CS-202, Fall 2023

Homework 1- Algorithm Analysis and Sorting

Date: 23.10.23

Student Name: Efe Tokar

Student ID: 22103299

Question 1:

(a) $0 \le f(n) \le cn^4$ for all $n \ge n_0$

$0 \le 8n^4 + 5n^2 - 2n + 4 \le cn^4$ for all $n \ge n_0$

if $n_0 = 1$ ⟶ all $n \ge 1$

$8n^4$ is prominent over other elements of this function so;
$c \ge 8$ to make this inequality true for all $n \ge 1$

so $f(n) = 8n^4 + 5n^2 - 2n + 4$ is $O(n^4)$ if $c = 8$, $n_0 = 1$

(b)

i) Insertion Sort ⇒

8, 33, 2, 10, 4, 1, 34, 7

↓

8, | 33, 2, 10, 4, 1, 34, 7
sorted | unsorted

↓

8, 33, | 2, 10, 4, 1, 34, 7
sorted | unsorted

↓

2, 8, 33, | 10, 4, 1, 34, 7
sorted | unsorted

↓

2, 8, 10, 33, | 4, 1, 34, 7
sorted | unsorted

↓

2, 4, 8, 10, 33, | 1, 34, 7
sorted | unsorted

↓

1, 2, 4, 8, 10, 33, | 34, 7
sorted | unsorted

↓

1, 2, 4, 8, 10, 33, 34, | 7
sorted | unsorted

↓

1, 2, 4, 7, 8, 10, 33, 34
find form

ii) Merge Sort ⇒

8, 33, 2, 10, 4, 1, 34, 7

8, 33, 2, 10   4, 1, 34, 7

8, 33   2, 10   4, 1   34, 7

8   33   2   10   4   1   34   7

8, 33   2, 10   1, 4   7, 34

2, 8, 10, 33   1, 4, 7, 34

1, 2, 4, 7, 8, 10, 33, 34
find form

iii Quick Sort ⇒

→ 8, 33, 2, 10, 4, 1, 34, 7

→ 8, 2, 33, 10, 4, 1, 34, 7

→ 8, 2, 4, 10, 33, 1, 34, 7

→ 8, 2, 4, 1, 33, 10, 34, 7

→ 8, 2, 4, 1, 7, 10, 34, 33

→ 7, 2, 4, 1, <u>8</u>, 10, 34, 33 // first partition with pivot 8

→ 1, 2, 4, <u>7</u>, 8, 10, 34, 33 // pivot ⇒ 7

→ <u>1</u>, 2, 4, 7, 8, 10, 34, 33 // pivot ⇒ 1

→ 1, <u>2</u>, 4, 7, 8, 10, 34, 33 // pivot ⇒ 2

→ 1, 2, 4, 7, 8, <u>10</u>, 34, 33 // pivot ⇒ 10

→ 1, 2, 4, 7, 8, 10, 33, <u>34</u> // pivot ⇒ 34

(c.)

→ $T(1) = 1$

$T(n) = T(n/2) + n^2$

$T(n) = T(n/4) + n^2/4 + n^2$

$T(n) = T(n/8) + n^2/16 + n^2/4 + n^2$

...

$T(n) = T(n/2^k) + (n/2^k)^2 + (n/2^{k-1})^2 + (n/2^{k-2})^2 + \dots (n/2)^2 + n^2$

we do this until $n/2^k = 1 \Rightarrow n/2^k = 1 \rightarrow n = 2^k \rightarrow k = \log_2(n)$

→ sum of the series ⇒

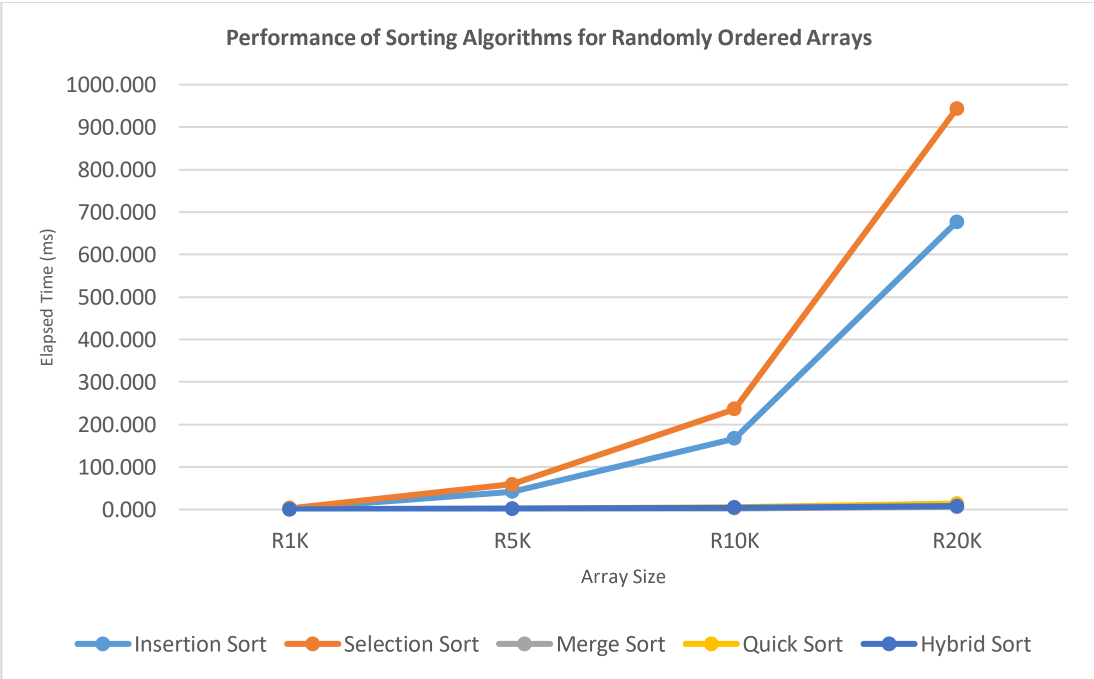
$T(n) = 1 + n^2/4 + n^2/4 + n^2/8 + n^2/16 + n^2/2^k \Rightarrow$ this is a geometric series with ratio $= 1/4$

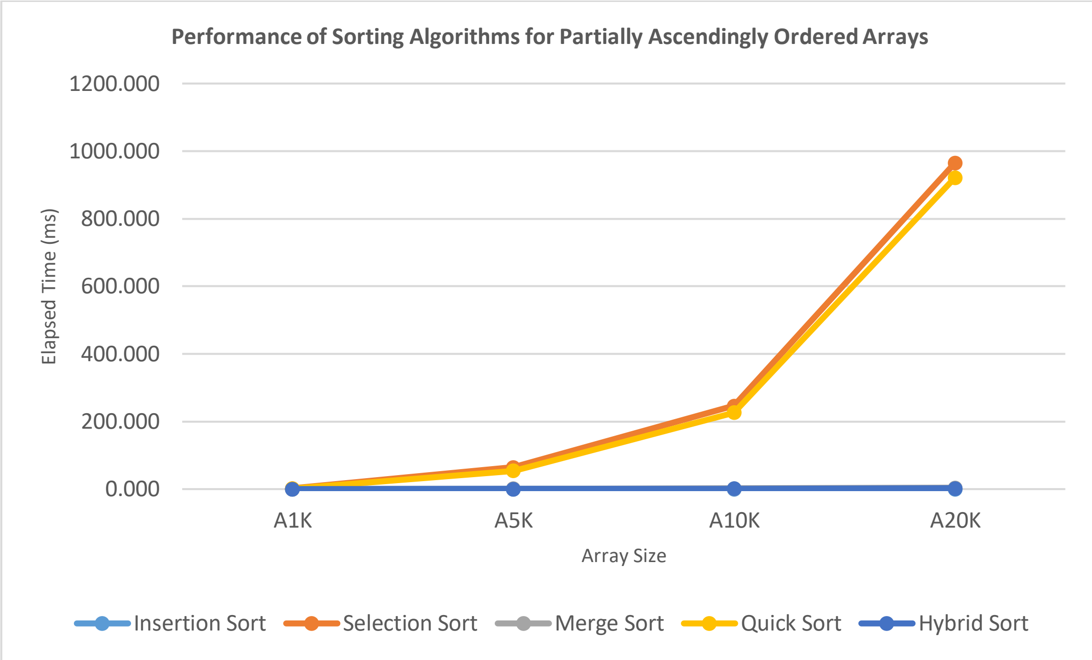
$$1 + \frac{n^2}{4}\left(1 - \left(\frac{1}{4}\right)^k\right) = 1 + \frac{n^2}{4}\left(1 - \left(\frac{1}{4}\right)^{\log n}\right) \Rightarrow O(n^2)$$

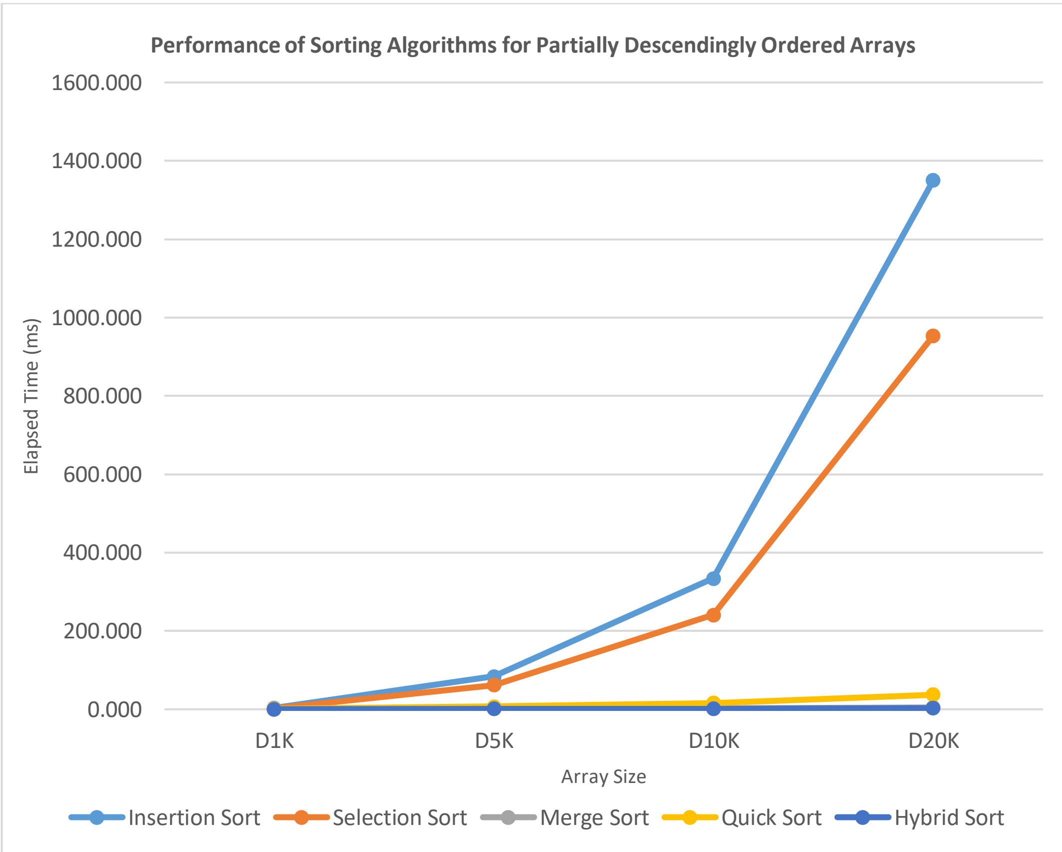
// the asymptotic running time of reccurence relation $T(n) = T(n/2) + n^2$ with the best case $T(1) = 1$

Question 3:

Array	Elapsed Time					Number of Comparisons					Number of Data Moves				
	Insertion Sort	Selection Sort	Merge Sort	Quick Sort	Hybrid Sort	Insertion Sort	Selection Sort	Merge Sort	Quick Sort	Hybrid Sort	Insertion Sort	Selection Sort	Merge Sort	Quick Sort	Hybrid Sort
R1K	1,665	2,451	0,260	0,252	0,321	248594	499500	8702	12511	15682	251591	2997	19952	13254	21540
R5K	41,592	59,350	1,408	1,736	1,888	6225934	12497500	55127	165915	103739	6240931	14997	123616	63525	141431
R10K	166,797	236,187	2,938	4,630	3,870	24768666	49995000	120272	584190	218127	24798663	29997	267232	173112	302499
R20K	677,079	943,740	6,068	13,894	7,990	99551149	199990000	260401	2150307	460157	99611146	59997	574464	294543	648697
A1K	0,024	2,556	0,181	1,753	0,123	585	499500	5309	364912	6385	3582	2997	19952	3561	13263
A5K	0,063	64,240	0,974	54,609	0,608	1167	12497500	32414	11482456	30683	16164	14997	123616	15933	82448
A10K	0,103	246,439	2,030	225,966	1,213	1592	49995000	69539	47798931	61300	31589	29997	267232	31089	183039
A20K	0,188	964,894	4,275	921,279	2,588	1507	199990000	148545	195401411	125487	61504	59997	574464	61098	403252
D1K	3,366	2,447	0,192	1,263	0,241	493814	499500	6144	88433	12558	496811	2997	19952	128748	21300
D5K	83,605	61,690	1,001	7,309	0,885	12371465	12497500	37700	612249	50773	12386462	14997	123616	750516	99107
D10K	334,232	241,003	2,195	16,039	1,543	49493873	49995000	80148	1470427	84335	49523870	29997	267232	1494177	199086
D20K	1349,860	953,502	4,553	36,870	2,869	197987907	199990000	168928	3973421	157984	198047904	59997	574464	3043341	419614







In this assignment, we were expected to write a code that records the comparison numbers, move numbers, and the time taken for the Insertion Sort, Selection Sort, Merge Sort, Quick Sort, and Hybrid Sort, a mixture of Bubble Sort-Merge Sort, which is specially prepared for this assignment. 12 different arrays with sizes 1000, 5000, 10000, 20000 and 4 of them randomly ordered, 4 of them with a partially ascending order, and 4 of them with a partially descending order were used to observe the difference between them according to the three parameters which were specified. While the code I wrote using the <ctime> algorithm showed times below 1 ms as 0 when I ran it in the *CLion environment*, I was able to achieve more precise results when I ran the code on the command line and I included these results in the table. To analyze each sorting algorithm separately,

1. **Insertion Sort:**
   For insertion sort, the **time complexity is O (n^2) for average and worst cases and O (n) for best case.** This is a relatively ineffective time complexity. It can be observed that the elapsed time is much larger than the other algorithms while sorting the randomly ordered array and the array with a partially descending order. However, in a scenario where the array is in an ascending order, this sorting algorithm can be useful. **It was the least efficient algorithm, in terms of time while sorting partially ascending arrays.** The comparison count can be a maximum of $n^2/2$ for an array of n size. We can observe that the comparison size never surpasses this limit through the table.
2. **Selection Sort:**
   Time complexity is O (n^2) for all cases for selection sort. This is a relatively ineffective time complexity and can be observed as the elapsed time is much larger than the other algorithms while sorting arrays with larger sizes (10000, 20000). **It was the least efficient algorithm, in terms of time while sorting randomly ordered and partially descending arrays.** The comparison count can be a maximum of n(n-1)/2 for an array of n size. We can observe that the comparison size never surpasses this limit through the table.
3. **Merge Sort:**
   Time complexity is O (n * log (n)) for all cases for merge sort. Merge sort is a very efficient algorithm compared to the others, as can also be seen in the table. **It was the most efficient algorithm, in terms of time while sorting randomly ordered arrays.** Also, the comparison count is relatively smaller than the other algorithms, which makes it the superior choice over the other algorithms.
4. **Quick Sort:**
   Time complexity is O (n * log (n)) for the average and the best-case scenarios for Quick Sort, which is similar to the Merge Sort. However as can be seen in the sorting process of the partially ascendingly ordered arrays, the Quick Sort algorithm is not efficient in this scenario, as we also choose the pivot as the first index of the array which comes with **a time complexity of O (n^2).** The comparison count number also depends on the scenario, which makes the Quick Sort an efficient algorithm except when the array is sorted ascendingly and the pivot is chosen as the first index.
5. **Hybrid Sort:**
   Time complexity is O (n * log (n)) for the hybrid sort due to the usage of the merge sort dominantly. In addition, while hybrid sort worked slightly less efficiently than merge sort in randomly generated arrays, it gave more efficient results in partially ascending and partially descending arrays with sizes over 1000. Therefore **it was the most efficient algorithm, in terms of time while sorting partially ascending and partially descending arrays.** Even though the elapsed time is small, the comparison count number and move count number are bigger than the merge sort algorithm as there is a usage of bubble sort during the sorting process.