CSE204
Offline 7 – Report

Submitted by:
ID: 1805012

Complexity Analysis:
Merge Sort:

The merge sort algorithm follows the divide and conquer method.
The operation is as follows:

Divide: Divide the n-element sequence in two subsequences of n/2 elements each.

Conquer: Sort the two subsequences recursively using merge sort.

Combine: Merge the two sorted subsequences to produces the sorted sequence.

Irrespective of the order (ascending, descending, random) the merge sort algorithm always divides the array into two equal(almost) sized array.

So, the time complexity of merge sort can be expressed in terms of the steps.

$T(n) = 2T(n/2) + D(n) + C(n)$

Here, $D(n)$ = time complexity of divide

$C(n)$ = time complexity of conquer step

$2T(n/2)$ is for sorting the 2 subsequences recursively.

Divide takes constant time, which is computing the middle of the array. So, $D(n) = \Theta(n)$

Combine step merges the two sorted subsequences which takes linear time. $C(n) = \Theta(n)$

$T(n) = 2T(n/2) + \Theta(n) + \Theta(1) = 2T(n/2) + \Theta(n)$, which results in $\Theta(n\log n)$.

Quick Sort:

The quick sort algorithm also follows the divide and conquer method.
The operation:

Divide: Partition the array into two subarray such that the elements of one partition is smaller than a certain element (pivot) and the elements of another partition is equal or bigger than the pivot.

Conquer: Sort the two subarrays by recursive calls to quicksort.

Combine: After recursive calls the subarrays are already sorted, no further procedure is needed.

Quick sort depends on the order of the numbers, more specifically the pivot element. If the pivot is the smallest or the largest element in the sequence, then one partition will be empty and the another partition will have (n-1)-elements.

For ascending or descending order, the time complexity of quicksort will be -

$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$, which results in

$T(n) = \Theta(n^2)$

Here, $\Theta(n)$ for partitioning, $T(0)$ for 0 element partition, $T(n-1)$ for n-1 element partition.

For random order, let’s say the pivot partitions the array into n/p and (n-n/p) size.

So, $T(n) = T(n/p) + T(n-n/p) + \Theta(n)$, which is close to $\Theta(n\log n)$

Machine Configuaration:

Processor:

Architecture	:	x86_64
CPU op-mode(s)	:	32-bit, 64-bit
Model name	:	Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz
CPU MHz	:	800.144
CPU max MHz	:	4600.0000
CPU min MHz	:	800.0000
L1d cache	:	32K
L1i cache	:	32K
L2 cache	:	256K
L3 cache	:	9216K

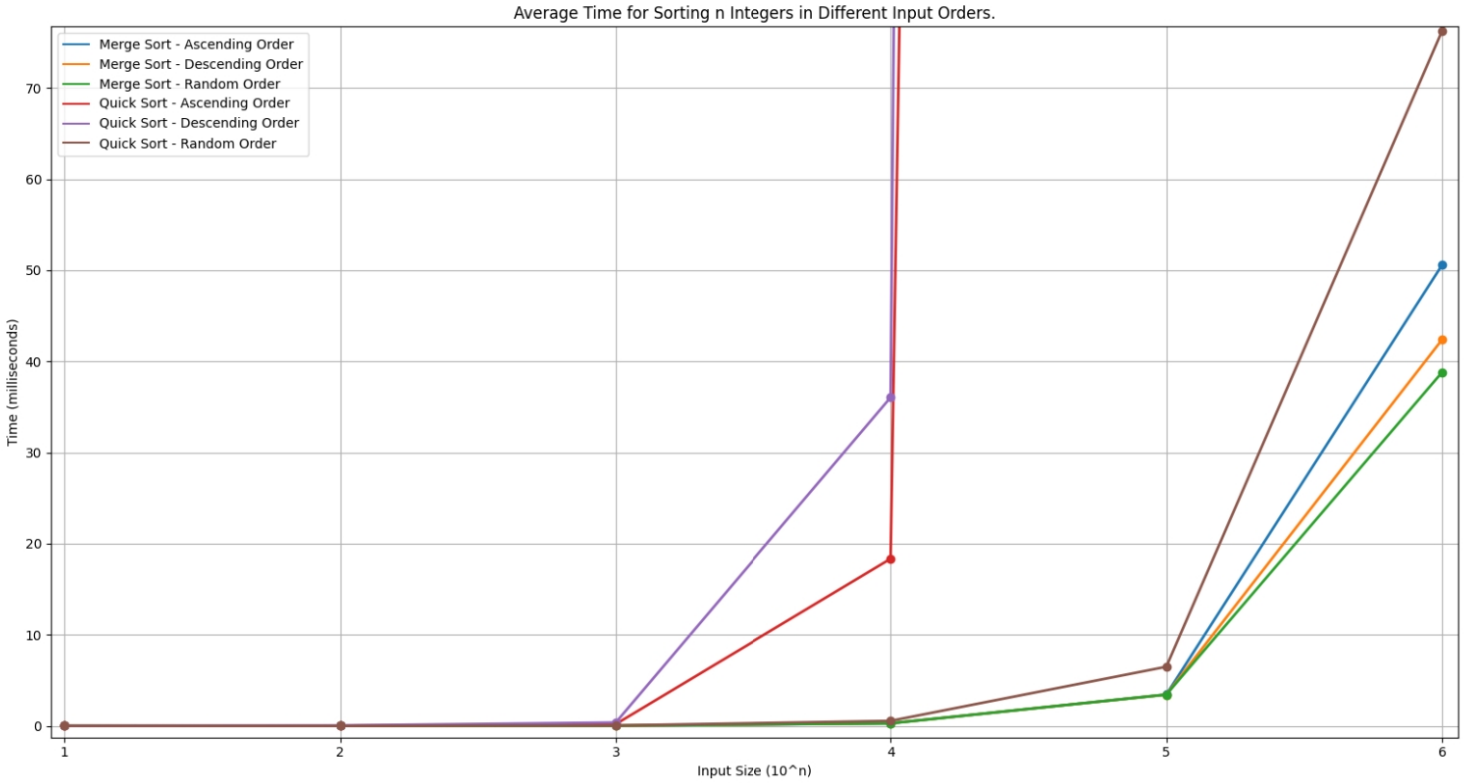
Memory :

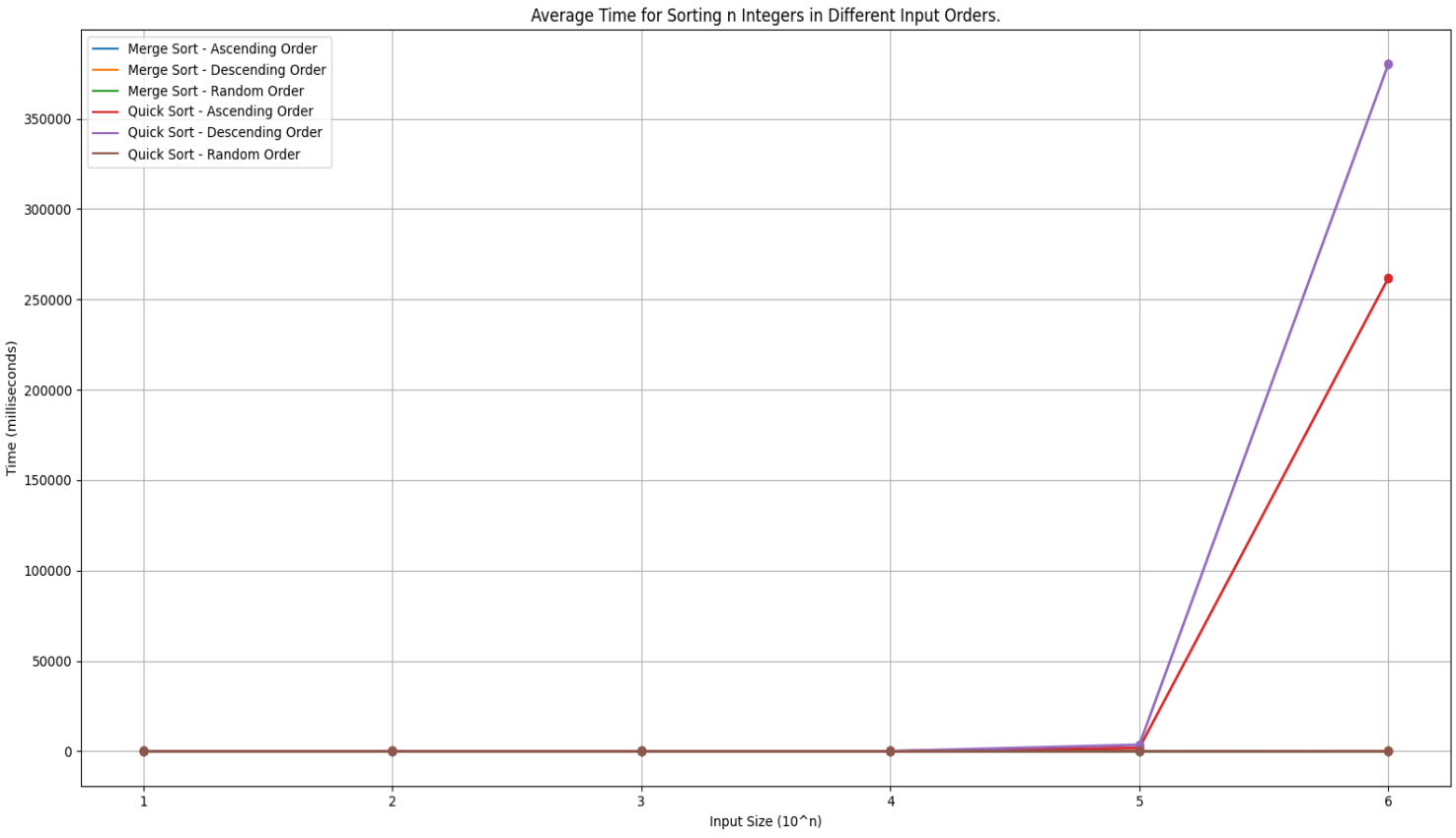
Total Width: 64 bits
Data Width: 64 bits
Size: 8192 MB
Type: DDR4
Type Detail: Synchronous
Speed: 2400 MHz
Configured Memory Speed: 2400 MHz

Table: Average time for sorting n integers in different input orders

Input Order	N = / Sorting Algorithm	10	100	1000	10000	100000	1000000
Ascending	Merge	0.0205 ms	0.01108 ms	0.02059 ms	0.2708 ms	3.4365 ms	50.5702 ms
	Quick	0.0161 ms	0.01023 ms	0.1923 ms	18.3373 ms	1840.7850 ms	261823.9506 ms
Descending	Merge	0.0150 ms	0.0035 ms	0.0210 ms	0.2796 ms	3.4236 ms	42.3814 ms
	Quick	0.0129 ms	0.0479 ms	0.3699 ms	36.0456 ms	3586.4785 ms	380210.8088 ms
Random	Merge	0.0199 ms	0.0022 ms	0.0212 ms	0.2825 ms	3.4235 ms	38.8009 ms
	Quick	0.0108 ms	0.0040 ms	0.0415 ms	0.5473 ms	6.4872 ms	76.2583 ms

Plot:


Average Time for Sorting n Integers in Different Input Orders.
Merge Sort - Ascending Order
Merge Sort - Descending Order
Merge Sort - Random Order
Quick Sort - Ascending Order
Quick Sort - Descending Order
Quick Sort - Random Order
Time (milliseconds)
Input Size (10^n)


Average Time for Sorting n Integers in Different Input Orders.
Merge Sort - Ascending Order
Merge Sort - Descending Order
Merge Sort - Random Order
Quick Sort - Ascending Order
Quick Sort - Descending Order
Quick Sort - Random Order
Time (milliseconds)
Input Size (10^n)