CS2383 – Fall 2024

Assignment 4 – Sorting Algorithms

Due: Tuesday Oct. 15, 10am (class time)

NO LATE ASSIGNMENT ACCEPTED THIS TIME, as I may discuss the solution in class on that day, in preparation for the midterm.

IMPORTANT: individual work please!

Tasks:

1. Show how the array below would get sorted using each of the following sorting algorithms: selection sort, insertion sort, mergesort, and quicksort (with no initial shuffling of the array). Show what the array looks like at each iteration of the algorithm, when you run it manually.

0	1	2	3	4	5	6	7	8	9
20	10	15	8	29	21	17	3	25	7

2. Compare Mergesort and Quicksort using an empirical analysis (i.e., similar to what you did for Asgn3 Question 4) – both on average over a few executions for each value of N that you decided to use. You can use the code for these algorithms (provided in D2L) as a starting point. Which one is faster? Please submit a printout of your code as well as your experimental runtimes and analysis.

3. If you had to show that Quicksort's worst case is O(N^2) using an empirical analysis, how would you proceed?

Submission: just on paper (no online submission)

Question 1 – Selection sort

0	1	2	3	4	5	6	7	8	9
20	10	15	8	29	21	17	3	25	7

0	1	2	3	4	5	6	7	8	9
2	10	15	8	29	21	17	20	25	7

0	1	2	3	4	5	6	7	8	9
2	7	15	8	29	21	17	20	25	10

0	1	2	3	4	5	6	7	8	9
2	7	8	15	29	21	17	20	25	10

0	1	2	3	4	5	6	7	8	9
2	7	8	10	29	21	17	20	25	15

0	1	2	3	4	5	6	7	8	9
2	7	8	10	15	21	17	20	25	29

0	1	2	3	4	5	6	7	8	9
2	7	8	10	15	17	21	20	25	29

0	1	2	3	4	5	6	7	8	9
2	7	8	10	15	17	20	21	25	29

The last 2 iterations (for 21 and 25) are not changing the array

Question 1 – Insertion sort

0	1	2	3	4	5	6	7	8	9
20	10	15	8	29	21	17	3	25	7

0	1	2	3	4	5	6	7	8	9
10	20	15	8	29	21	17	3	25	7

0	1	2	3	4	5	6	7	8	9
10	15	20	8	29	21	17	3	25	7

0	1	2	3	4	5	6	7	8	9
8	10	15	20	29	21	17	3	25	7

0	1	2	3	4	5	6	7	8	9
8	10	15	20	29	21	17	3	25	7

No change for 29

0	1	2	3	4	5	6	7	8	9
8	10	15	20	21	29	17	3	25	7

0	1	2	3	4	5	6	7	8	9
8	10	15	17	20	21	29	3	25	7

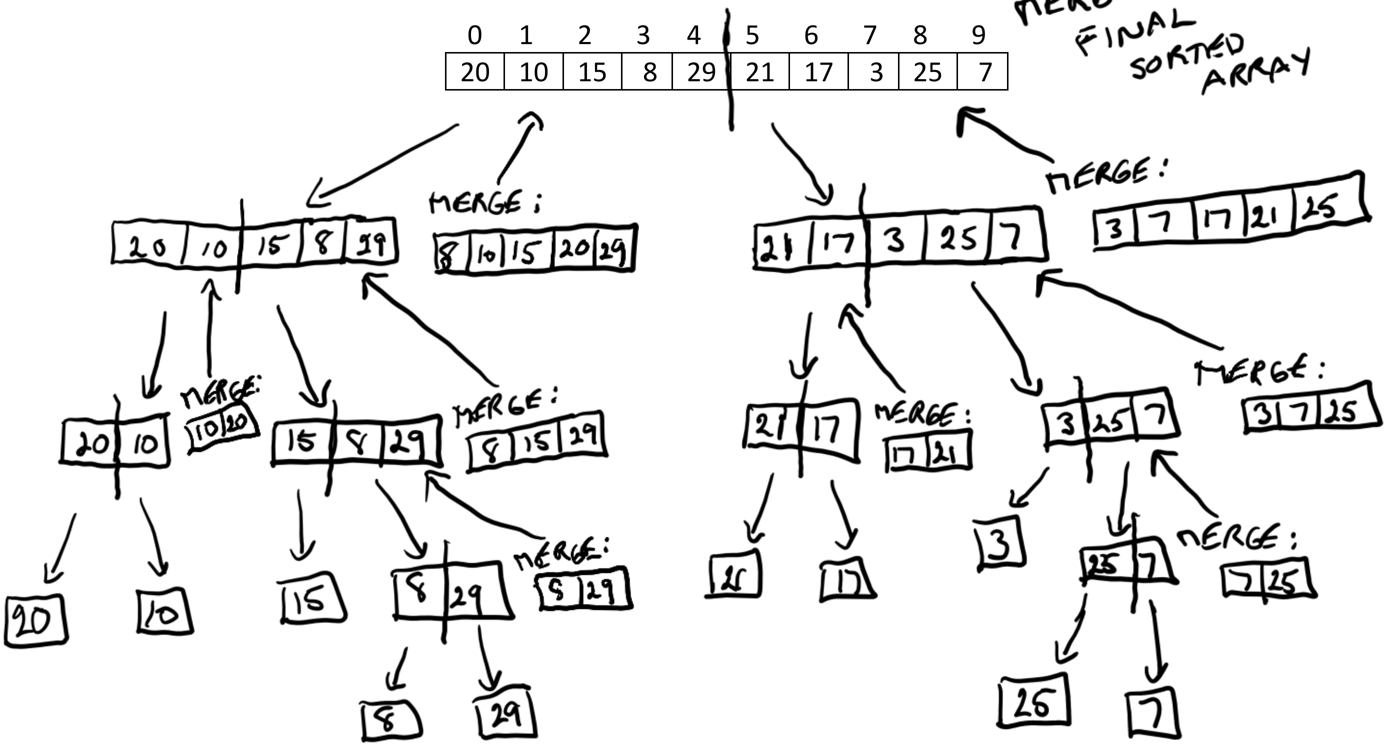
0	1	2	3	4	5	6	7	8	9
3	8	10	15	17	20	21	29	25	7

0	1	2	3	4	5	6	7	8	9
3	8	10	15	17	20	21	25	29	7

0	1	2	3	4	5	6	7	8	9
3	7	8	10	15	17	20	21	25	29

Question 1 – Mergesort

0	1	2	3	4	5	6	7	8	9
20	10	15	8	29	21	17	3	25	7



Question 1 – Quicksort

0	1	2	3	4	5	6	7	8	9
20	10	15	8	29	21	17	3	25	7
PIVOT									

0	1	2	3	4	5	6	7	8	9
20	10	15	8	29	21	17	3	25	7
				i					j

0	1	2	3	4	5	6	7	8	9
20	10	15	8	7	21	17	3	25	7
					i		j		

0	1	2	3	4	5	6	7	8	9
20	10	15	8	7	3	17	21	25	29
						j	i		

0	1	2	3	4	5	6	7	8	9
17	10	15	8	7	3	20	21	25	29
PIVOT							PIVOT		

0	1	2	3	4	5	6	7	8	9	
17	10	15	8	7	3	20	21	25	29	
					j	i			j	i

0	1	2	3	4	5	6	7	8	9
3	10	15	8	7	17	20	21	25	29
PIVOT								PIVOT	

0	1	2	3	4	5	6	7	8	9	
3	10	15	8	7	17	20	21	25	29	
				j	i				j	i

0	1	2	3	4	5	6	7	8	9
3	10	15	8	7	17	20	21	25	29
	PIVOT								PIVOT — BUT DONE

0	1	2	3	4	5	6	7	8	9
3	10	15	8	7	17	20	21	25	29
		i		j					

0	1	2	3	4	5	6	7	8	9
3	10	7	8	15	17	20	21	25	29
			j	i					

0	1	2	3	4	5	6	7	8	9
3	8	7	10	15	17	20	21	25	29
	PIVOT			PIVOT BUT DONE					

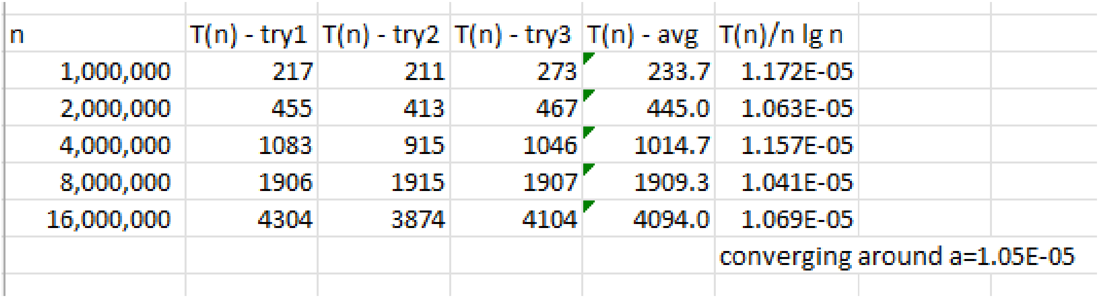
0	1	2	3	4	5	6	7	8	9
3	8	7	10	15	17	20	21	25	29
		j	i						

0	1	2	3	4	5	6	7	8	9
3	7	8	10	15	17	20	21	25	29

Question 2

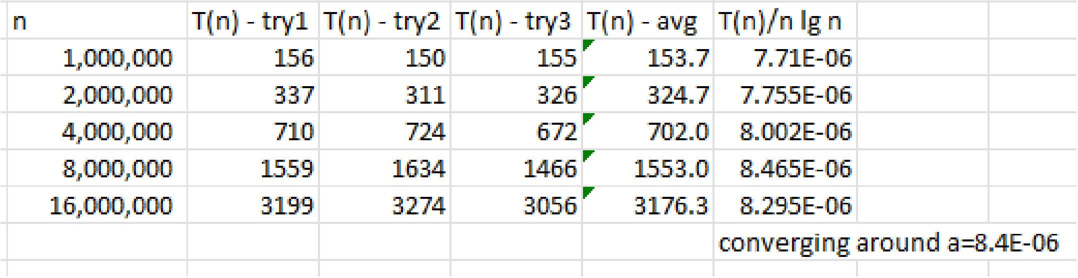
Important: each run should use the exact same array as starting point for both algorithms, otherwise it would be hard to really make a comparison (see my code).

For Mergesort:



n	T(n) - try1	T(n) - try2	T(n) - try3	T(n) - avg	T(n)/n lg n
1,000,000	217	211	273	233.7	1.172E-05
2,000,000	455	413	467	445.0	1.063E-05
4,000,000	1083	915	1046	1014.7	1.157E-05
8,000,000	1906	1915	1907	1909.3	1.041E-05
16,000,000	4304	3874	4104	4094.0	1.069E-05
					converging around a=1.05E-05

For Quicksort:



n	T(n) - try1	T(n) - try2	T(n) - try3	T(n) - avg	T(n)/n lg n
1,000,000	156	150	155	153.7	7.71E-06
2,000,000	337	311	326	324.7	7.755E-06
4,000,000	710	724	672	702.0	8.002E-06
8,000,000	1559	1634	1466	1553.0	8.465E-06
16,000,000	3199	3274	3056	3176.3	8.295E-06
					converging around a=8.4E-06

So both are truly O(n lg n) algorithms due to the convergence, but the multiplying factor (a) is much smaller for Quicksort. Also, the actual running times were clearly lower for Quicksort than for Mergesort.

Question 3

Since we know that the worst case for Quicksort is an already-sorted array, we would simply generate the array of N elements as containing the values 1 to N in that sequence, and then run the algorithm.