

Algorithm Efficiency and Sorting

Maryam Shahid



21801344

CS202

Assignment 1

Question 1: Tracing [4, 8, 3, 7, 6, 2, 1, 5]

a) Insertion sort

 Light blue is the first element of the unsorted region;
 Dark blue elements represent the sorted part of the array.

Initial Array:

4	8	3	7	6	2	1	5
---	---	---	---	---	---	---	---

After Pass 1:

4	8	3	7	6	2	1	5
---	---	---	---	---	---	---	---

After Pass 2:

3	4	8	7	6	2	1	5
---	---	---	---	---	---	---	---

After Pass 3:

3	4	7	8	6	2	1	5
---	---	---	---	---	---	---	---

After Pass 4:

3	4	6	7	8	2	1	5
---	---	---	---	---	---	---	---

After Pass 5:

2	3	4	6	7	8	1	5
---	---	---	---	---	---	---	---

After Pass 6:

1	2	3	4	6	7	8	5
---	---	---	---	---	---	---	---


After Pass 7:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Sorted Array:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

b) Selection sort

 Light blue element is selected;

 Dark blue elements represent the sorted part of the array.

Initial Array:

4	8	3	7	6	2	1	5
---	---	---	---	---	---	---	---

After 1st Swap:

4	5	3	7	6	2	1	8
---	---	---	---	---	---	---	---

After 2nd Swap:

4	5	3	1	6	2	7	8
---	---	---	---	---	---	---	---

After 3rd Swap:

4	5	3	1	2	6	7	8
---	---	---	---	---	---	---	---

After 4th Swap:

4	2	3	1	5	6	7	8
---	---	---	---	---	---	---	---

After 5th Swap:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Sorted Array:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

c) **Bubble sort**



Light blue element is selected;



Dark blue elements represent the sorted part of the array.

1st Pass:

4	8	3	7	6	2	1	5
---	---	---	---	---	---	---	---

4	8	3	7	6	2	1	5
---	---	---	---	---	---	---	---

4	3	8	7	6	2	1	5
---	---	---	---	---	---	---	---

4	3	7	8	6	2	1	5
---	---	---	---	---	---	---	---

4	3	7	6	8	2	1	5
---	---	---	---	---	---	---	---

4	3	7	6	2	8	1	5
---	---	---	---	---	---	---	---

4	3	7	6	2	1	8	5
---	---	---	---	---	---	---	---

4	3	7	6	2	1	5	8
---	---	---	---	---	---	---	---

2nd Pass:

4	3	7	6	2	1	5	8
---	---	---	---	---	---	---	---

3	4	7	6	2	1	5	8
---	---	---	---	---	---	---	---

3	4	7	6	2	1	5	8
---	---	---	---	---	---	---	---

3	4	6	7	2	1	5	8
---	---	---	---	---	---	---	---

3	4	6	2	7	1	5	8
---	---	---	---	---	---	---	---

3	4	6	2	1	7	5	8
---	---	---	---	---	---	---	---

3	4	6	2	1	5	7	8
---	---	---	---	---	---	---	---

3rd Pass:

3	4	6	2	1	5	7	8
---	---	---	---	---	---	---	---

3	4	6	2	1	5	7	8
---	---	---	---	---	---	---	---

3	4	6	2	1	5	7	8
---	---	---	---	---	---	---	---

3	4	2	6	1	5	7	8
---	---	---	---	---	---	---	---

3	4	2	1	6	5	7	8
---	---	---	---	---	---	---	---

3	4	2	1	5	6	7	8
---	---	---	---	---	---	---	---

4th Pass:

3	4	2	1	5	6	7	8
---	---	---	---	---	---	---	---

3	4	2	1	5	6	7	8
---	---	---	---	---	---	---	---

3	2	4	1	5	6	7	8
---	---	---	---	---	---	---	---

3	2	1	4	5	6	7	8
---	---	---	---	---	---	---	---

3	2	1	4	5	6	7	8
---	---	---	---	---	---	---	---

5th Pass:

3	2	1	4	5	6	7	8
---	---	---	---	---	---	---	---

2	3	1	4	5	6	7	8
---	---	---	---	---	---	---	---

2	1	3	4	5	6	7	8
---	---	---	---	---	---	---	---

2	1	3	4	5	6	7	8
---	---	---	---	---	---	---	---

6th Pass:

2	1	3	4	5	6	7	8
---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

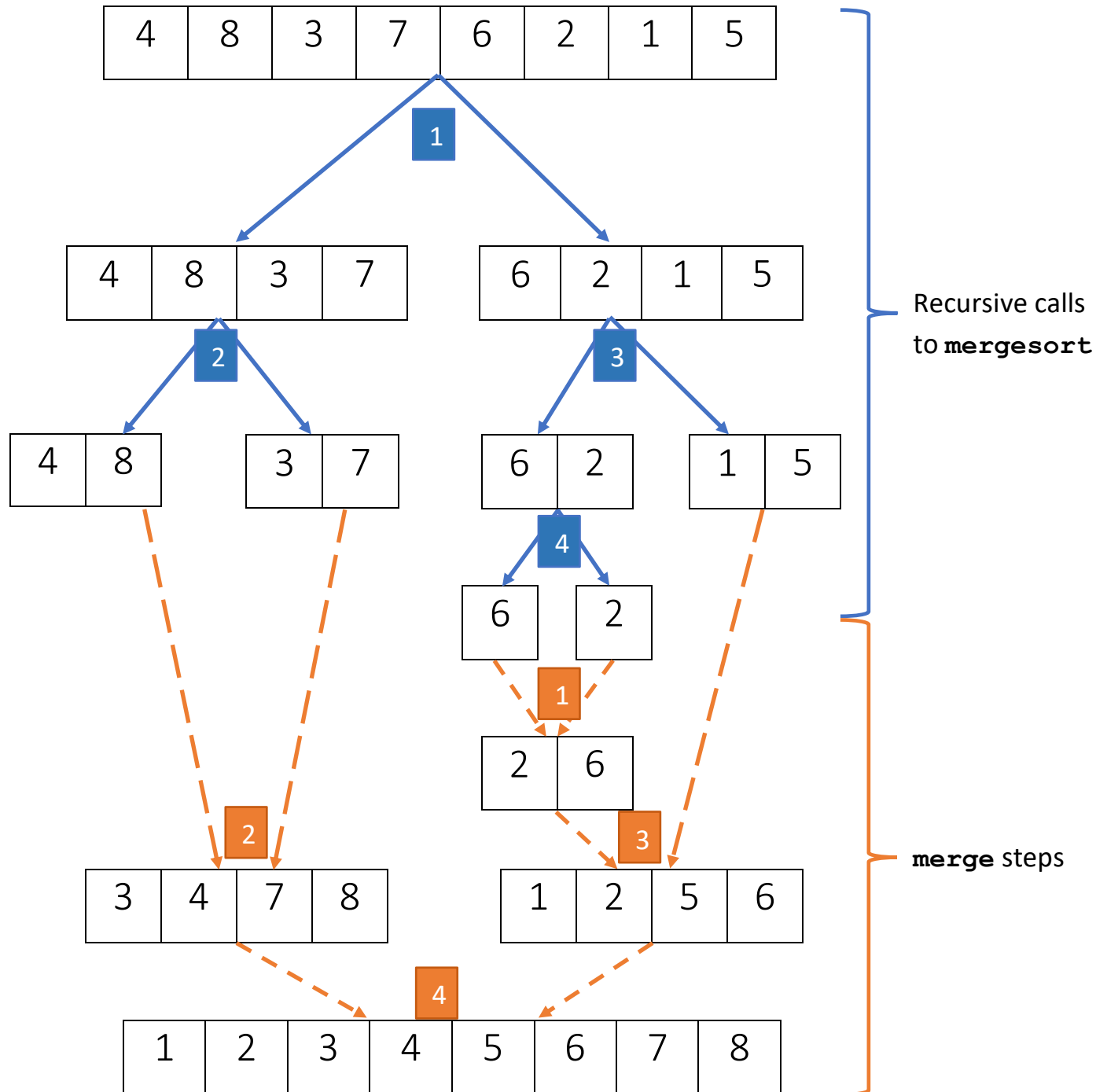
Sorted Array:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

d) Merge sort

Blue arrow represents calls to **merge**;

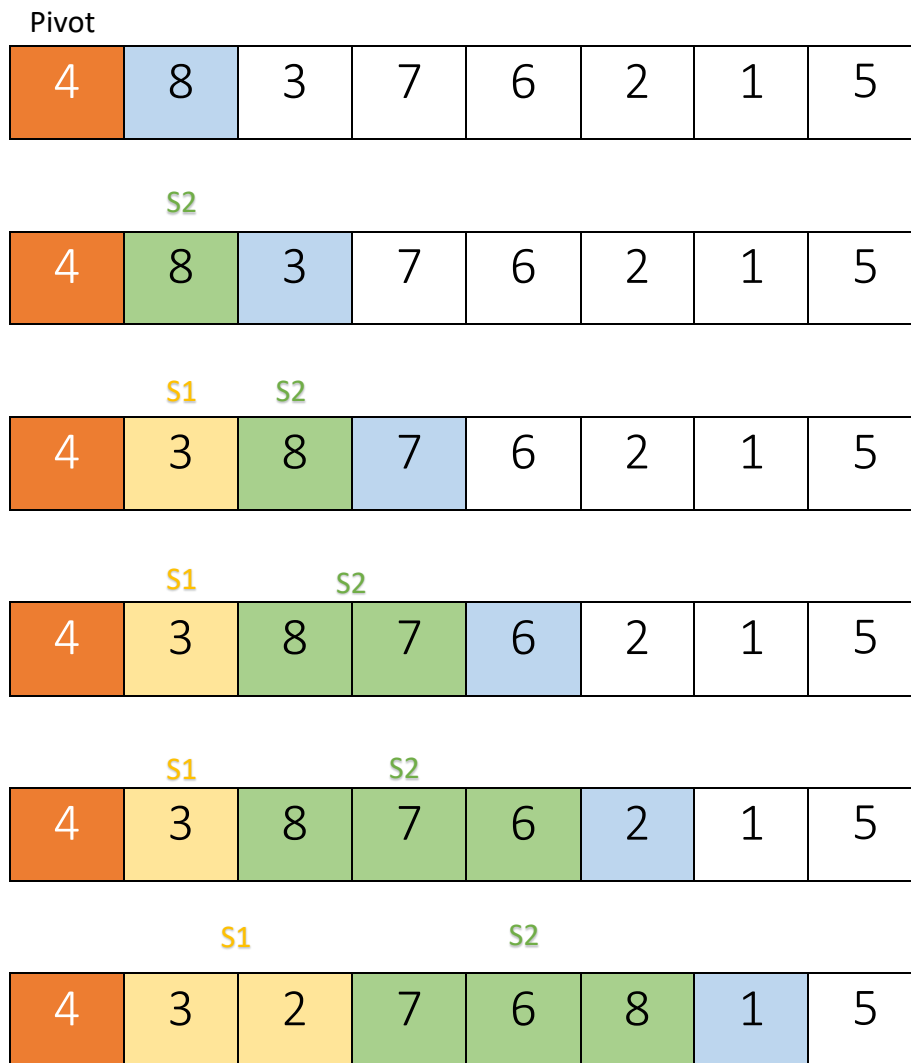
Orange arrow represents calls to **mergesort**.



e) Quick Sort

- Orange element is the pivot;
- Light blue is the first element of the unsorted region;
- Yellow is sublist 1, elements lessor than pivot;
- Green is sublist 2, elements greater than pivot;
- Dark blue elements represent the sorted part of the array.

1st call to **partition**:



S1				S2			
4	3	2	1	6	8	7	5

S1				S2			
4	3	2	1	6	8	7	5

S1				S2			
1	3	2	4	6	8	7	5

1	3	2	4	6	8	7	5
---	---	---	---	---	---	---	---

Recursive calls of **quicksort** on S1 and S2

S1:

2nd call to **partition** on S1:

Pivot

1	3	2	4	6	8	7	5
---	---	---	---	---	---	---	---

S2'

1	3	2	4	6	8	7	5
---	---	---	---	---	---	---	---

S2'

1	3	2	4	6	8	7	5
---	---	---	---	---	---	---	---

Recursive call of **quicksort** on S2':

3rd call to **partition** on S2':

Pivot

1	3	2	4	6	8	7	5
---	---	---	---	---	---	---	---

s1'

1	3	2	4	6	8	7	5
---	---	---	---	---	---	---	---

s1'

1	2	3	4	6	8	7	5
---	---	---	---	---	---	---	---

1	2	3	4	6	8	7	5
---	---	---	---	---	---	---	---

S2:

4th call to **partition** on S2:

Pivot

1	2	3	4	6	8	7	5
---	---	---	---	---	---	---	---

s2''

1	2	3	4	6	8	7	5
---	---	---	---	---	---	---	---

1	2	3	4	6	8	7	5
---	---	---	---	---	---	---	---

s2''

1	2	3	4	6	5	7	8
---	---	---	---	---	---	---	---

s1''

s2''

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

s1''

s2''

Sorted Array:

1	2	3	4	5	7	6	5
---	---	---	---	---	---	---	---

Question 2: Recurrence equation for the time requirements in the worst case of:

Merge sort

<u>Merge-Sort</u> (A, p, r)	→	$T(n)$
if p = r then	→	$\Theta(1)$
return		
else	→	$\Theta(1)$
q ← ⌊ (p+r)/2 ⌋	→	
<u>Merge-Sort</u> (A, p, q)	→	$T(n/2)$
<u>Merge-Sort</u> (A, q+1, r)	→	$T(n/2)$
<u>Merge</u> (A, p, q, r)	→	$\Theta(n)$
endif		

From this merge sort algorithm, we note that the equation for recurrence relation is

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + n & \text{otherwise} \end{cases}$$

Since $T(n) = 2T\left(\frac{n}{2}\right) + n$ → Equation 1

∴ $T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right) + \frac{n}{2}$, replacing $T\left(\frac{n}{2}\right)$ in equation 1 gives:

$$T(n) = 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n$$

$$T(n) = 2^2T\left(\frac{n}{2^2}\right) + 2n \rightarrow \text{Equation 2}$$

$\therefore T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}$, replacing $T\left(\frac{n}{2^2}\right)$ in equation 2 gives:

$$T(n) = 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + n$$

$$T(n) = 2^3T\left(\frac{n}{2^3}\right) + 3n \quad \longrightarrow \text{Equation 3}$$

⋮
Continue for k times

$$T(n) = 2^kT\left(\frac{n}{2^k}\right) + kn$$

$$\text{Assume } T\left(\frac{n}{2^k}\right) = T(1)$$

$$\therefore \frac{n}{2^k} = 1 \text{ so, } n = 2^k \text{ and } k = \log(n)$$

$$T(n) = n T(1) + \log(n) n$$

$$T(n) = n + n \log(n)$$

$$\text{Thus, } \theta(n \log(n))$$

Quick Sort

In worst case of quick sort, the list is already sorted. In that case, the partition will be at the start of the list. A list of size n will be recursively divided into two sublists of size 0 and $(n - 1)$, and since a recursive call is $O(n)$, we form the following recurrence relation:

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n - 1) + n & \text{otherwise} \end{cases}$$

Since $T(n) = T(n - 1) + n \longrightarrow$ Equation 1

$\therefore T(n - 1) = T(n - 2) + n - 1$, replacing $T(n - 1)$ in equation 1 gives:

$$T(n) = (T(n - 2) + n - 1) + n$$

$$T(n) = T(n - 2) + (n - 1) + n \longrightarrow \text{Equation 2}$$

$\therefore T(n - 2) = T(n - 3) + n - 2$, replacing $T(n - 2)$ in equation 2 gives:

$$T(n) = (T(n - 3) + n - 2) + (n - 1) + n$$

$$T(n) = T(n - 3) + (n - 2) + (n - 1) + n \longrightarrow \text{Equation 3}$$

⋮
Continue for k times

$$T(n) = T(n - k) + (n - (k - 1)) + (n - (k - 2)) + \cdots + (n - 1) + n$$

Assume $T(n - k) = T(0)$

$$\therefore n - k = 0 \text{ so, } n = k$$

$$T(n) = T(n - n) + (n - n + 1) + (n - n + 2) + \cdots + (n - 1) + n$$

$$T(n) = T(0) + 1 + 2 + \cdots + (n - 1) + n$$

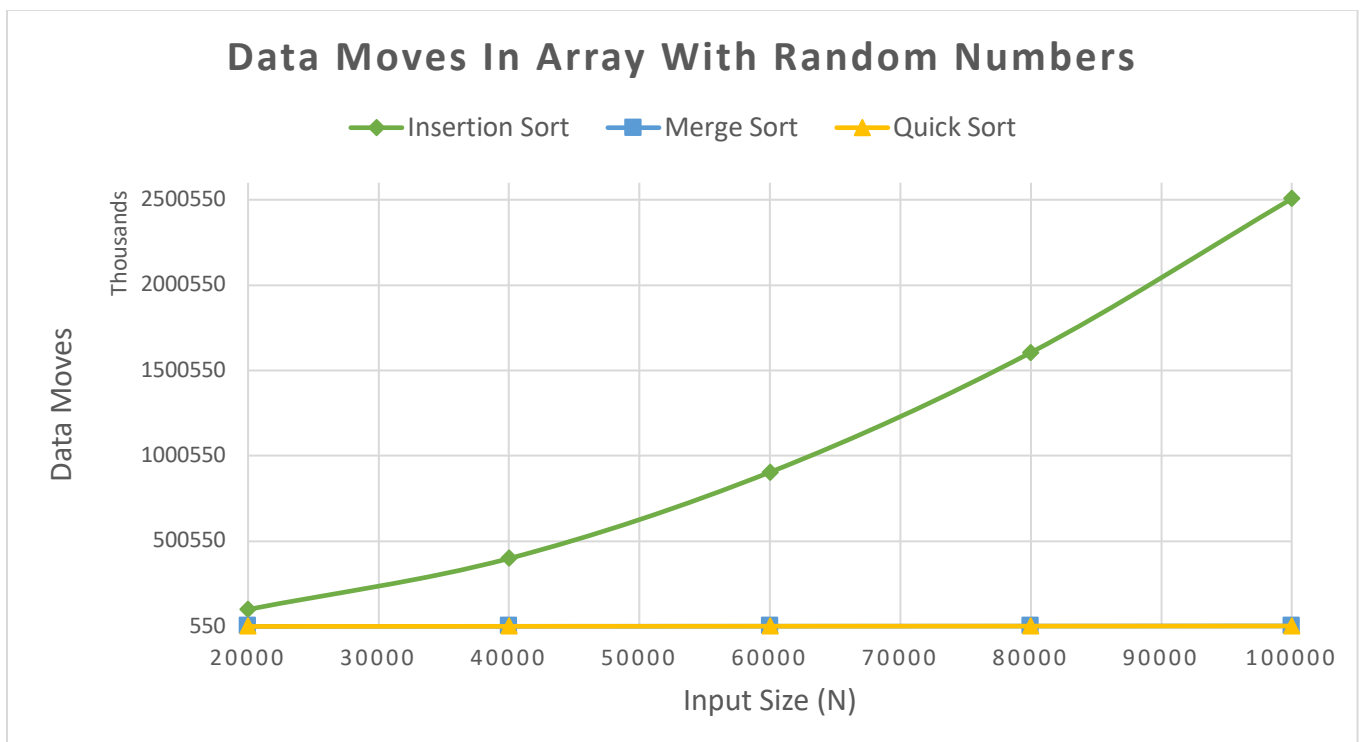
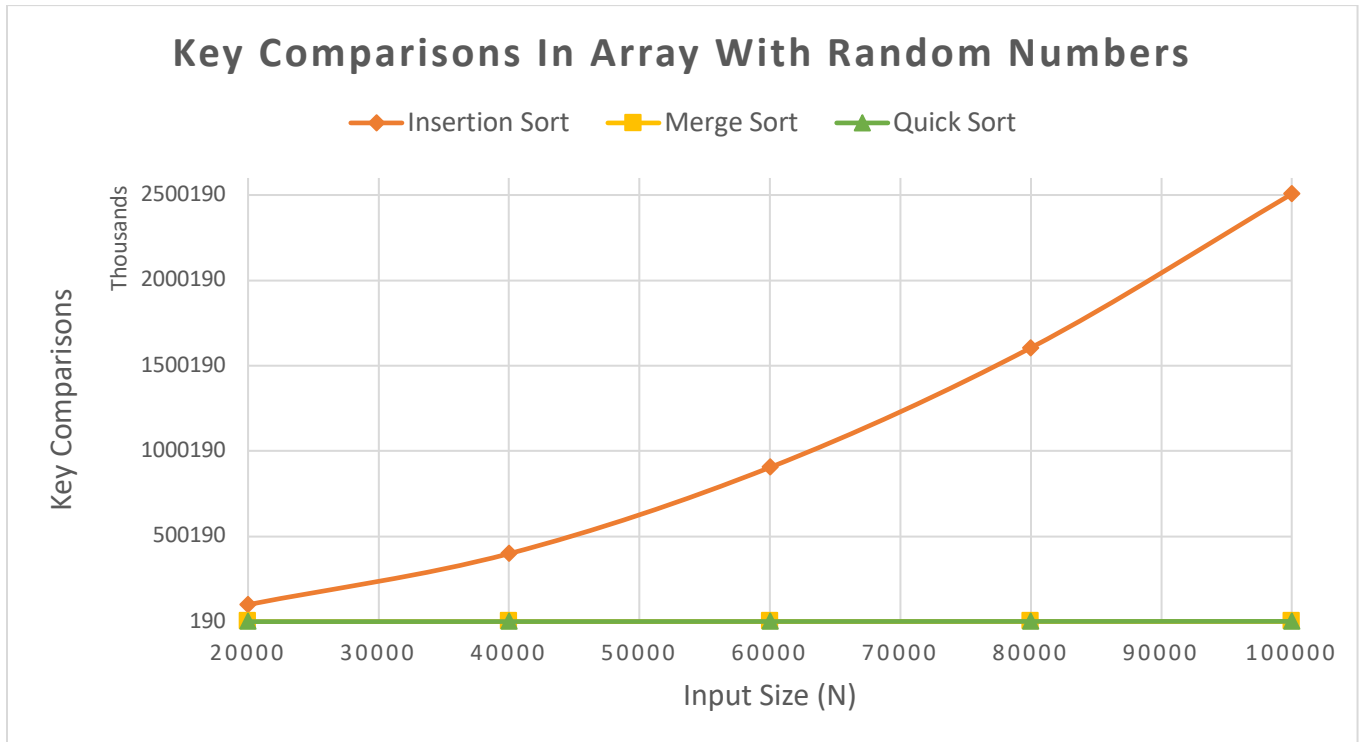
$$T(n) = 1 + \sum_{i=0}^n i$$

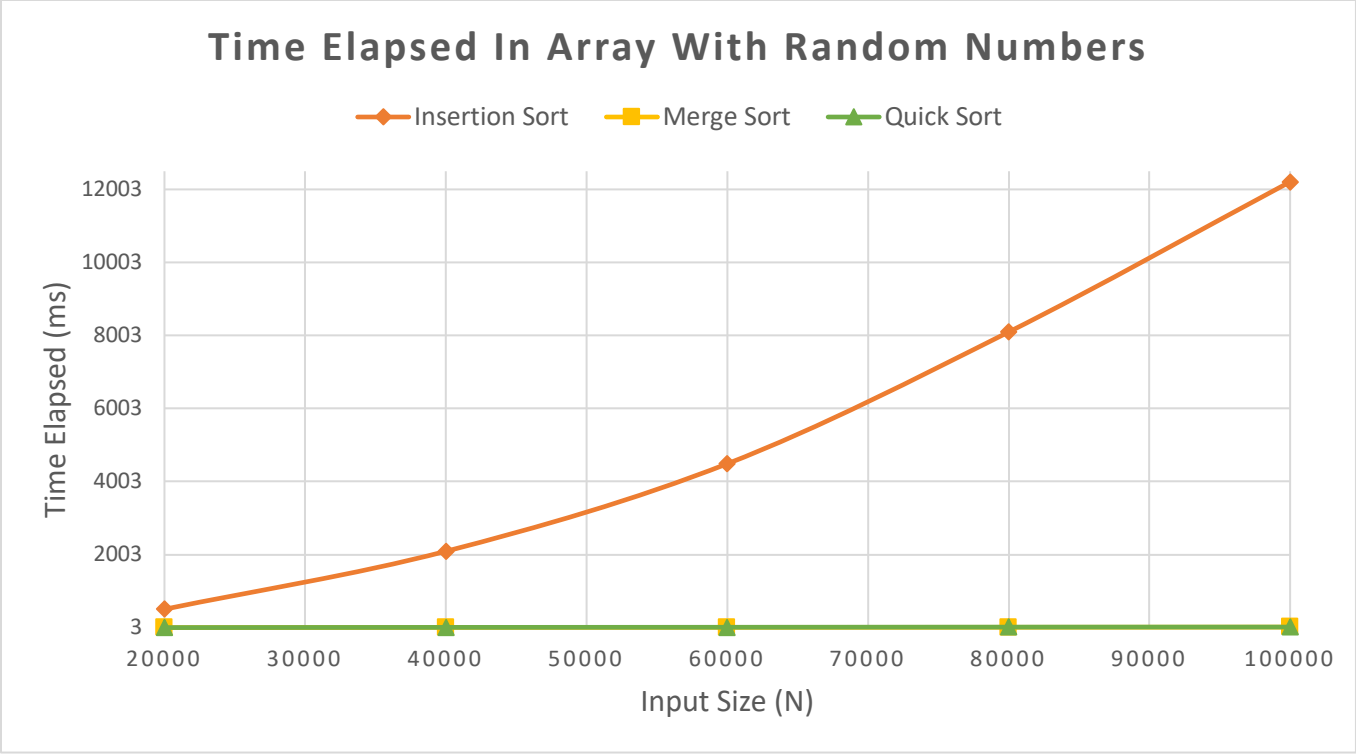
$$T(n) = 1 + \frac{n(n + 1)}{2}$$

Thus, $\theta(n^2)$

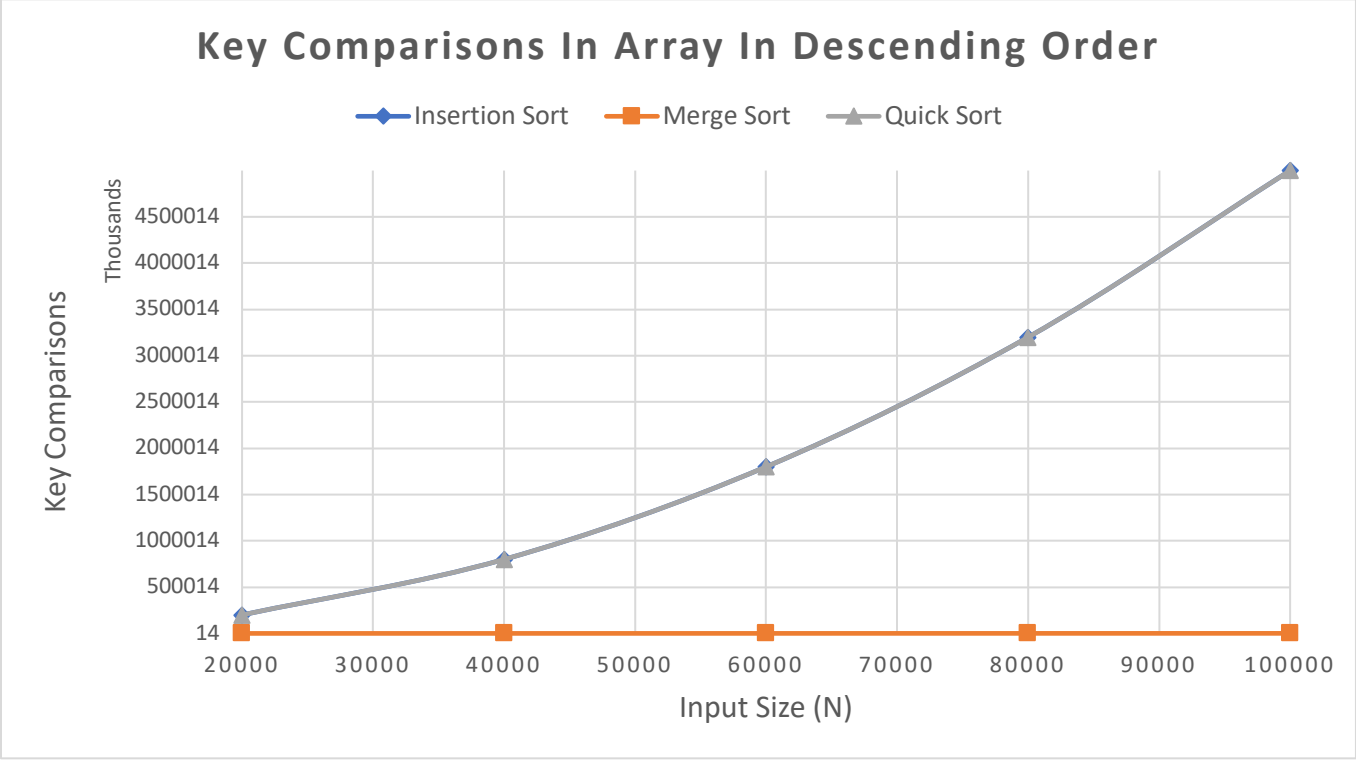
Question 3

Graphs for Array with Random Numbers:

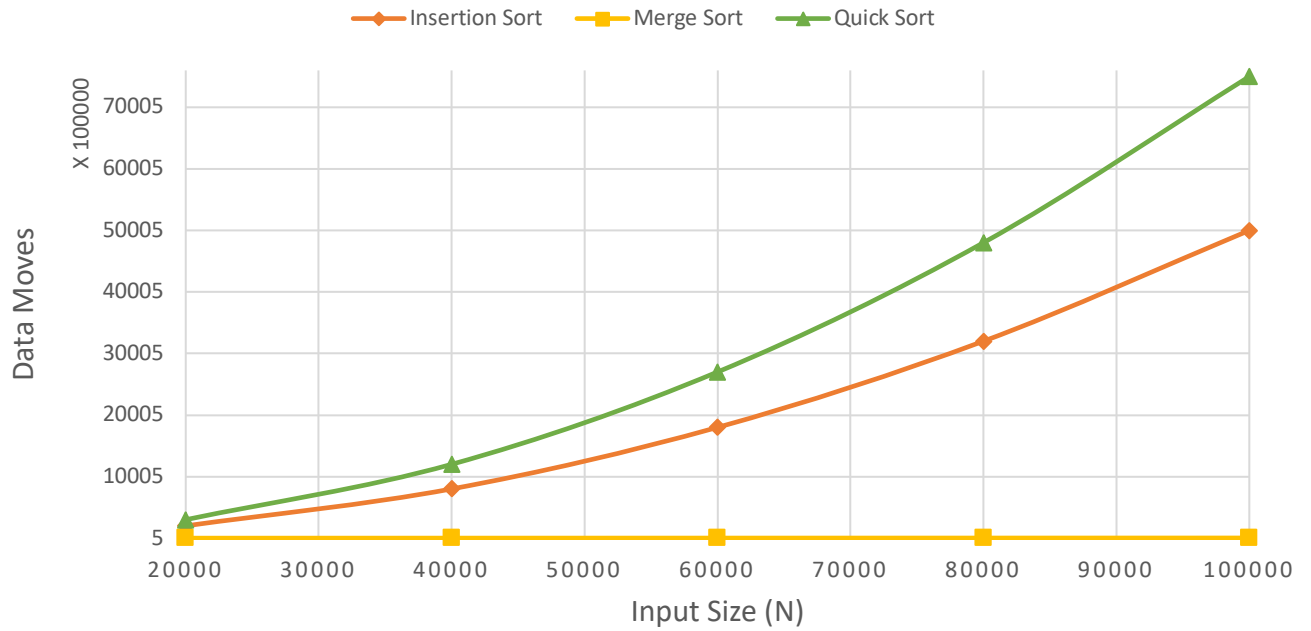




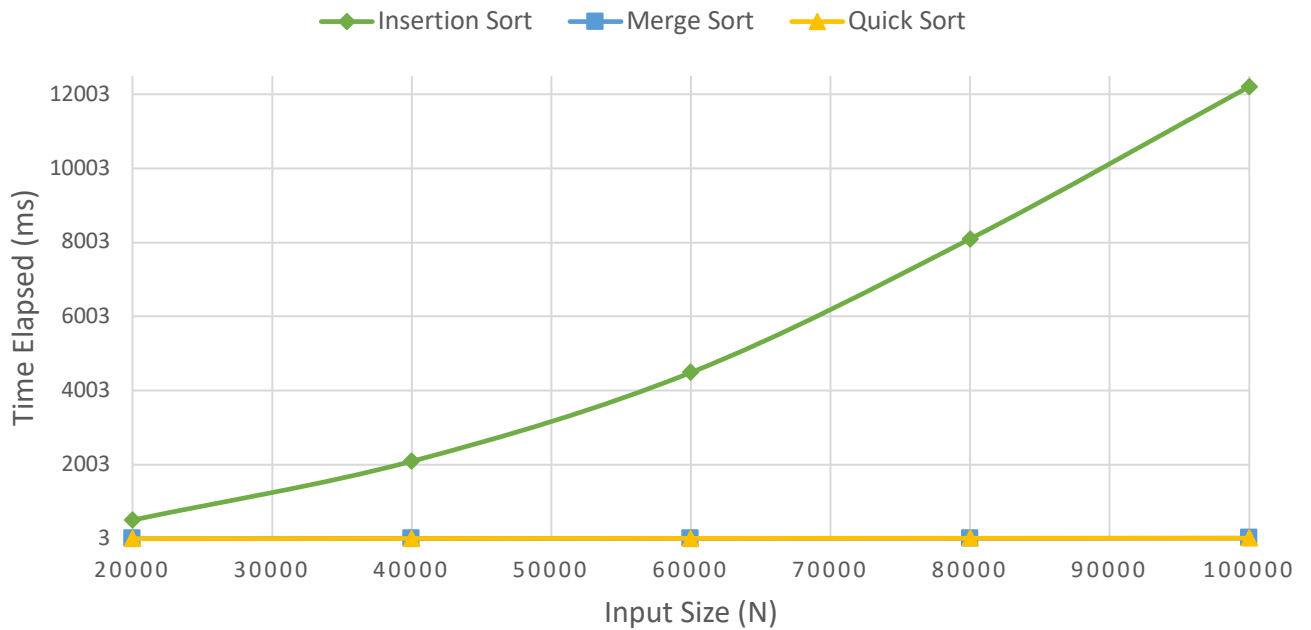
Graphs for Array Sorted in Descending Order:



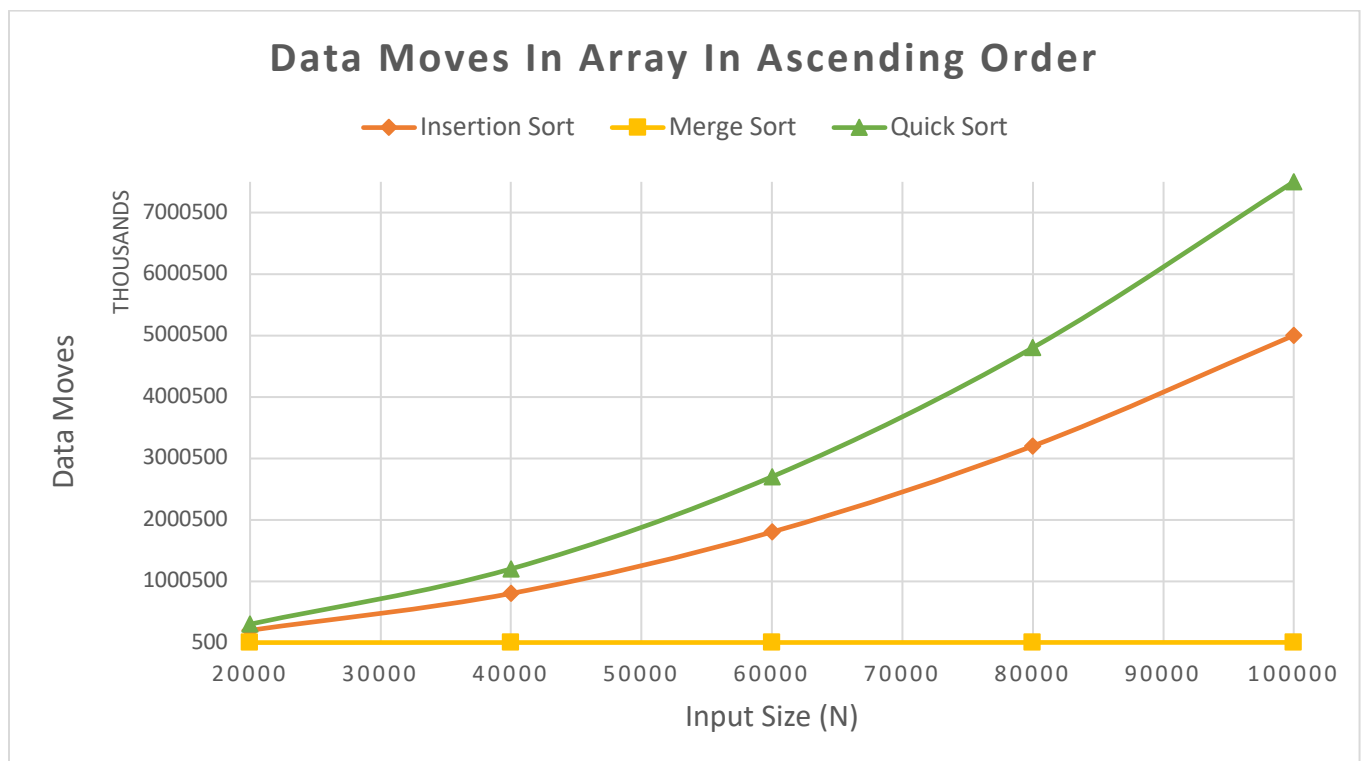
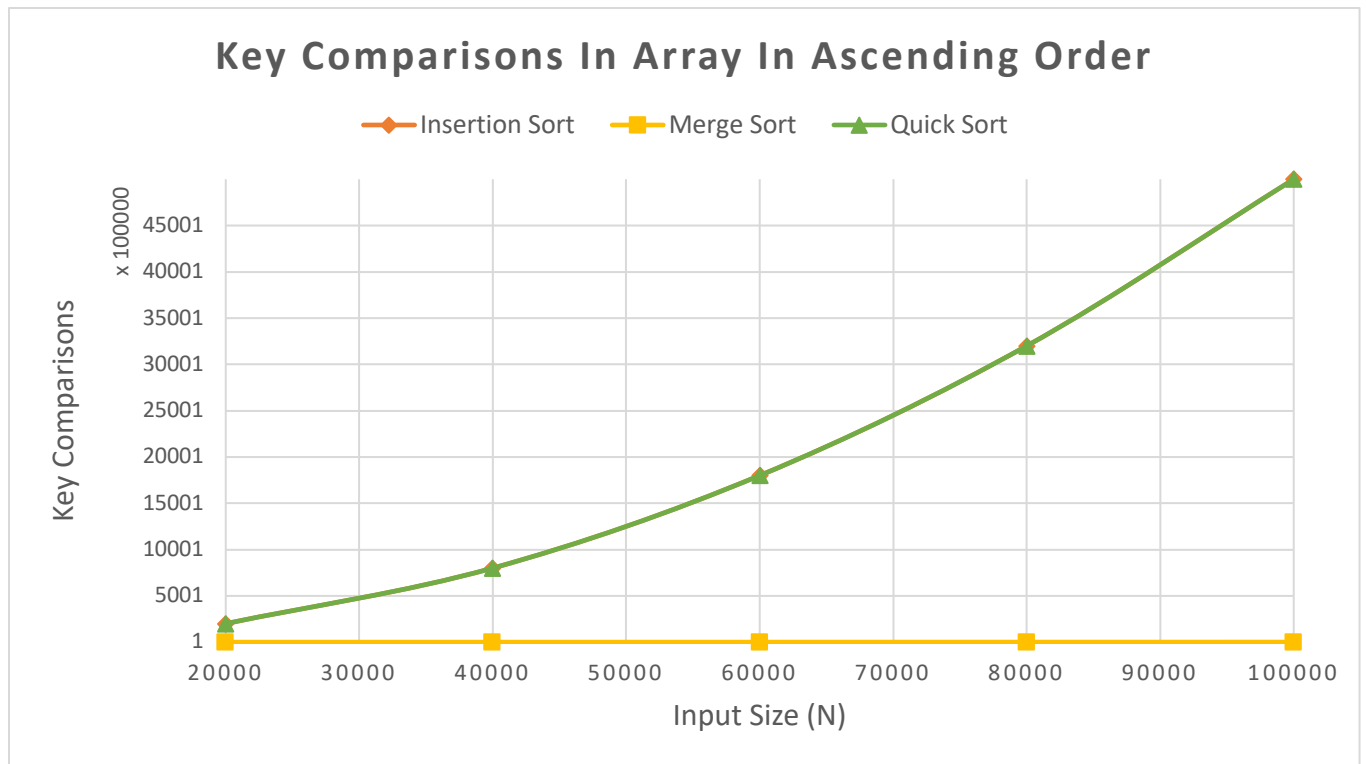
Data Moves In Array In Descending Order



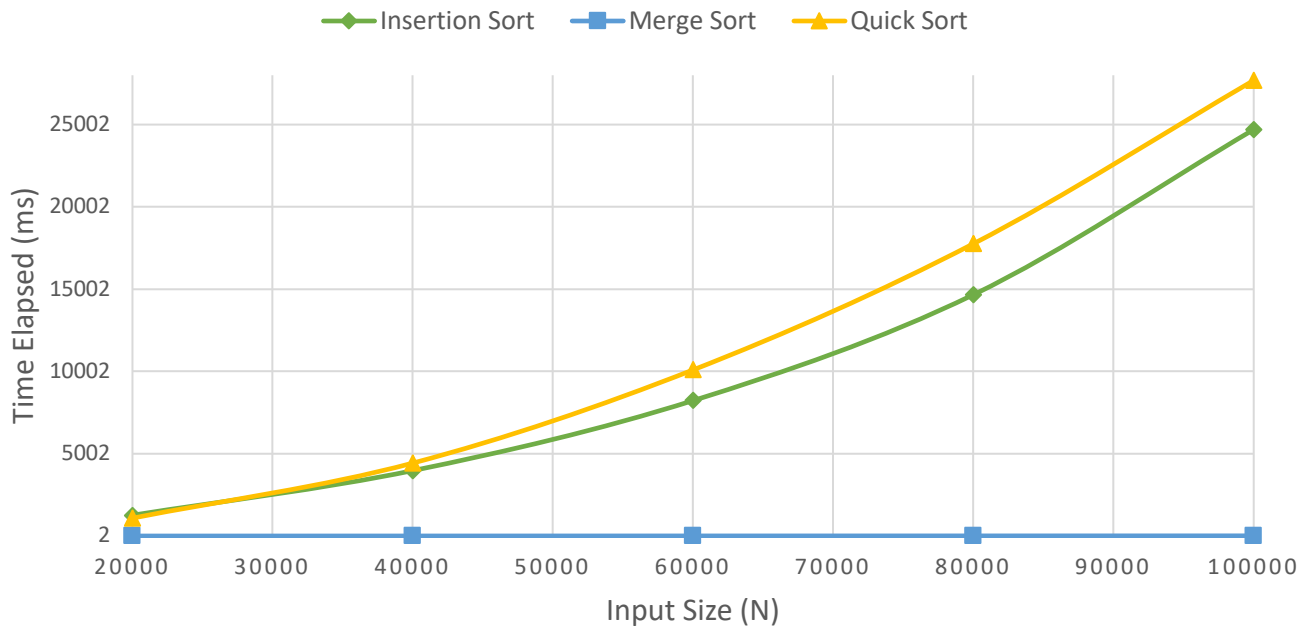
Time Elapsed In Array In Descending Order



Graphs for Array Sorted in Ascending Order:



Time Elapsed In Array In Ascending Order



Question 4

In case of random numbers, in all three graphs (key comparisons, data moves and time elapsed), insertion sort corresponds to a time complexity of $O(n^2)$ which is the same as theoretical results. The parabola formed in all three graphs portrays this relationship. For merge sort and quick sort, the trend showed by the graphs is not entirely obvious. However, the fact that the graph barely rises indicates a logarithmic trend. Theoretically, the average time complexity for both sorting algorithms is $O(n \log n)$ which is not easy to observe from the graphs. This is due to the fact that only 5 set of inputs were used. However, compared to the rise seen by insertion sort, we can infer that merge sort and quick sort would have followed a logarithmic pattern. Using a wider range of input sizes would have resulted in a more accurate estimate of the time complexities. Hence, for a random array, merge sort and quick sort are both highly efficient, followed by the insertion sort which has a much larger time complexity.

For arrays sorted in descending order, for insertion sort all three graphs are the same as before and portray a relationship of $O(n^2)$ which is the same as theoretical results. For quick sort, since the array is already sorted, all three graphs suggest $O(n^2)$ as well, which matches theoretical results and is one of the worst cases for this algorithm since each partition will take place at the last element of the array and divide the list of size n into two sublists of 0 and $n-1$. In the case of merge sort, again, the graph hardly rises as compared to the sharp increase of insertion and quick sort. However, theoretically merge sort exhibits the same complexity of $O(n \log n)$ regardless of whether the array is sorted. This can be inferred from the comparison of the three, in case more input sizes were used, the graph would have followed a logarithmic trend. Thus merge sort is the fastest (although it has a large

number of data moves and takes more space than the other two), followed by quick sort and insertion sort.

In case of arrays sorted in ascending order, the theoretical time complexity of insertion sort in case of array sorted in ascending order is $O(n)$. This is because the inner loop which comprises of n^2 moves is not executed. This can be seen in graphs of data moves and time elapsed. Although the curve is somewhat curved towards the end, it goes on to be linear as the input size increases. This is because of the small number of values used to graph this trend. Moreover, as a comparison to quick sort which is clearly $O(n^2)$, it can be inferred that insertion sort is $O(n)$. As mentioned earlier, quick sort portrays $O(n^2)$ which matches its theoretical time complexity. This is another worst case for quick sort as each partition will take place on the first element of the array. As for merge sort, the trend is not easy to distinguish from the graph since it barely rises, however the theoretical time analysis for it is $O(n \log n)$. This would have been easier to observe if more input sizes had been taken. Conclusively, insertion sort is most efficient in this case with the lowest time complexity, followed by merge sort, and then quicksort.

Tables Used for Question 3:

Tables for Array with Random Numbers

Key Comparisons

Input Size	Insertion Sort	Merge Sort	Quick Sort
20000	99771740	191384	339886
40000	398365705	411184	723970
60000	903777111	627649	1121072
80000	1604907787	886049	1489587
100000	2506060618	1116380	1899122

Data Moves

Input Size	Insertion Sort	Merge Sort	Quick Sort
20000	99811727	574464	554483
40000	398445689	1228928	1218297
60000	903897096	1908928	1901965
80000	1605067772	2617856	2441236
100000	2506260603	3337856	2971305

Time Elapsed (ms)

Input Size	Insertion Sort	Merge Sort	Quick Sort
20000	505.482	3.027	3.31
40000	2092.86	8.916	9.389
60000	4490.15	10.573	12.662
80000	8098.24	12.638	16.735
100000	12210.5	17.601	17.815

Tables for Array in Descending Order

Key Comparisons

Input Size	Insertion Sort	Merge Sort	Quick Sort
20000	199990000	139216	199990000
40000	799980000	298432	799980000
60000	1799970000	469008	1799970000
80000	3199960000	636864	3199960000
100000	4999950000	815024	4999950000

Data Moves

Input Size	Insertion Sort	Merge Sort	Quick Sort
20000	200029998	574464	300079996
40000	800059998	1228928	1200159996
60000	1800089998	1908928	2700239996
80000	3200119998	2617856	4800319996
100000	5000149998	3337856	7500399996

Time Elapsed (ms)

Input Size	Insertion Sort	Merge Sort	Quick Sort
20000	1264.9	3.544	1086.98
40000	3974.7	5.507	4433.08
60000	8229.6	8.198	10095.4
80000	14664.6	10.504	17761.5
100000	24701.8	13.806	27687.8

Tables for Array in Ascending Order

Key Comparisons

Input Size	Insertion Sort	Merge Sort	Quick Sort
20000	19999	148016	199990000
40000	39999	316032	799980000
60000	59999	485456	1799970000
80000	79999	672064	3199960000
100000	99999	853904	4999950000

Data Moves

Input Size	Insertion Sort	Merge Sort	Quick Sort
20000	39998	574464	79996
40000	79998	1228928	159996
60000	119998	1908928	239996
80000	159998	2617856	319996
100000	199998	3337856	399996

Time Elapsed (ms)

Input Size	Insertion Sort	Merge Sort	Quick Sort
20000	0.12	2.509	563.826
40000	0.34	5.339	2308.48
60000	0.37	9.794	5140.89
80000	0.59	12.864	9022.57
100000	0.76	16.383	13549.20