Question 1

What system did you run your tests on? Describe the relevant specifications. What factors might have affected the timing results?

System:
I used Eclipse workspace.
Computer with the following specifications:
Intel(R) Core(TM) i5-8365U CPU @ 1.60GHz 1.90 GHz
RAM: 8GB
Windows 10

Factors

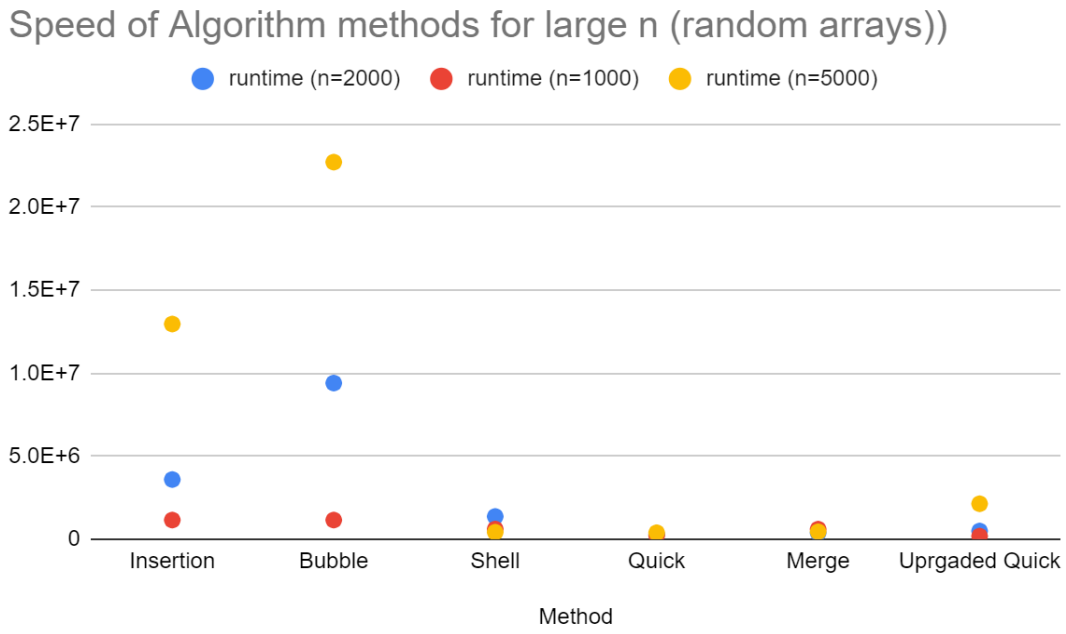
1. Cpu speed, power of the computer
2. Size of the input data
3. Number of running processes
4. Memory availability

Question 2

Which method(s) is (are) faster for large n? How does each method scale with n for random input? Show your results either in a table or with a plot.

Quick sort, upgraded quick sort and merge sort algorithms show high efficiency in terms runtime for large n values (for random arrays)

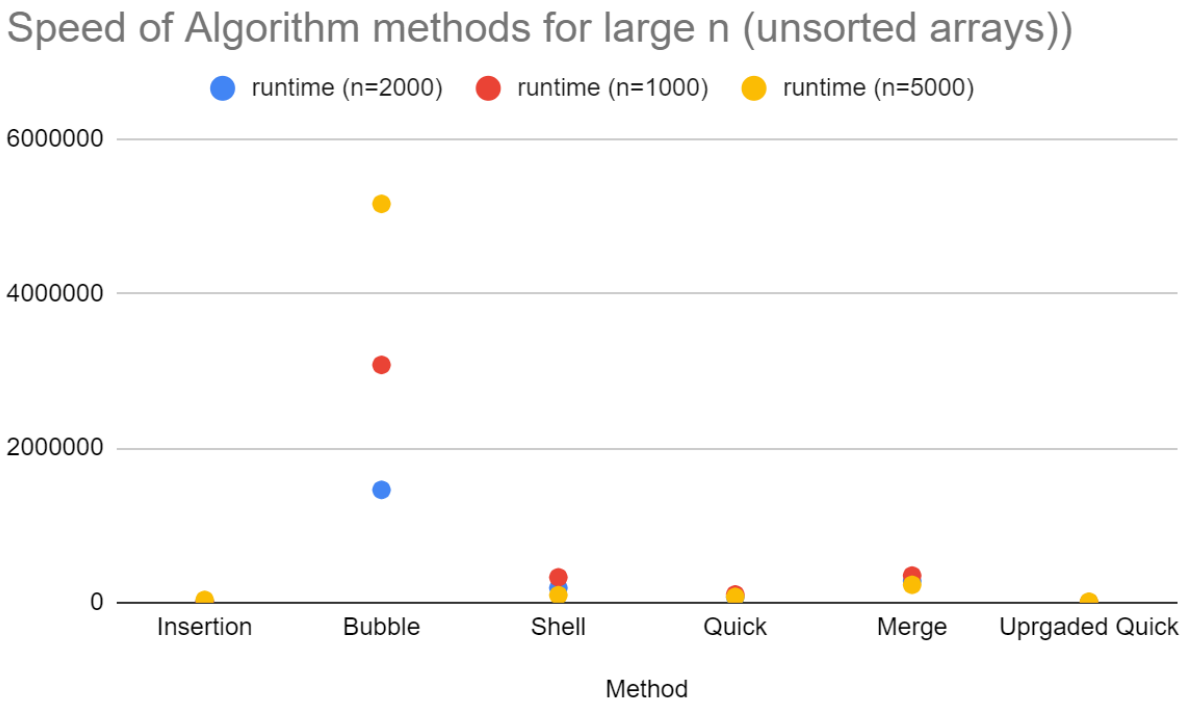
Insertion and bubble sorts provide poor speed efficiency especially for random and unsorted
Shell sort provides better speed efficiency than insertion and bubble sort
(Attached tables and graphs for large n (n=1000,2000 and 5000) with comments)



From the graph, Upgraded quick sort, merge sort and quicksort are showing efficiency for random arrays.
Insertion sort and bubble sort are showing poor efficiency for random arrays
Shell sort is showing better efficiency than insertion and bubble sort

Speed of algorithm methods for large n (random arrays)

Method	runtime (n=2000)	runtime (n=1000)	n=5000
Insertion	3595701	1154900	12962500
Bubble	9405001	1154900	22713500
Shell	1370800	613100	423200
Quick	293300	218300	391300
Merge	376300	607100	448700
Upgraded Quick	494800	186300	2136600

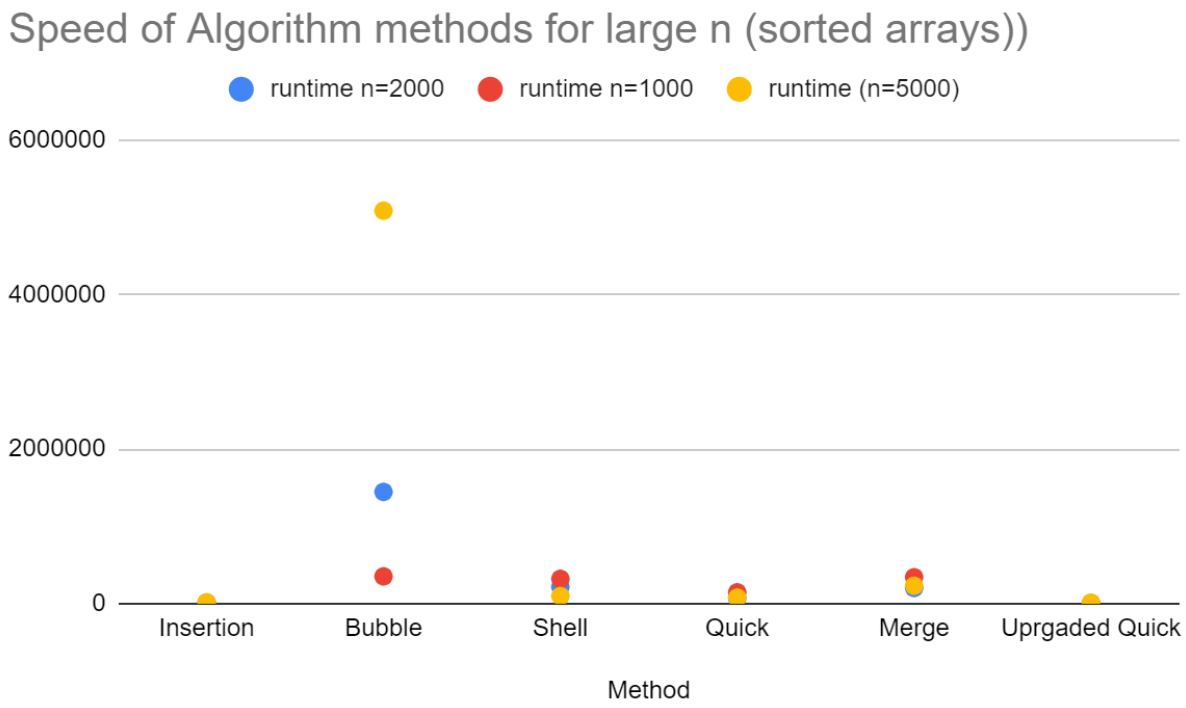


From the graph, Upgraded quick sort, merge sort and quicksort and insertion are showing efficiency for unsorted arrays.
bubble sort is showing poor efficiency for unsorted arrays
Shell sort is showing better efficiency than insertion bubble sort

Speed of Algorithm methods for large n (unsorted arrays))

Method	runtime (n=2000)	runtime (n=1000)	runtime n=5000
Insertion	19000	7400	35900
Bubble	1459601	3076200	5159700
Shell	190900	328100	98000
Quick	72400	106800	75800

Merge	279800	350700	231300
Upgraded Quick	5000	3900	12900



From the graph, Upgraded quick sort, insertion and quick sort are showing efficiency for sorted arrays.
bubble sort is showing poor efficiency for random arrays
Shell sort and merge sort are showing better efficiency than bubble sort

Speed of Algorithm methods for large n (sorted arrays))

Method	runtime n=2000	runtime n=1000	runtime n=5000
Insertion	14599	7500	17000
Bubble	1446200	353500	5084900

Shell	217100	321600	99000
Quick	70100	147000	79000
Merge	203500	342200	231400
Upgraded Quick	7000	5500	12800

Question 3 :
Is your answer to the previous question consistent with the theory? Explain.

The results are consistent with the theory.
For large k, quick sort, merge and upgraded quicksort are showing high efficiency for random, unsorted and sorted arrays
For large k (random arrays), insertion sort and bubble sort are showing low efficiency

Insertion sort shows efficiency for sorted or almost sorted arrays
And bubble sort shows low efficiency for random, unsorted and sorted arrays of large k
Shell sort is showing efficiency for arrays of large k
(check the comments on each graph from the previous question)

Question 4:
Is insertion sort ever preferable? Your answer should include examples with benchmarks.

Insertion sort is preferable for small arrays and arrays that are sorted or almost sorted as it showed high speed efficiency
From the benchmark table, it's noticed how the insertion method performs poorly for random arrays of large n but it shows efficiency for small arrays and arrays that are unsorted and sorted. (attached proof with the benchmark table)

n	random	unsorted	sorted
10	1700	1100	1000
20	3700	1100	1300
50	19100	2100	2300

100	71900	3500	3400
200	322400	7900	7900
500	431000	7000	6800
1000	1154900	7400	7500
2000	3595701	19000	14599
5000	12962500	35900	17000

Question 5:
Are there input cases where one method performs poorly? Again, show examples with benchmarks.

For random arrays (of large n), insertion method performed poorly
For n = 5000 the runtime was 12962500
For n = 2000 the runtime was 3595701
For n= 1000 the runtime was 1154900

For arrays of small n, merge sort performed poorly specifically for sorted and unsorted arrays compared with others methods such as insertion method which shows high speed efficiency for small arrays speciality the unsorted and sorted arrays
(merge sort for n = 10 and 20)

n	random	unsorted	sorted
10	7400	6400	6100
20	14700	11400	27100

Insertion sort for n =10 and 20

n	random	unsorted	sorted
10	1700	1100	1000
20	3700	1100	1300

Question 6

- **Make a table to show for each of the input sizes (n = 10,20 ...) and input type (random, sorted, reverse-sorted) which sorting algorithm is optimal and how much time they take.**

n	Random arrays	Unsorted arrays	Sorted arrays
10	Insertion sort Runtime: 1700	Insertion sort Runtime: 1100	Insertion sort Runtime: 1100
20	Insertion sort Runtime: 3700	Insertion sort Runtime: 1100	Insertion sort Runtime: 1300
50	Quick sort Runtime: 15300	Insertion sort Runtime: 2100	Insertion sort Runtime: 2100
100	Shell sort Runtime: 35500	Insertion sort Runtime: 3500	Quick sort Runtime: 15300
200	Shell sort Runtime: 132800	Insertion sort Runtime: 7900	Insertion sort Runtime: 7900
500	Quick sort Runtime : 117500	Insertion sort Runtime: 7000	Upgraded quick sort Runtime: 6700
1000	Upgraded quick sort Runtime: 186300	Upgraded quick sort Runtime: 3900	Upgraded quick sort Runtime: 5500
2000	quick sort Runtime: 293300	Upgraded quick sort Runtime: 5000	Upgraded quick sort Runtime: 7000
5000	Quick sort Runtime: 391300	Upgraded quick sort Runtime: 12900	Upgraded quick sort Runtime: 12800