Ömer Mesud TOKER

21302479 – Section 1

HW 1

Question 1)

a) Insertion Sort

Initial array: [9, 8, 2, 5, 7, 4] => key = 8

[9, 9, 2, 5, 7, 4] => Shifting until find smaller than 8 or beginning of array

[8, 9, 2, 5, 7, 4] => Insert 8; key = 2

[8, 8, 9, 5, 7, 4] => Shifting until find smaller than 2 or beginning of array

[2, 8, 9, 5, 7, 4] => Insert 2; key = 5

[2, 8, 8, 9, 7, 4] => Shifting until find smaller than 5 or beginning of array

[2, 5, 8, 9, 7, 4] => Insert 5; key = 7

[2, 5, 8, 8, 9, 4] => Shifting until find smaller than 7 or beginning of array

[2, 5, 7, 8, 9, 4] => Insert 7; key = 4

[2, 5, 5, 7, 8, 9] => Shifting until find smaller than 4 or beginning of array

[2, 4, 5, 7, 8, 9] => Insert 4; the end

b) Selection Sort

Initial array: [9, 8, 2, 5, 7, 4] => Find largest in unsorted part, which is 9

[4, 8, 2, 5, 7, 9] => Swap 9 with the end of the unsorted part, which is 4

[4, 8, 2, 5, 7, 9] => Find largest in unsorted part, which is 8

[4, 7, 2, 5, 8, 9] => Swap 8 with the end of the unsorted part, which is 7

[4, 7, 2, 5, 8, 9] => Find largest in unsorted part, which is 7

[4, 5, 2, 7, 8, 9] => Swap 7 with the end of the unsorted part, which is 5

[4, 5, 2, 7, 8, 9] => Find largest in unsorted part, which is 5

[4, 2, 5, 7, 8, 9] => Swap 5 with the end of the unsorted part, which is 2

[4, 2, 5, 7, 8, 9] => Find largest in unsorted part, which is 4

[2, 4, 5, 7, 8, 9] => Swap 4 with the end of the unsorted part, which is 2, the end

c) Bubble Sort

Initial array: [9, 8, 2, 5, 7, 4] => start from index 0

[8, 9, 2, 5, 7, 4] => swap 9 with 8

[8, 2, 9, 5, 7, 4] => swap 9 with 2

[8, 2, 5, 9, 7, 4] => swap 9 with 5

[8, 2, 5, 7, 9, 4] => swap 9 with 7

[8, 2, 5, 7, 4, 9] => swap 9 with 4

[8, 2, 5, 7, 4, 9] => start from index 0

[2, 8, 5, 7, 4, 9] => swap 8 with 2

[2, 5, 8, 7, 4, 9] => swap 8 with 5

[2, 5, 7, 8, 4, 9] => swap 8 with 7

[2, 5, 7, 4, 8, 9] => swap 8 with 4

[2, 5, 7, 4, 8, 9] => start from index 0

[2, 5, 4, 7, 8, 9] => swap 7 with 4

[2, 5, 4, 7, 8, 9] => start from index 0

[2, 4, 5, 7, 8, 9] => swap 5 with 4

[2, 4, 5, 7, 8, 9] => sorted is true, the end

d) Merge Sort

Initial array: [9, 8, 2, 5, 7, 4] => divide

[9, 8, 2] - [5, 7, 4] => divide

[9, 8] - [2] -- [5, 7] - [4] => divide

[9] - [8] -- [2] ----- [5] - [7] -- [4] => start to merge

[8, 9] - [2] -- [5, 7] - [4] => merge

[2, 8, 9] - [4, 5, 7] => merge

[2, 4, 5, 7, 8, 9] => copy the temp array back into original array; the end

Function Calls:

mergesort (arr, 0, 5)

mergesort (arr, 0, 2)

mergesort (arr, 0, 1)

mergesort (arr, 0, 0)

mergesort (arr, 1, 1)

merge (arr, 0, 0, 1)

mergesort (arr, 2, 2)

merge (arr, 0, 1, 2)

mergesort (arr, 3, 5)

mergesort (arr, 3, 4)

mergesort (arr, 3, 3)

mergesort (arr, 4, 4)

merge (arr, 3, 3, 4)

mergesort (arr, 5, 5)

merge (arr, 3, 4, 5)

merge (arr, 0, 2, 5)

e) Quick Sort

Initial array: [9, 8, 2, 5, 7, 4] => pivot = 4, put 4 into first place

[4, 8, 2, 5, 7, 9] => first unknown is 8, which is greater than 4, S2 starts with 8

[4, 8, 2, 5, 7, 9] =>2 is smaller so swap it to increase S1

[4, 2, 8, 5, 7, 9] => 5, 7 and 9 are bigger, no swap

[2, 4, 8, 5, 7, 9] => put 4 between S1 and S2

Sort S1, which has only 2 so it is already sorted

Sort S2

[8, 5, 7, 9] => pivot = 9, put 9 into first place

[9, 5, 7, 8] => first unknown is 5, which is smaller than 9, S1 starts with 5

[9, 5, 7, 8] => 7 and 8 are smaller, no swap

[8, 5, 7, 9] => put 9 between S1 and S2 (S2 has no elements)

Sort S1

[8, 5, 7] => pivot = 7, put 7 into first place

[7, 5, 8] => first unknown is 5, which is smaller than 7, S1 starts with 5

[7, 5, 8] => 8 is greater, no swap

[5, 7, 8] => put 7 between S1 and S2

Sort S1, which has only 5 so it is already sorted

Sort S2, which has only 8 so it is already sorted

[5, 7, 8, 9] => combining

[2, 4, 5, 7, 8, 9] => combining

Function Calls:

quicksort (array, 0, 5)

partition (array, 0, 5, ?)

quicksort (array, 0, 0)

quicksort (array, 2, 5)

partition (array, 2, 5, 1)

quicksort (array, 2, 4)

partition (array, 2, 4, 5)

quicksort (array, 2, 2)

quicksort (array, 5, 5)

quicksort (array, 6, 5)

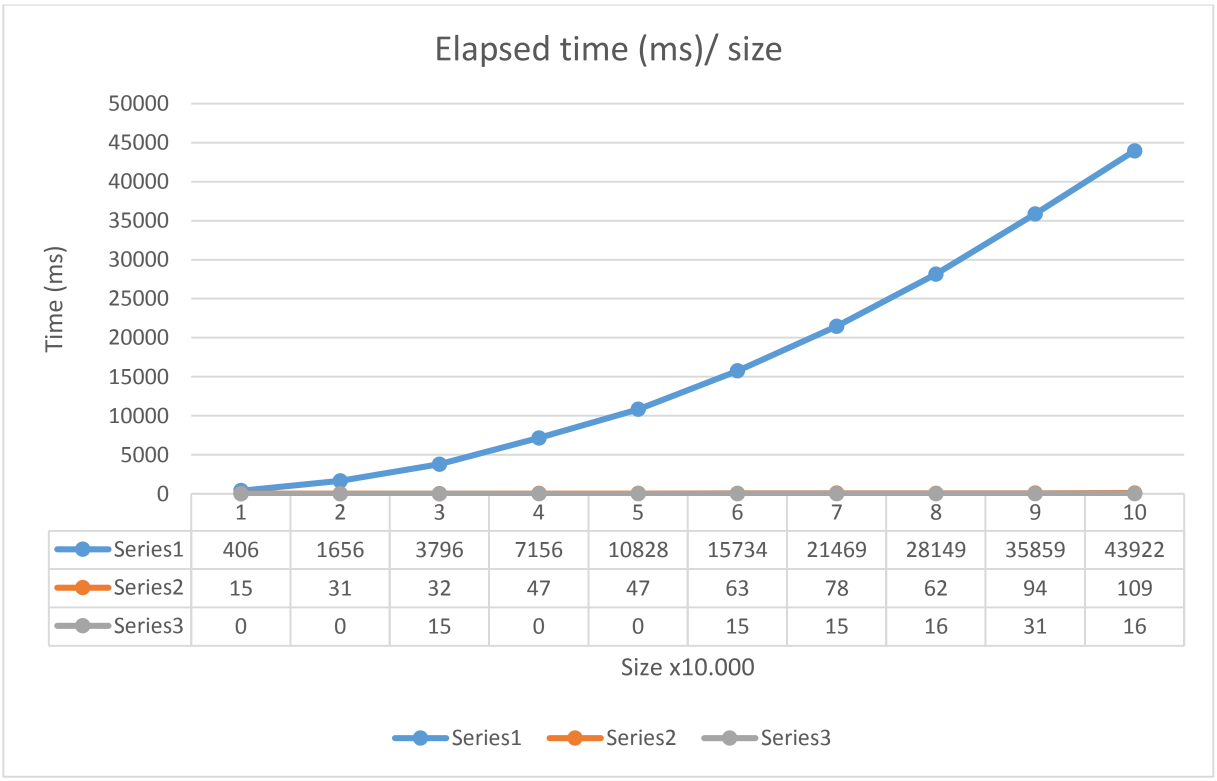
Question 2)

PART 1

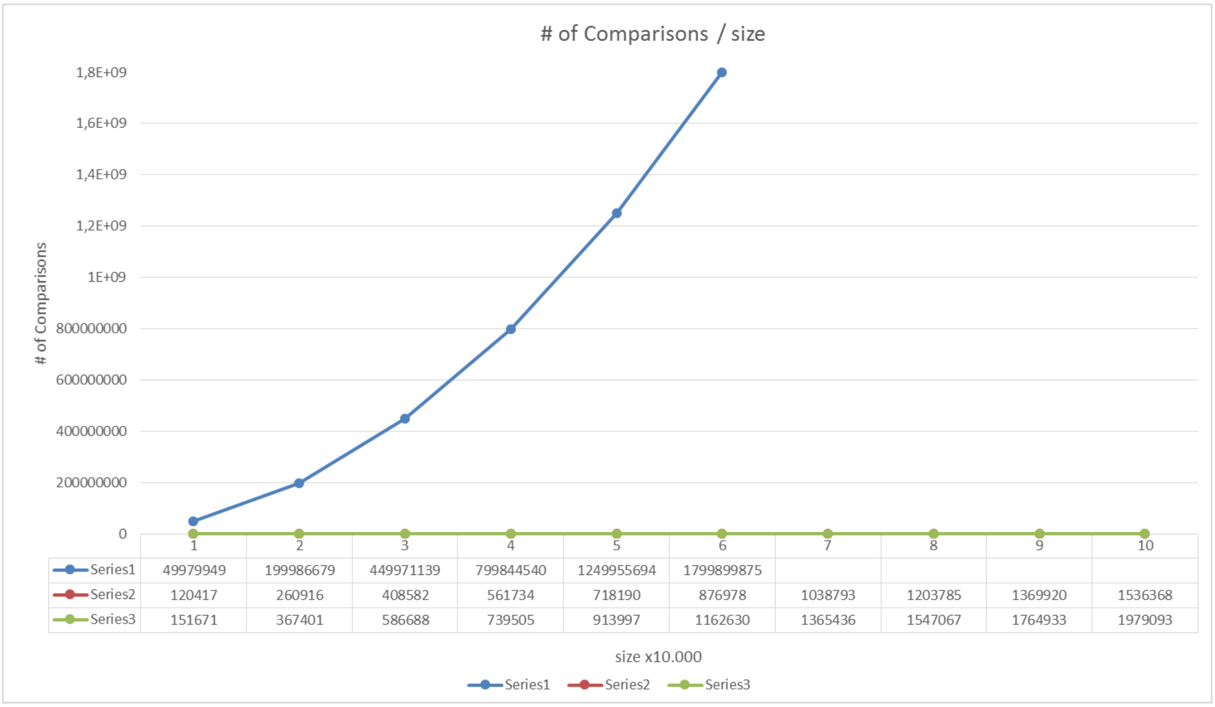
Series 1: Bubble sort

Series 2: Merge Sort

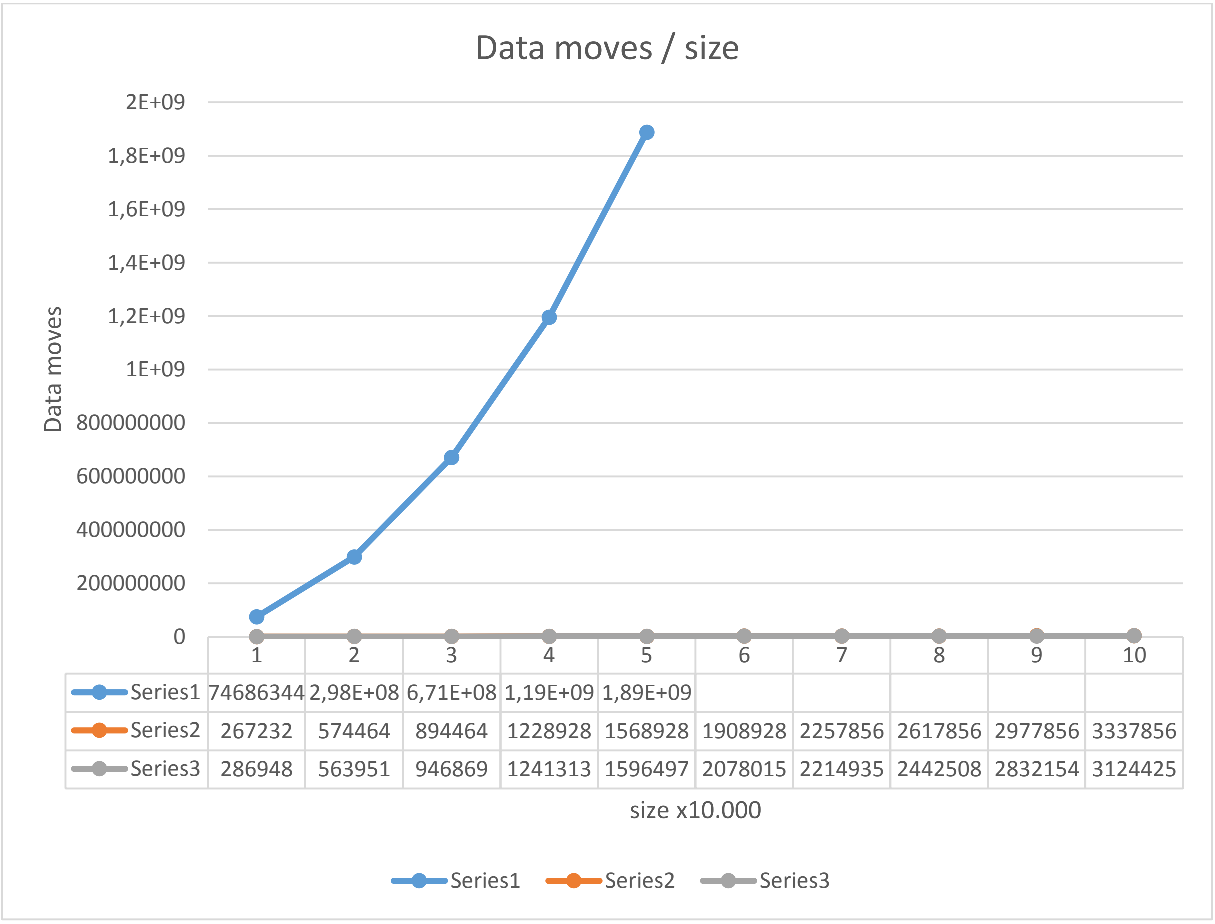
Series 3: Quick sort



	1	2	3	4	5	6	7	8	9	10
Series1	406	1656	3796	7156	10828	15734	21469	28149	35859	43922
Series2	15	31	32	47	47	63	78	62	94	109
Series3	0	0	15	0	0	15	15	16	31	16



	1	2	3	4	5	6	7	8	9	10
Series1	49979949	199986679	449971139	799844540	1249955694	1799899875				
Series2	120417	260916	408582	561734	718190	876978	1038793	1203785	1369920	1536368
Series3	151671	367401	586688	739505	913997	1162630	1365436	1547067	1764933	1979093



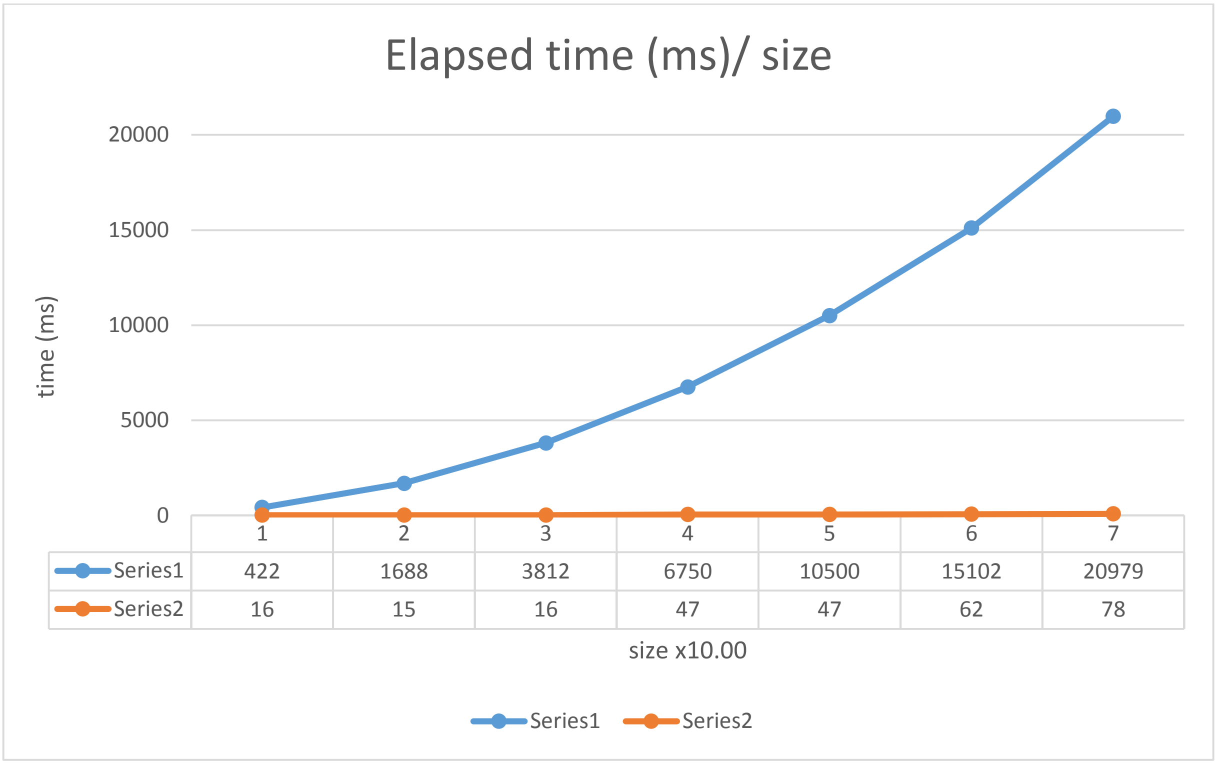
	1	2	3	4	5	6	7	8	9	10
Series1	74686344	2,98E+08	6,71E+08	1,19E+09	1,89E+09					
Series2	267232	574464	894464	1228928	1568928	1908928	2257856	2617856	2977856	3337856
Series3	286948	563951	946869	1241313	1596497	2078015	2214935	2442508	2832154	3124425

PART 2

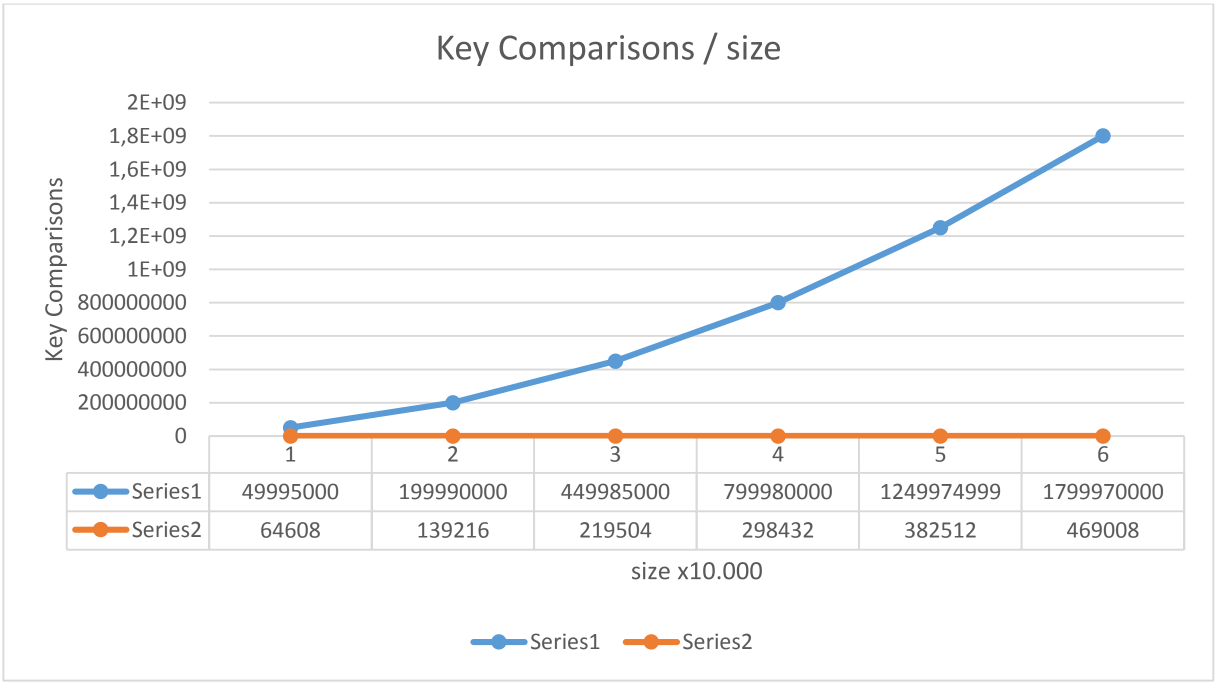
Series 1: Bubble Sort

Series 2: Merge Sort

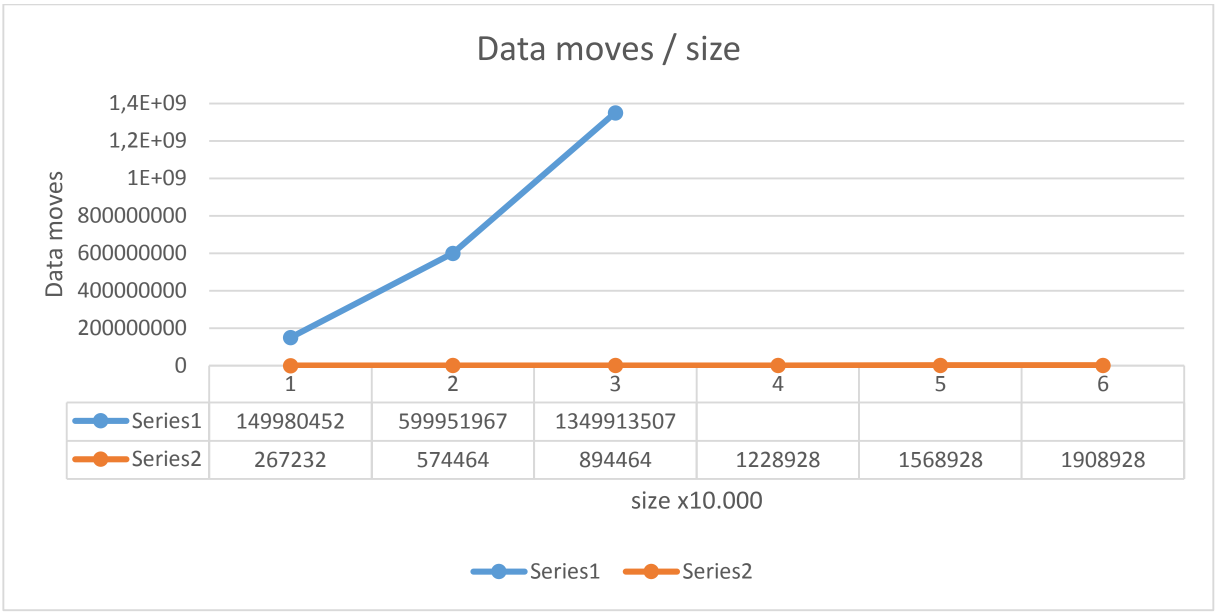
Note: For this part quick sort is not included in the graph since it fails after size 10000.



	1	2	3	4	5	6	7
Series1	422	1688	3812	6750	10500	15102	20979
Series2	16	15	16	47	47	62	78



	1	2	3	4	5	6
Series1	49995000	199990000	449985000	799980000	1249974999	1799970000
Series2	64608	139216	219504	298432	382512	469008



	1	2	3	4	5	6
Series1	149980452	599951967	1349913507			
Series2	267232	574464	894464	1228928	1568928	1908928

PART 3

Note: For this part quick sort is not included in the graph since it fails after size 10000.

Elapsed time for bubble sort is 0, number of comparisons are size - 1, and number of data moves are 0. It is best case for bubble sort, which is O(n).

Number of data moves for merge sort is the same as part 1 and 2. Therefore there is no need to plot the graphs.

Question 3)

If algorithms use randomly generated numbers (average cases), they are closer to the theoretic results and quick sort seems to be best algorithm in those 3. For bubble sort worst (descending part) and average (random part) cases are O(n^2) and best (ascending part) case is O(n). For merge sort all situations are the same as we expect in theoretical, which is O(n logn). For quick sort, best and average (random) cases are O(n logn), however, worst (ascending and descending) is O(n^2).

Part 2 and part 3, data moves for merge sort is the same for same sizes. (Actually, number of data moves for merge sort is always same for the same sizes in part 1, 2 and 3.) However, in these parts quick sort fails after size 10000. Ascending or descending orders are worst for quick sort, however for merge sort all of them is O(n log n). Even bubble sort is better than quick sort for these parts.

Part 3, elapsed time for bubble sort is 0, comparisons are size -1, and data moves are 0. It is best case for bubble sort, which is O(n).

Question 4)

1) $\mathrm{n}! = 1.2.3.4 \ldots \mathrm{n} = [1.\mathrm{n}][2(\mathrm{n}-1)][3(\mathrm{n}-2)] \ldots \left[\frac{\mathrm{n}}{2}.\frac{\mathrm{n}}{2}\right] < \left(\frac{\mathrm{n}^2}{4}\right)^{\frac{\mathrm{n}}{2}} = \left(\frac{n}{2}\right)^n$

$$\Rightarrow \log n! < \log\left(\frac{n}{2}\right)^n = n.\log\left(\frac{n}{2}\right) = O(n\ logn)$$

$$\therefore \log n! = O(n \log n)$$

2) It actually demonstrates the merge sort algorithm.

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = 2\left(2T\left(\frac{n}{4}\right) + O\left(\frac{n}{2}\right)\right) + O(n) = 4T\left(\frac{n}{4}\right) + 2O\left(\frac{n}{2}\right) + O(n)$$
$$= 4T\left(\frac{n}{4}\right) + 2O(n) = 4\left(2T\left(\frac{n}{8}\right) + O\left(\frac{n}{4}\right)\right) + 2O(n)$$
$$= 8T\left(\frac{n}{8}\right) + 4O\left(\frac{n}{4}\right) + 2O(n) = 8T\left(\frac{n}{8}\right) + 3O(n) = 2^kT\left(\frac{n}{2^k}\right) + kO(n)$$

$$2^k = n \Rightarrow k = \log n$$

$$T(n) = nT(1) + \log n\, O(n) = O(n) + O(n)\log n = O(n \log n)$$

3) $f(n) = O\big(T(n)\big) \Rightarrow f(n) \leq c.T(n)\ when\ n > n_0$

If we take c = 8 and n_0 = 2;

$$f(n) = 4n^5 + 3n^2 + 1 < 8.T(n) = 8.n^5\ where\ n > 2$$

The condition is satisfied.

$$\therefore f(n) = 4n^5 + 3n^2 + 1 = O(n^5)$$