CS202 HW1

Elif Gülşah Kaşdoğan/21601183

Q1.

(a) $f(n) = 4n^5 + 3n^2 + 1$ is O(n^5) if $4n^5 + 3n^2 + 1 \leq n^5 * c$ for some constant c where $n \geq n_0$

For c = 5 and n_0=2 equation will be satisfied.

(b) ->T(n) = T(n - 1) + n^2, T(1) = 1

T(n-1) = T(n - 2) + $(n-1)^2$

T(n-2) = T(n - 3) + $(n-2)^2$

T(n-3) = T(n - 4) + $(n-3)^2$

From 1 to k

T(n-k) = T(n – (k+1)) + $(n-k)^2$

T(n) = T(n - 4) + $(n-3)^2$+ $(n-2)^2$+ $(n-1)^2$ + n^2

T(n)=T(n-k) + $(n-k+1)^2$ +…+ $(n-3)^2 + (n-2)^2$+ $(n-1)^2$ + n^2

T(1)=T(0)+1 -> T(0)=0

T(n)=T(n-k) + $\sum_0^{k-1}(n-i)^2$

For n=k

T(n)= $(2n^3 - n^2 + n)/6$ because T(0)=0 -> $\theta(n^3)$

->T(n) = 2 T(n/2) + n/2, T(1) = 1

T(n/2)=2T(n/4) + n/4,

T(n/4)=2T(n/8)+n/8,…

T(n) = 2(2(2T(n/8)+n/8) + n/4))

If we go up to k

T(n)=2^k T(n/2^k) + kn/2

Say n = 2^i and $log_2 n$ = I

T(2^i)= 2^k T($2^i/2^k$) + k(2^i)/2 when i=k

T(2^k)= 2^k $\theta(1)$+k*2^{k-1}

T(n)=n + nlgn/2 -> $\theta(nlgn)$

(c) Original array: [8, 4, 5, 1, 9, 6, 2, 3]

Selection Sort:

Initial Array: 8, 4, 5, 1, 9, 6, 2, 3|

After 1st swap: 8, 4, 5, 1, 3, 6, 2, | 9

After 2nd swap: 4, 5, 1, 3, 6, 2, |8, 9

After 3rd swap: 4, 5, 1, 3, 2, |6, 8, 9

After 4th swap: 4, 1, 3, 2, |5, 6, 8, 9

After 5th swap: 1, 3, 2, |4, 5, 6, 8, 9

After 6th swap: 1, 2, |3, 4, 5, 6, 8, 9

After 7th swap: 1|2, 3, 4, 5, 6, 8, 9

Bubble Sort:

Initial Array: 8, 4, 5, 1, 9, 6, 2, 3|

Pass1: 4, 8, 5, 1, 9, 6, 2, 3

4, 5, 8, 1, 9, 6, 2, 3

4, 5, 1, 8, 9, 6, 2, 3

4, 5, 1, 8, 9, 6, 2, 3

4, 5, 1, 8, 6, 9, 2, 3

4, 5, 1, 8, 6, 2, 9, 3

After Pass1: 4, 5, 1, 8, 6, 2, 3, 9

After Pass2: 4, 1, 5, 6, 2, 3, 8, 9

After Pass3: 1, 4, 5, 2, 3, 6, 8, 9

After Pass4: 1, 4, 2, 3, 5, 6, 8, 9

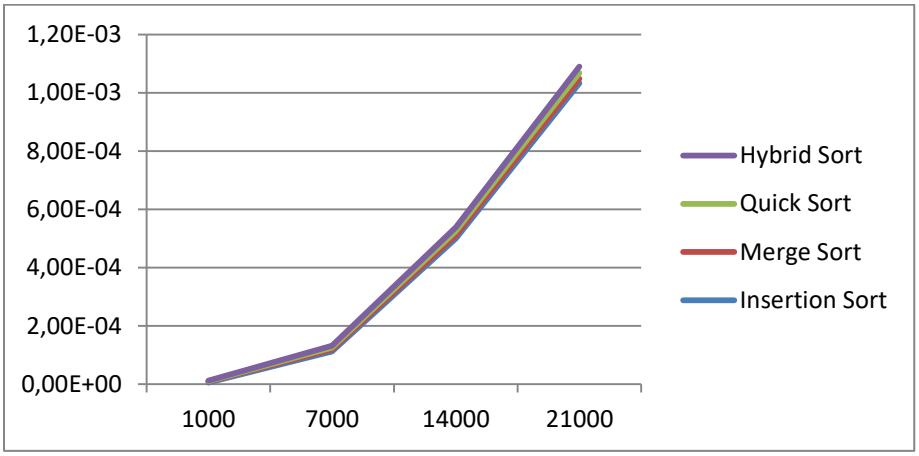
After Pass5: 1, 2, 3, 4, 5, 6, 8, 9

After Pass6: 1, 2, 3, 4, 5, 6, 8, 9

Q3. Report: Performance Analysis

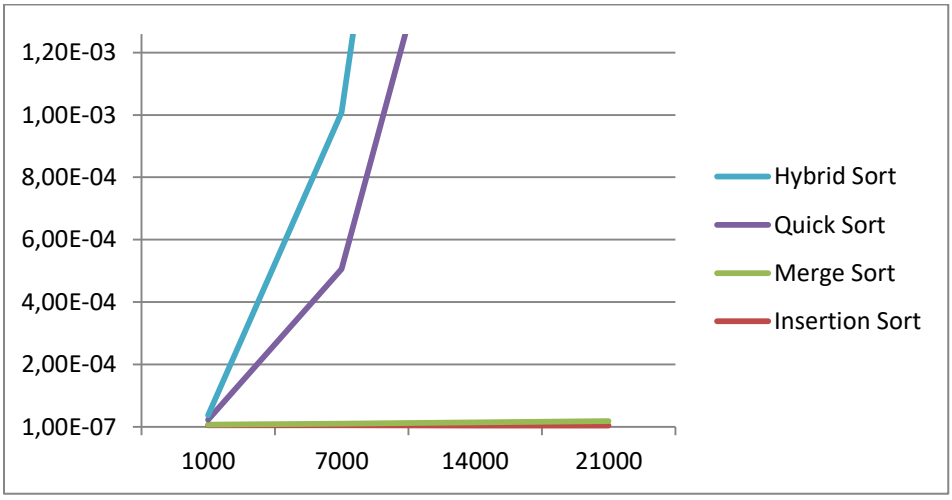
Arrays	Elapsed Time (in miliseconds)			
	Insertion Sort	Merge Sort	Quick Sort	Hybrid Sort
R1K	6,00E-06	3,00E-06	1,00E-06	2,00E-06
R7K	1,12E-04	7,00E-06	6,00E-06	7,00E-06
R14K	5,00E-04	1,10E-05	1,40E-05	1,30E-05
R21K	1,03E-03	1,50E-05	2,00E-05	2,20E-05
A1K	5,00E-06	2,00E-06	1,50E-05	1,50E-05
A7K	5,00E-06	5,00E-06	4,96E-04	5,02E-04
A14K	4,00E-06	1,00E-05	2,07E-03	1,86E-03
A21K	4,00E-06	1,40E-05	0,003949	0,004034
D1K	7,00E-06	1,00E-06	1,00E-05	9,00E-06
D7K	2,72E-04	6,00E-06	3,55E-04	3,78E-04
D14K	9,96E-04	9,00E-06	1,56E-03	1,39E-03
D21K	2,09E-03	1,30E-05	0,002838	0,002403

Random integers can be seen as average case in this report. Insertion sort works O(n^2), merge sort works O(n log(n)),quick sort and hybrid sort works in O(n log(n)) theoretically. I expect insertion sort to increase in order n^2 if we look at random7k and random 14k we can see that input size doubles itself and runtime ratio increases by nearly four if we look at 5/1.12=4.46. Merge sort input size increases from 1k to 7k, I expect runtime to increase about 7*log7 times, I could not get the exact runtime as I expected at this point but this may be related to array items and timing problems. For quick sort from 1k to 7k I expect it to increase by 5.447 times it is more than that but meets the expectations in terms of ratio. Hybrid sort will give us values similar to quick sort. We can capture similar ratio as we captured in quick sort from 1k to 7k. Similar ratios can be captured other parts of the table too.

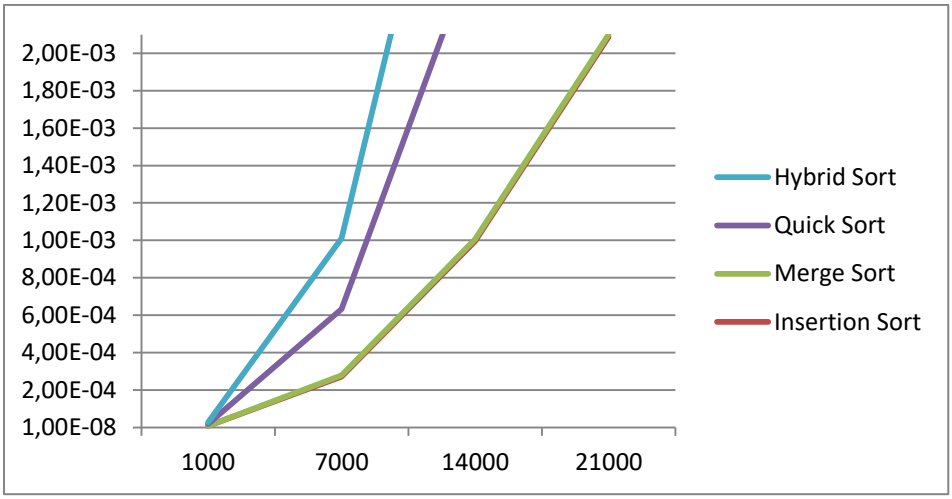


In ascending order insertion sort works in O(n), merge sort works in O(n log(n)), quick and hybrid works in O(n^2) theoretically. Insertion sort gives its best performance in ascending ordered arrays, as we can see it from the table especially for 7k, 14k and 21k. Merge sort gives better performance than

expected but it can be also related to timing issues. In quicksort form 1k to 7k we expect an increasing of running time by ratio 49 in theory, I obtained 33.07 in practice, from 7k to 21k I expect increasing by ratio 9, I obtained 7 in practice which means quick sort also works better than expected. Hybrid sort ratios are also consistent.



In descending order insertion sort works in O(n^2), merge sort works in O(n log(n)), quick and hybrid sort works in O(n^2) theoretically. Insertion sort works in worst case in descending order as we can see from the table. Practical ratio did not catch up the theoretical for insertion sort however for mergesort, quicksort and hybrid sort we can catch some ratios. Merge sort catchs up the ratio nlgn from 7k to 14k, 1k to 7k and so on. It can be said that Merge Sort is consistent in its column. Similarly quick sort's ratios will be less than expected on some points. But it can be said that quick sortand hybrid sort are not convinient to use for descending ordered arrays.



In general insertion sort can be preferred over merge sort and quick sort for either small data sets or for arrays sorted in ascending order since insertion sort works in its best case O(n) but quick sort works in its worst which is O(n^2) Merge sort will be in its best performance the elements are in ascending or descending order since merge sort works in O(nlgn) time for all cases and quick sort works in O(n^2) in its worst merge sort can be preferred over quick sort if items are already sorted. Hybrid sort can be chosen over quick sort if data size is small and array is sorted to gain some time. For large data sets there will not be so much difference.