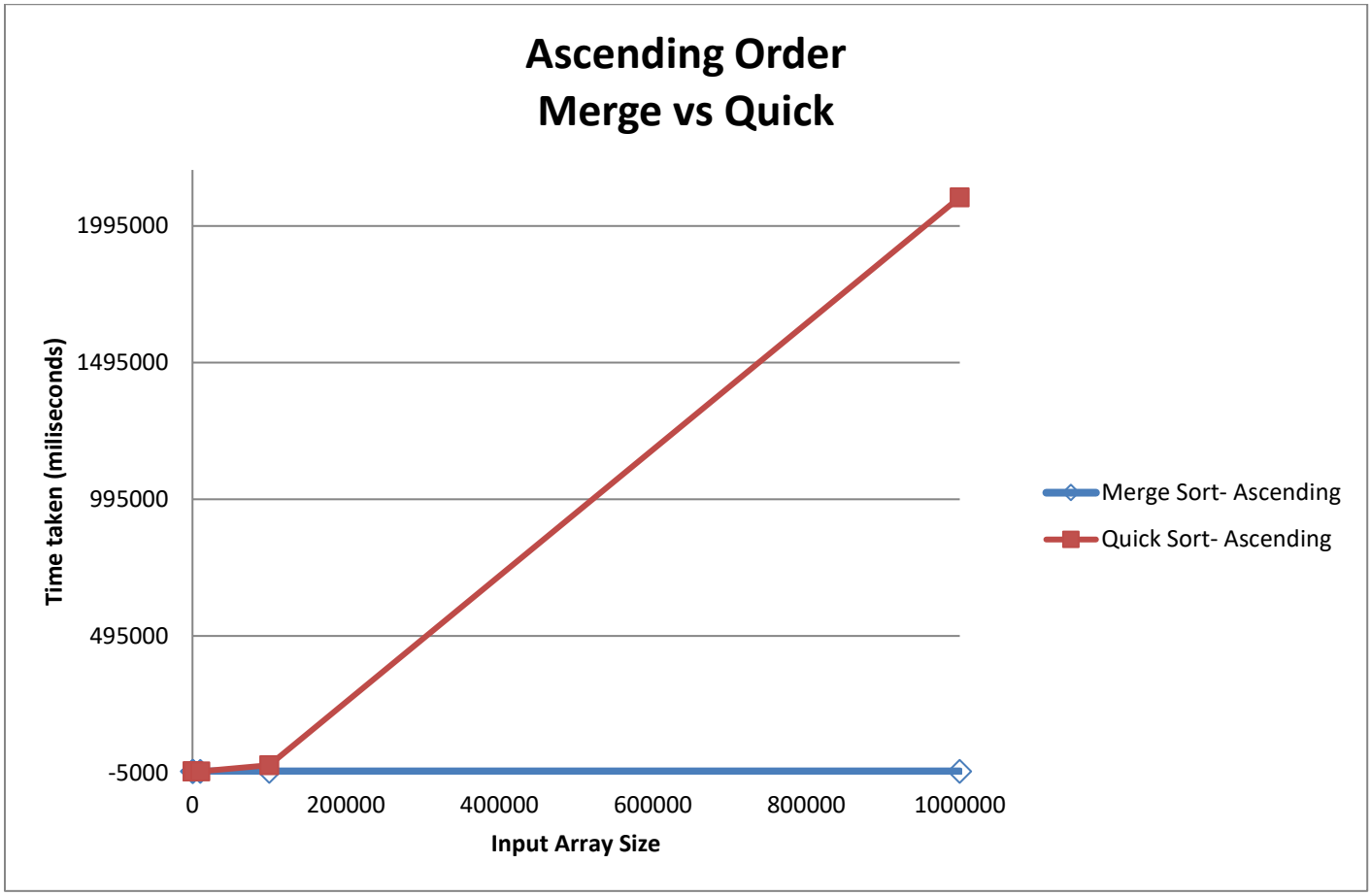
Input Order	n= / Sorting Algorithm	10	100	1000	10000	100000	1000000
Ascending	**Merge**	0.00044	0.0055	0.059	0.69	8.5	107
	Quick	0.0004	0.0246	2.335	228.03	23000	2100000
Descending	**Merge**	0.00031	0.0046	0.06	0.78	8.7	104
	Quick	0.00031	0.0169	1.559	151.94	15800	1150000
Random	**Merge**	0.00036	0.0063	0.106	1.24	15	181
	Quick	0.00012	0.0027	0.072	1.09	13.5	207

Plot

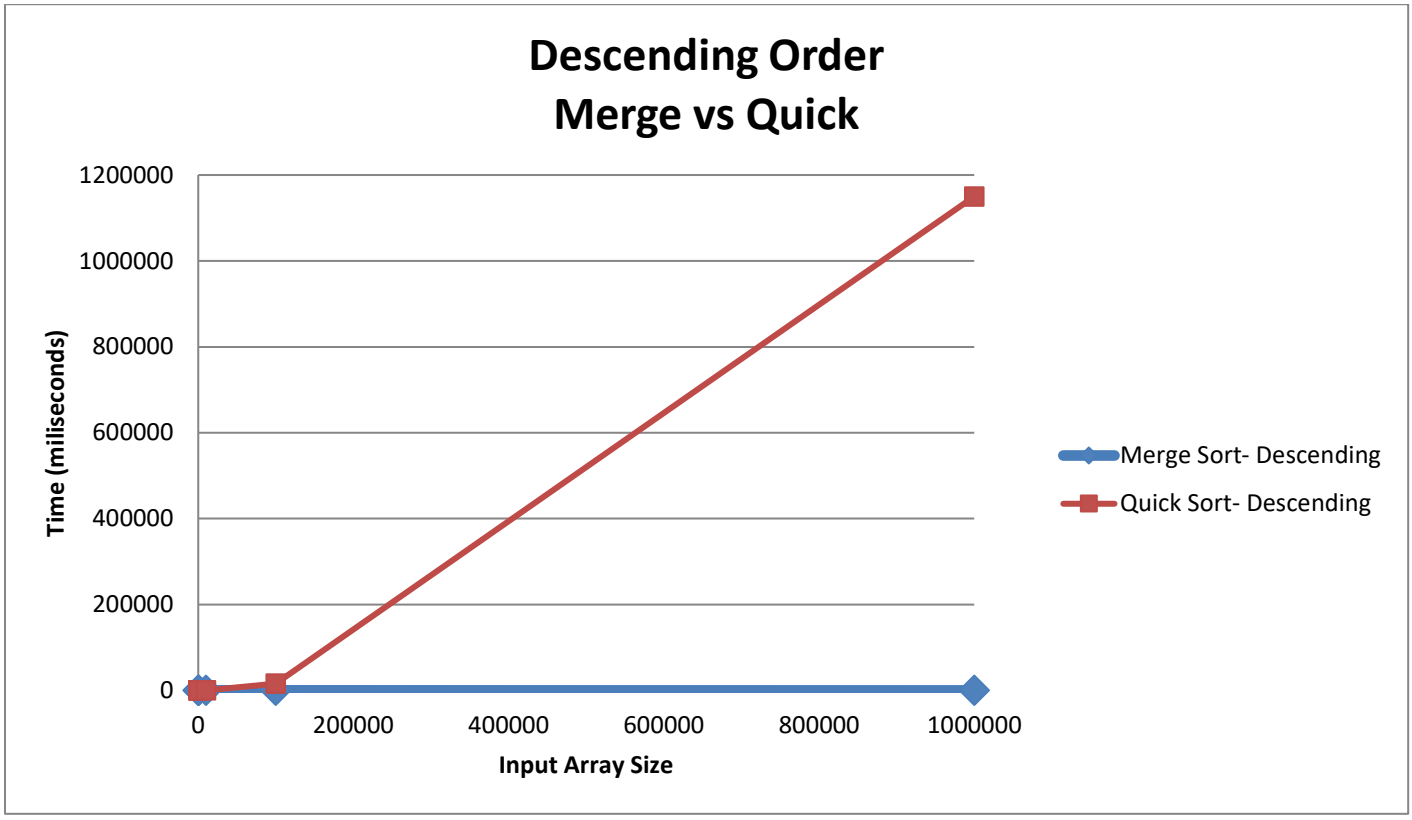
#1. A single plot of time vs. n using all statistics:



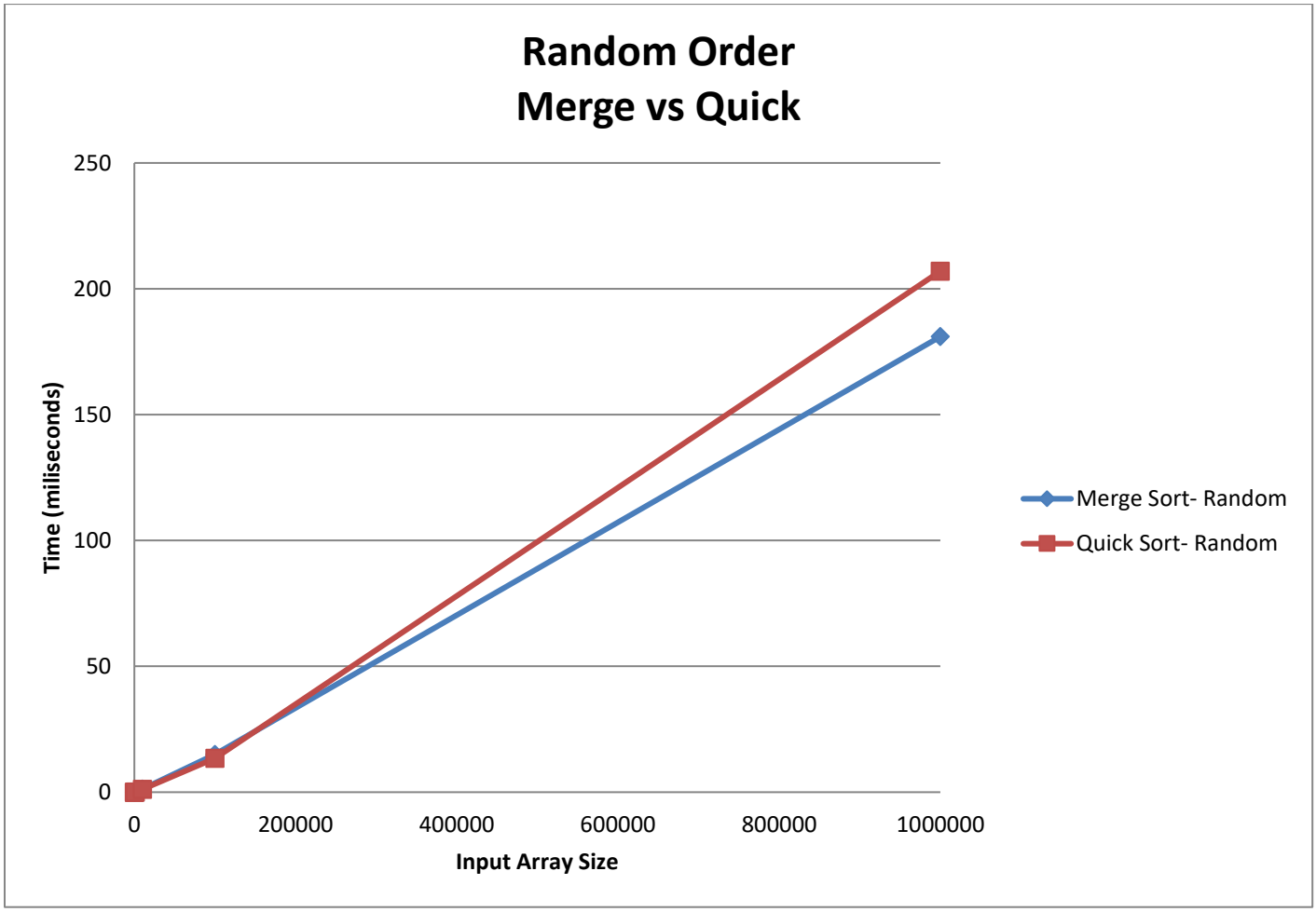
#2. Ascending Order (Merge Sort vs. Quick Sort):



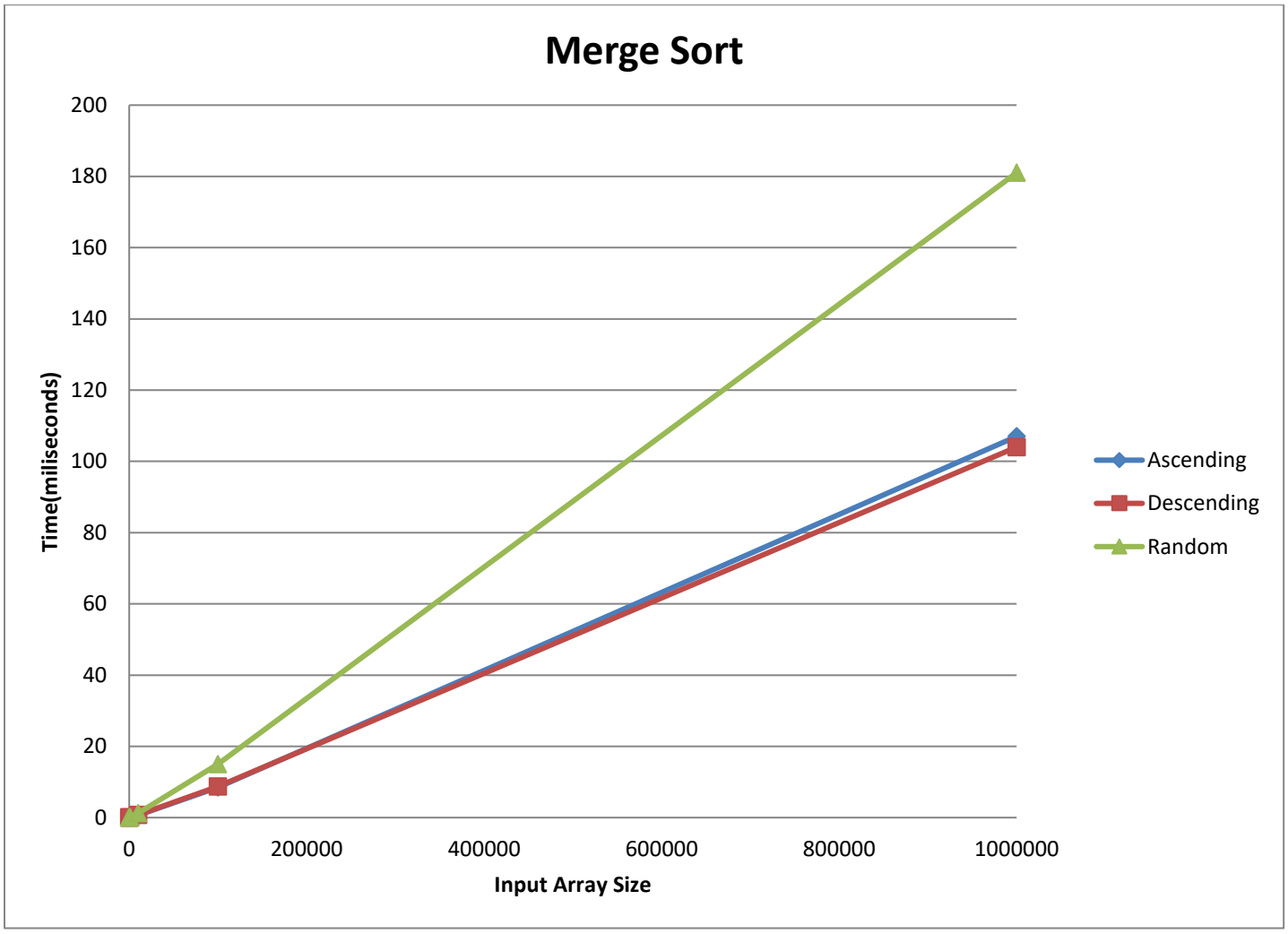
#3. Descending Order (Merge Sort vs. Quick Sort)



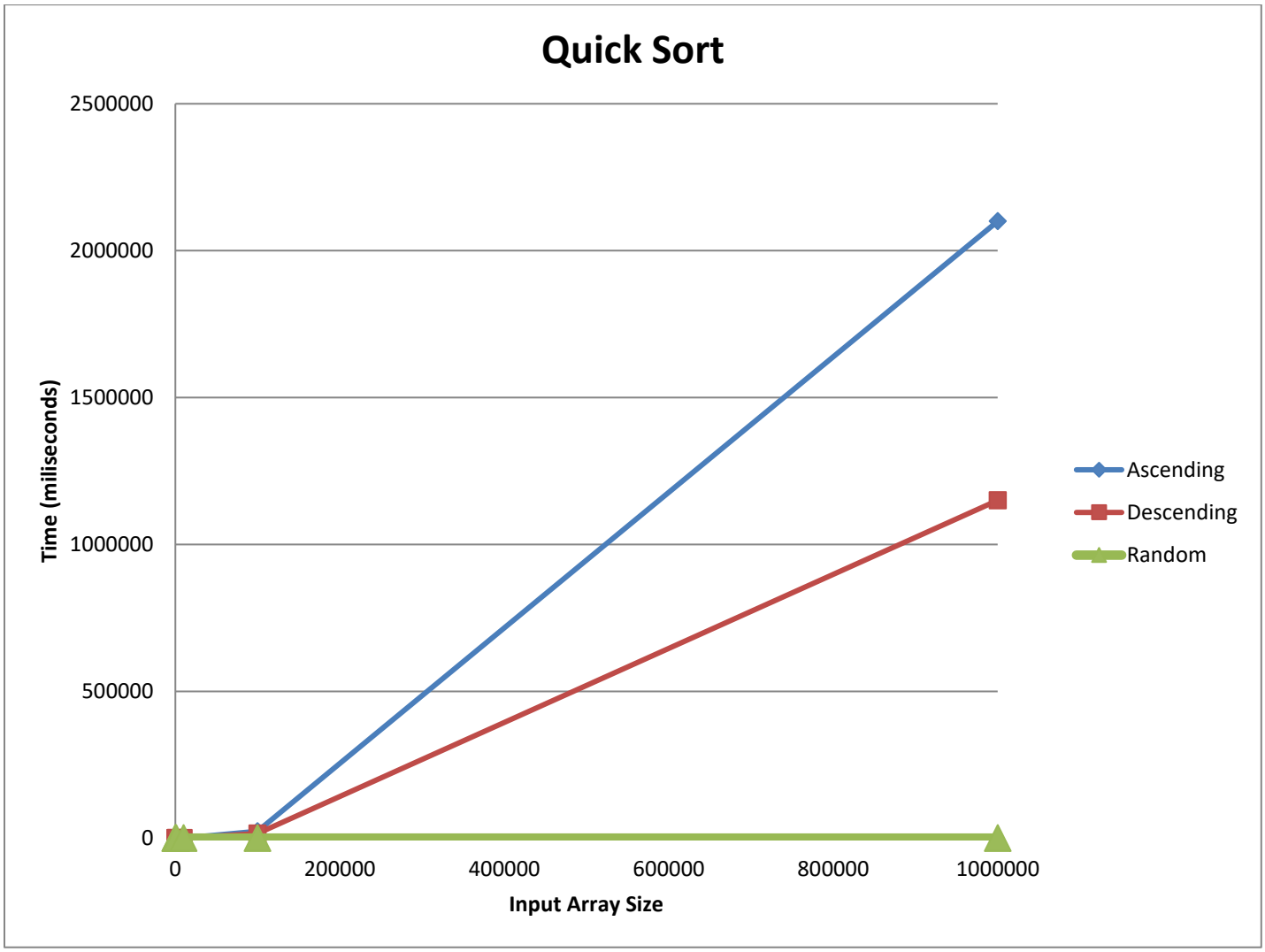
#4. Random Order (Merge Sort vs. Quick Sort)



#5. Merge Sort (Ascending vs. Descending vs. Random)



#6. Quick Sort (Ascending vs. Descending vs. Random)



Complexity Analysis

Quick Sort

Ascending Order:

When the elements are sorted in ascending order, the partition always picks the last(largest) element of the array as pivot. So the partition function produces a subproblem with (n-1) elements and another subproblem with 0 elements. So the recurrence is:

$T(n) = T(0) + T(n-1) + \theta(n)$

$\Rightarrow T(n) = T(n-1) + \theta(n)$

The solution of above recurrence is $\boldsymbol{\theta(n^2)}$

Descending Order:

When the elements are sorted in descending order, the partition always picks the last(smallest) element of the array as pivot. So the partition function produces a subproblem with (n-1) elements and another subproblem with 0 elements. So the recurrence is:

$T(n) = T(0) + T(n-1) + \theta(n)$

$\Rightarrow T(n) = T(n-1) + \theta(n)$

The solution of above recurrence is $\boldsymbol{\theta(n^2)}$

Random Order:

This time the partition can pick any element in the array. So partition function will divide the problem into two subproblems. One subproblem will contain k elements and the other one will contain (n-k-1) elements. The depth of recursion tree is O(log n), where the cost of each level is O(n). So the running time complexity is $\boldsymbol{O(n \log n)}$

Merge Sort

The time complexity of Merge sort for every input order is same.

We follow divide and conquer method in merge sort. So-

Divide: The divide step just computes the middle of the subarray, which takes constant time $\theta(1)$

Conquer: We recursively solve two subproblems, each of size n=2, which contributes 2T(n/2) to the running time.

Combine: The MERGE procedure on an n-element subarray takes time $\theta(n)$

So the recurrence function is-

$T(n) = 2T(n/2) + \theta(n)$

If we design the recursion tree the top level has total cost cn.

The next level has total cost = c(n/2) + c(n/2) = cn

The next level has total cost = c(n/4)+ c(n/4)+ c(n/4)+ c(n/4) = cn and so on…

So, every level of recursion tree has total cost of cn.

As we divide the array in two equal subproblems in every level, so the depth of the recursion tree is = lg n + 1 (where n is the number of leaves as well as the input size)

To compute the total cost of the recurrence we simply add up the costs of all levels.

So the total cost = cn(lg n + 1) = cn lg n + cn. Ignoring the low order term (cn) and constant c—

The time complexity is = **θ(nlg n)**

Machine Configuration:

Processor: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 1992 MHz, 4 Core(s), 8 Logical Processor(s)
Installed Memory (RAM): 8.00 GB (7.88 GB usable)
System Type: 64-bit Operating System, x64-based processor
Microsoft Windows Version 1803 (OS Build 17134.765)