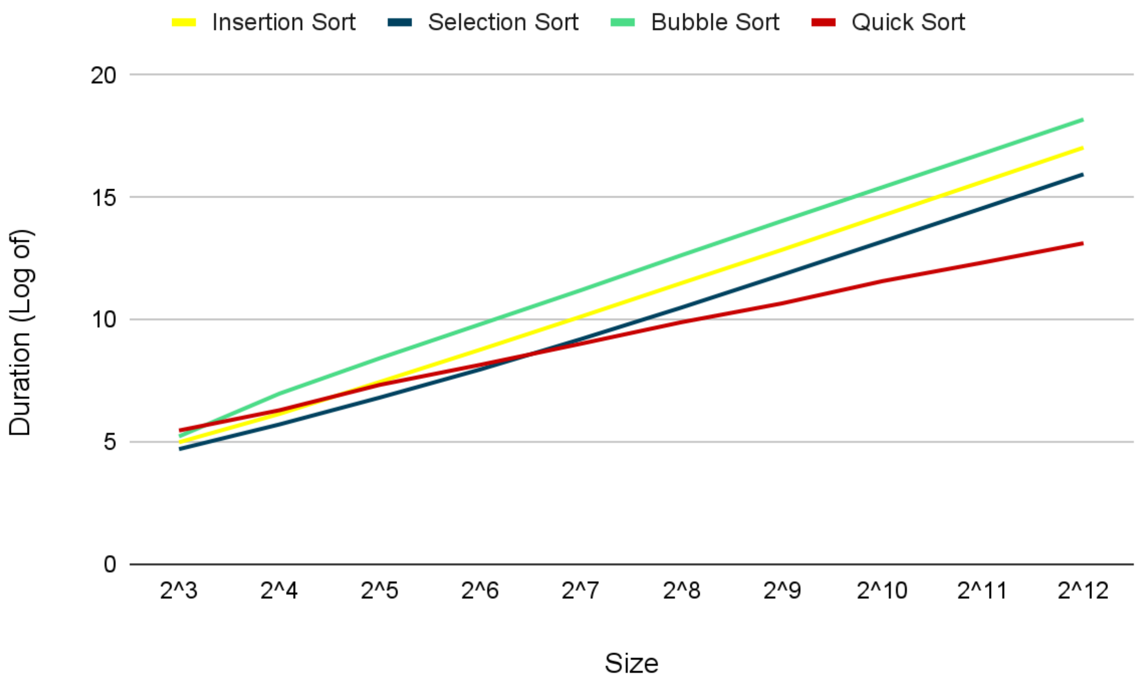
Mustafa Mert Gülhan
22201895
CS201-1

Homework 2: Algorithm Efficiency and Sorting

Task 1: Which Algorithm?

n	Insertion Sort	Selection Sort	Bubble Sort	Quick Sort
2^3	147	112	186	232
2^4	471	306	1074	545
2^5	1743	913	4572	1534
2^6	6517	2892	18384	3489
2^7	25183	9967	74569	8308
2^8	99364	36387	307860	19910
2^9	387948	138400	1254549	42855
2^{10}	1564946	539016	4999302	107081
2^{11}	6289668	2126725	19783951	228667
2^{12}	25215530	8447868	79384092	505681

Table 1: Time Estimations on Random Array



Plot 1: Time Estimations on Random Array (Size vs Log Duration)

Result Discussion:

As seen in both table and the plot most of the time quicksort does its job faster than the others. Also theoretically the average case of quicksort is O(nlogn) while others are O(n^2). Thus, using quicksort will be a good fit for the problem to save money and time.

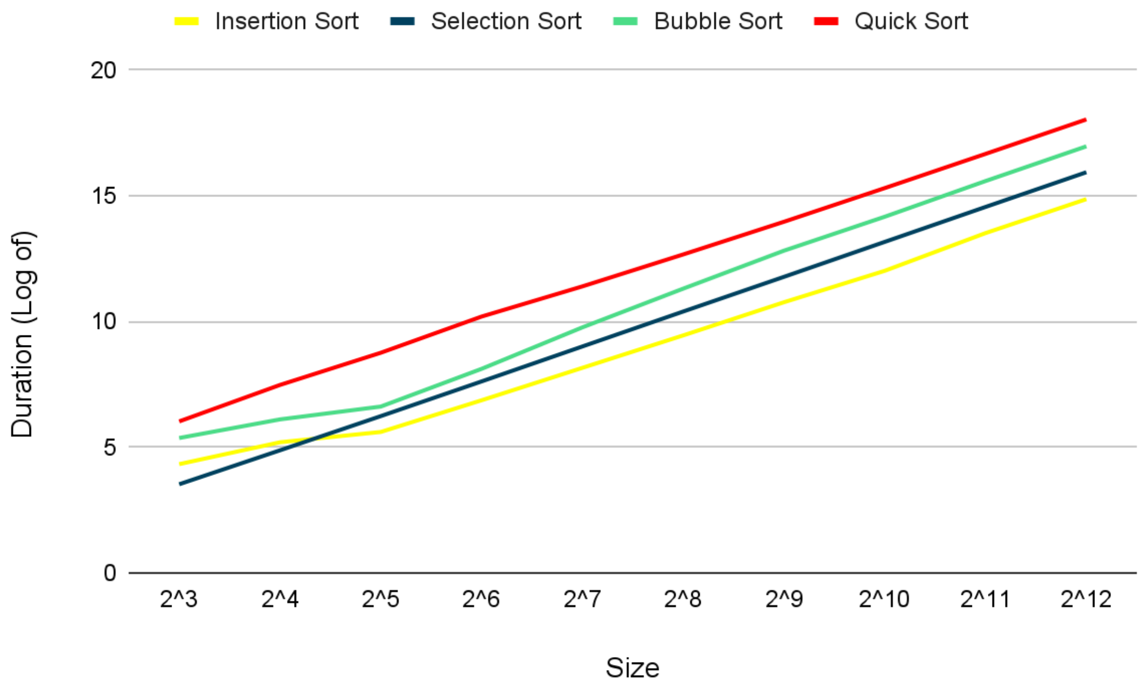
Task 2: An Assumption!

Recommended Algorithm and the Reason:

My suggestion would be Insertion Sort because the idea behind the Insertion Sort is finding the place for the current element in the sorted part while iterating, because most of the array is sorted the algorithm will only do swaps when it finds an unsorted element. Thus, Insertion Sort will be faster than the others.

n	Insertion Sort	Selection Sort	Bubble Sort	Quick Sort
2^3	76	34	214	415
2^4	181	132	450	1771
2^5	272	511	748	6376
2^6	963	2031	3361	26912
2^7	3495	8170	17456	89748
2^8	12764	32745	81438	316791
2^9	47694	131035	368913	1165436
2^{10}	166701	524208	1430530	4459597
2^{11}	746264	2096992	5879630	17403305
2^{12}	2860969	8388306	23389581	68438393

Table 2: Estimations on Almost Sorted Array



Plot 2: Time Estimations on Almost Sorted Array (Size vs Log Duration)

Results Discussion:

As mentioned in the recommendation part, both table and the plot confirm that Insertion Sort becomes useful for sorting almost sorted arrays. Thus, Insertion Sort achieves sorting more quicker and helps reducing the cost of the worker. Furthermore, quicksort was the most efficient in sorting a random array. However, in almost sorted arrays it is the worst. Therefore, this shows that additional properties may affect the sorting time. Thus, it shows that choosing the correct sorting method by analyzing the properties of the array is crucial.

Task 3: Increasing the number of workers

Recommended Algorithm and the Reason:

When the opportunity of dividing the array into two parts and giving each to a different worker is given, the Quick Sort algorithm immediately came to my mind because the divide and conquer algorithm behind the quick sort is similar to this idea. However, I thought because selection sort performs faster than the quick sort when size is less than 2^7 in the Task 1, using quicksort when the size of the array is bigger than 2^7 and otherwise using selection sort could give a more optimized result.

n	Worker 1 Minimum Time (s)	Worker 1 Maximum Time (s)	Worker 2 Minimum Time (s)	Worker 2 Maximum Time (s)
2^3	22	52	25	90
2^4	45	283	40	241
2^5	77	892	126	651
2^6	633	1713	696	1756
2^7	1172	5960	1531	6490

2^{8}	5191	13041	5032	10034
2^{9}	16638	28338	9750	23159
2^{10}	35965	62431	28550	57676
2^{11}	108851	123733	88296	116031
2^{12}	218530	266506	173247	256536

Table 3A: Maximum and Minimum Time Estimations for Each Worker

n	Quick and Selection Sort Combined with 2 Workers Average Time Estimation (s)	Quick and Selection Sort Combined with 2 Workers Average Cost (TL)	Quick Sort 1 Worker Average Time Estimation (s)	Quick Sort 1 Worker Average Cost (TL)
2^{3}	87	2.41	232	6.44
2^{4}	308	8.55	545	15.13
2^{5}	873	24.25	1534	42.61
2^{6}	2315	64.30	3489	96.92
2^{7}	7184	199.5	8308	230.78
2^{8}	16481	457.81	19910	553.06
2^{9}	40568	1126.89	42855	1190.42
2^{10}	89517	2486.58	107081	2974.47
2^{11}	212939	5914.97	228667	6351.86
2^{12}	459749	12770.80	505681	14046.69

Table 3B: Average Time Estimation and Costs (Suggested Algorithm vs Best Algorithm in Task 1)

Results Discussion:

As mentioned in the recommendation part, the algorithm became useful to reduce the costs and save time. The idea of doubling the workers seems to suit the idea of divide and conquer behind the Quick Sort algorithm. Thus, assigning the divided parts to the different workers helped the efficiency. Furthermore, when divided parts become small enough, switching to the selection sort instead of continuing on quick sort seems to also improve the algorithm because selection sort performs better and quicker in small sets. With these developments the Quick Sort algorithm has become less time consuming.