Title: Algorithm Analysis and Sorting
Author: Dilara Mandıracı
ID: 22101643
Section: 1
Homework: 1
Description: This homework mainly focuses on time complexity analysis of the common sorting algorithms and their comparisons
Date: 13.03.2023

QUESTION 1

Part a)

Q: show that $f(n) = 6n^4 + 9n^2 - 8$ is $O(n^4)$ for appropriate c and n0 values.

S: We need to find two positive constants c and n0 satisfies that

$0 \leq 6n^4 + 9n^2 - 8 \leq cn^4$ for all n ≥ n0

Choose c=7 and n0 =3

$6n^4 + 9n^2 - 8 \leq 7n^4$ for all $n \geq 3$

Try for n ≥ 3 and c = 7

$6n^4 + 9n^2 - 8 \leq cn^4, n \geq 3, c = 7$

$9n^2 - 8 \leq n^4(c - 6)$

$9 * n^2 - 8 \leq n^4(7 - 6)$

$9 * n^2 - 8 \leq n^4$ is $true$ for all $n \geq 3$

Part b)

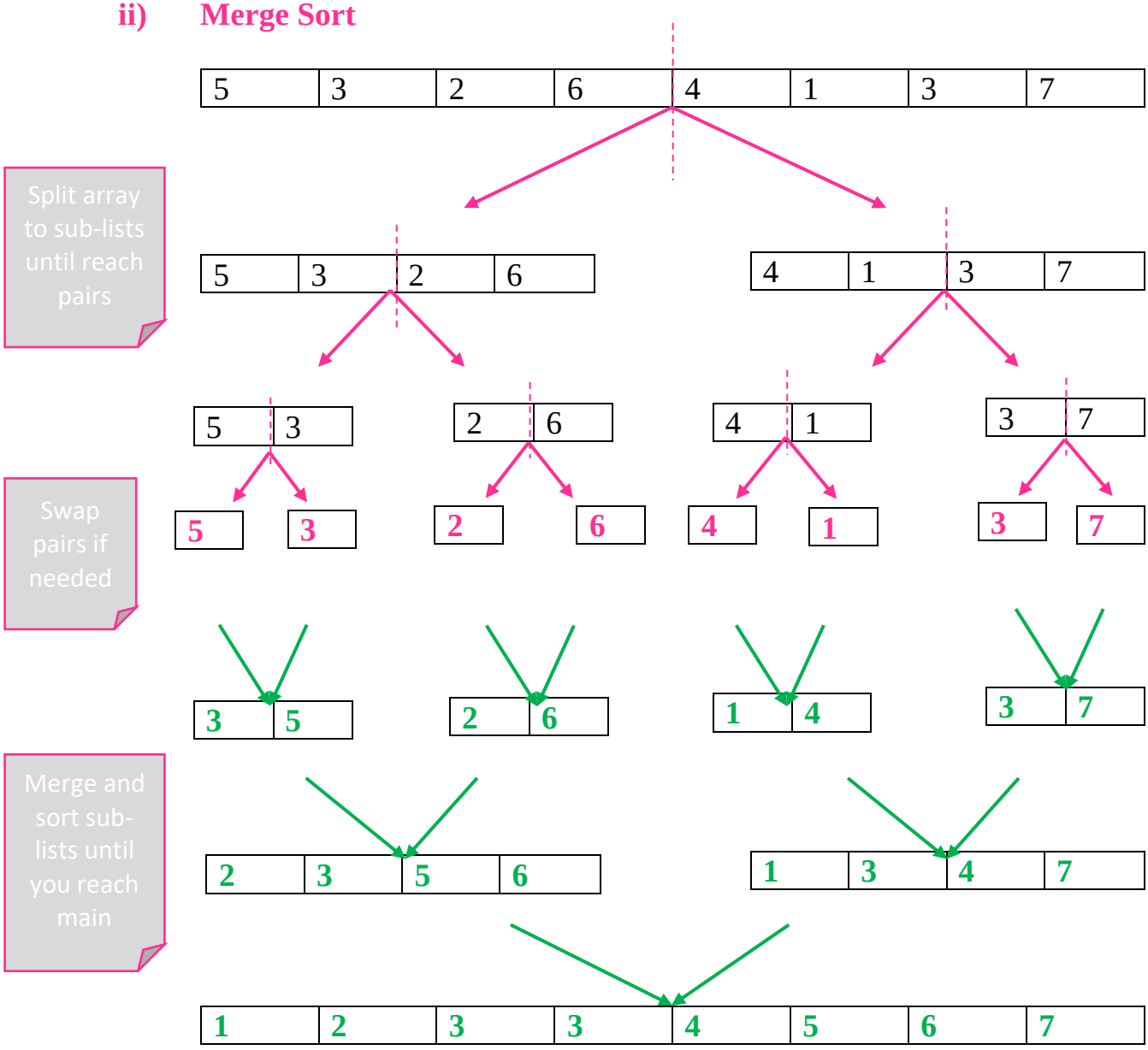
Tracing [5, 3, 2, 6, 4, 1, 3, 7] for different sorting algorithms.

i) Selection Sort

GREEN shows the sorted part
PINK shows the next min element
BLUE shows the element that is going to be swapped

5	3	2	6	4	1	3	7
1	3	2	6	4	5	3	7
1	2	3	6	4	5	3	7
1	2	3	6	4	5	3	7
1	2	3	3	4	5	6	7
1	2	3	3	4	5	6	7

ii) Merge Sort



iii) Quick Sort

RED denotes pivot elements
BLUE denotes the elements swapped
GREEN denotes they are in the correct place

5	3	2	6	4	1	3	**7**

All elements smaller than 7 is left to the 7. So, 7 is in the right position.

5	3	2	6	4	**1**	**3**	**7**

1	3	2	6	4	**5**	**3**	**7**

1	3	**2**	**3**	4	6	**5**	**7**

Elements smaller than 3 is on the left of the pivot larger others in the right.

1	**2**	**3**	3	4	**5**	**6**	**7**

1	**2**	**3**	3	4	**5**	**6**	**7**

Elements < 2 Elements < 5 Elements > 5

1	2	3	3	**4**	**5**	**6**	**7**

Since all elements smaller than 4 and bigger than 4 is the right position, no change in here.

1	**2**	**3**	**3**	**4**	**5**	**6**	**7**

Part c)

$T(n) = 2T(n-1) + n^2$ where $T(1) = 1$.
So,

$$
\begin{aligned}
T(n) &= 2T(n-1) + n^2 \\
&= 2(2T(n-2) + (n-1)^2) + n^2 \\
&= 2(2(2T(n-3) + (n-2)^2) + (n-1)^2) + n^2 \\
&\;\;\vdots \\
&= 2^k\, T(n-k) + (2^{(k-1)}\,(n-k+1)^2 + 2^{(k-2)}\,(n-k+2)^2 + \cdots + n^2)
\end{aligned}
$$

For k $=$ n $-$ 1 and $T(1) = 1$

$$= 2^{n-1}\, T(1) + (2^{(n-2)}\,(2)^2 + 2^{(n-3)}\,(3)^2 + \cdots + n^2)$$

$$= 2^{n-1} \cdot 1 + \sum_{i=0}^{n-2} 2^i (n-i)^2$$

$$= 2^{n-1} \cdot 1 + (-n^2 - 4n + 11 \cdot 2^{n-1} - 6)$$

$$= 12 \cdot 2^{n-1} - n^2 - 4n - 6$$

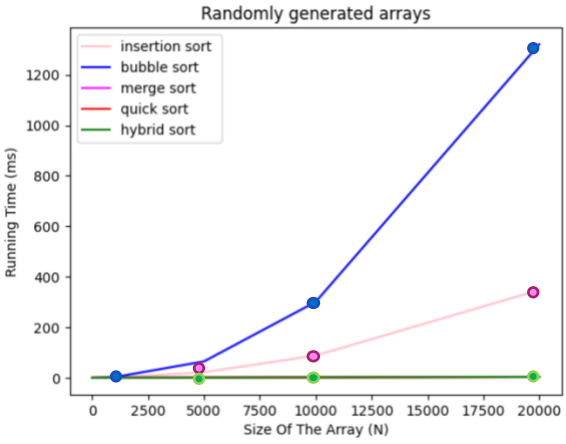
Since the element with highest growth rate dominates the growth rate of the function *and it is* 2^n. (2^n *grows faster than* n^2)

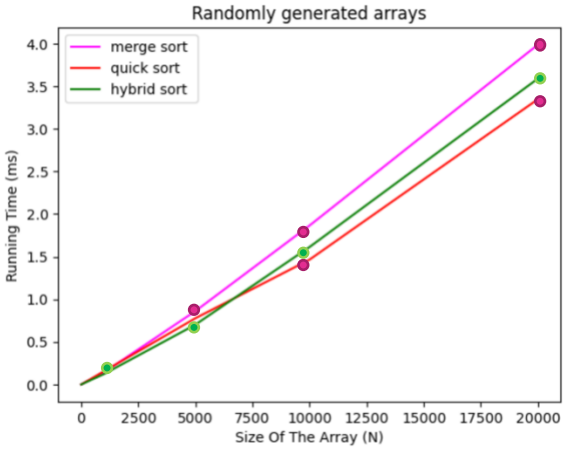
$$T(n) = O(2^n)$$

ARRAY	Elapsed Time (ms)					Number of Comparisons					Number of Data Moves				
	Insertion Sort	Bubble Sort	Merge Sort	Quick Sort	Hybrid Sort	Insertion Sort	Bubble Sort	Merge Sort	Quick Sort	Hybrid Sort	Insertion Sort	Bubble Sort	Merge Sort	Quick Sort	Hybrid Sort
R1K	1.009	2.502	0.143	0.156	0.123	261490	498324	8731	11706	14677	262489	781473	19952	21059	23653
R5K	21.961	64.744	0.861	0.777	0.702	6451707	12495085	55218	69919	85889	6456706	19340124	123616	104490	119193
R10K	88.317	300.743	1.865	1.465	1.613	26291454	49979424	120335	148315	179936	26301453	78844365	267232	232488	260852
R20K	347.391	1320.949	3.987	3.347	3.589	103537082	199969699	260970	326356	389064	103557081	310551249	574464	580150	637923
A1K	0.011	0.013	0.090	0.397	0.379	1887	2994	5691	112776	112089	2886	2664	19952	10434	4840
A5K	0.050	0.060	0.522	3.757	3.554	9582	14994	35757	1066640	1051664	14581	13749	123616	100735	27859
A10K	0.119	0.117	1.124	26.197	25.854	19229	29994	76007	7726915	7705375	29228	27690	267232	163045	52262
A20K	0.206	0.232	2.332	96.541	96.182	38570	59994	161891	28629988	28607140	58569	55713	574464	265671	105668
D1K	1.678	2.728	0.090	0.911	0.919	499611	499500	5228	167356	167708	500610	1495836	19952	269256	269270
D5K	42.055	67.022	0.524	32.454	32.509	12497916	12497500	31495	6132264	6133769	12502915	37478751	123616	9442299	9439146
D10K	168.211	268.203	1.074	94.442	94.583	49995769	49995000	67815	17711531	17715110	50005768	149957310	267232	27761994	27756216
D20K	671.226	1072.242	2.297	412.133	411.453	199991428	199990000	144606	77866137	77878141	200011427	599914287	574464	119260304	119264871

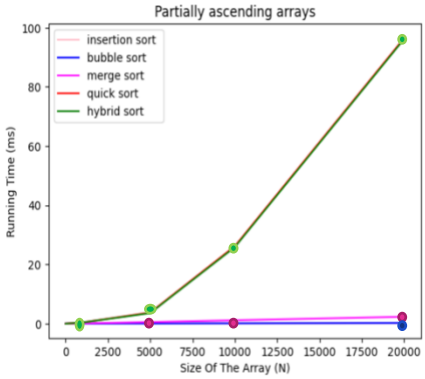
QUESTION 3

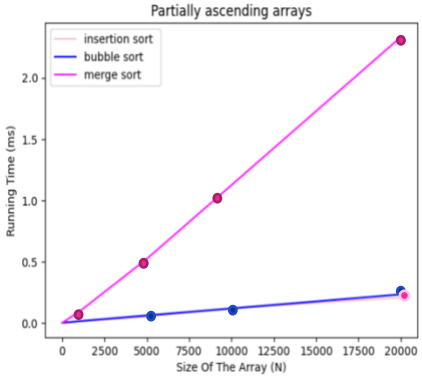
Since the growth rate of the merge, quick and hybrid sorts are relatively so much smaller than insertion and bubble sort, it is almost impossible to see them in the same graph. So, I preferred to show them in detail.

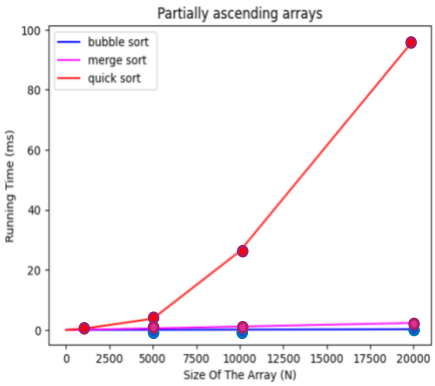




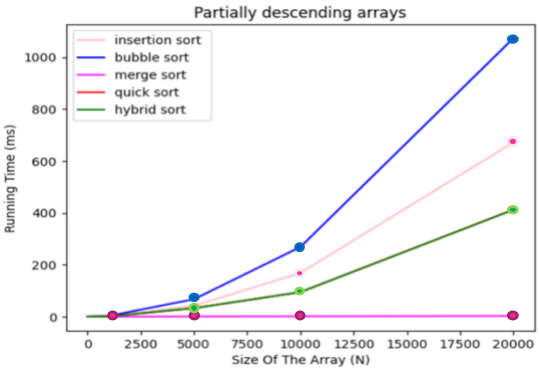
Same thing is valid for this situation. Quick sort and Hybrid sort algorithms are so close to each other in terms of running time and higher than others. Insertion sort and bubble sorts are too closed to each other.

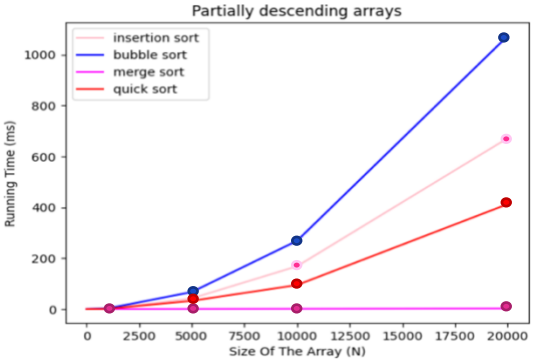






Same thing is valid for this situation. Quick sort and Hybrid sort algorithms are so close to each other I decided to show them in different graphs.





For all array types, if the array size is equal to or smaller than 1000, we can observe that elapsed times are similar in different algorithms. For different array types, it is observed that while partially ascending arrays have faster sorting time for some algorithms, such as bubble and insertion sort, some of them have the shortest time in the randomly generated arrays. Since the sort will be done in ascending order, partially descending arrays have the longest time except for merge sort.

Partially ascending arrays are created by randomly permutating every consecutive $\log_2 n$ element. This placement increases the chance that some parts can be already sorted, or some elements are in the correct place. So, this case is closer to the best case of insertion sort and bubble sort O(n), which is the array is already sorted. It can be observed from both graphs, and the table that partially ascending arrays have shorter elapsed time in bubble sort and insertion sort than other algorithms. Also, graphs verify that growth is linear for every given size from 1000 to 20000. Insertion and bubble sort has the worst and average case $O(n^2)$ growth rate. This can also be seen in the graphs from randomly generated and partially descending arrays since we sort the arrays in ascending order. Even though the elapsed time may be reasonable for small-

sized arrays, it is so much higher when the size increases to the 20K arrays because of quadratic time for randomly generated and partially descending arrays. In addition, the number of comparisons and move count are relatively much less in the partially ascending arrays than the other type of arrays.

Quick sort and Hybrid sort are not as efficient in partially ascending and descending arrays as they are in randomly distributed arrays. Since quick sort and hybrid sort are so similar and they both use partition, their worst case occurs if the pivot is the smallest or larger element in the array. Because both ascending and descending arrays include the smallest or largest elements in the first log2n part. In the algorithm, the pivot is always chosen as the first element of the array. It becomes closer to the worst case of the algorithms, which is $O(n^2)$. Even there is a chance to choose the smallest and largest one. If we analyze it by looking at size and elapsed time table, it can also be seen that elapsed time grows so much faster than the size ratio. Graphs also verify that quadratic time. The number of data moves is much less than in partially ascending arrays because some elements are already in the correct position or at least in the correct partition. So, the data movement is less in the partially ascending arrays compared to the other type of arrays. In addition, since hybrid sort uses bubble sort when the size becomes less than or equal to 20, it makes the move count almost half of the quick sort in the partially ascending arrays since less swap or swapping closer elements makes the array sorted.

The running time of merge sort always has similar results in same-sized but different-typed arrays, according to the results table. It is because the merge sort algorithm always O(nlogn) in the best, worst, and average cases. In partially ascending arrays, the merge sort has higher running time than both insertion and bubble sort because it is the best case of the two of them, which is O(n), but the merge sort always runs at O(n log(n)). So, even if the average growth rate is higher in these two if we have partially ascending arrays, it is better to use insertion and bubble sort, especially for a large amount of data. The move count of merge sort is always the same for same-sized arrays, not depending on the array type. It is because merge sort divides the array into two to reach the pairs every time. So, it makes a move count always the same for same-sized arrays.