Asuman Aydın

21502604

CS 202, Fall 2018

Homework #1 – Algorithm Efficiency and Sorting

Question 1:

a- Let say f(n) and g(n) be two functions and c is constant for this question

$f(n) = O(g(n))$ while $n \to n_0$ where in this question n_0 is ∞.

For c > 0 such that

$|f(n)| <= c*g(n)$

So that

$4n^5 + 3n^2 + 1 <= 4n^5(1 + 3/(4n^3) +1/(4n^5) = 8n^5$

$O(8n^5) = O(n^5)$

b- 1) $T(n) = T(n - 1) + n^2 \quad T(1) = 1$

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

$= [T(n - 2) + c*(n-1)^2] + c*n^2$

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

$= [T(n - 3) + c*(n-2)^2] + c*(n-1)^2 + c*n^2$

$= T(n - 3) + c*(n-2)^2 + c*(n-1)^2 + c*n^2 \dashrightarrow T(n – m) ....+ c*(n-m-2)^2 + c*(n-m-1)^2 + c*n^2$

When putting T(1) = 1 in place, we get n-m = 1. Thus

$T(1) + c*(2)^2 + c*(1)^2 ......+ c*n^2$

The formula becomes $n*(n+1)*(2n+1) = (2n^3 + 3n^2 + n)/6$ .as n^3 grows faster than others,

$T(n) = \Theta(n^3)$

2) $T(n) = 2 T(n/2) + c*n/2$

$= 2 [2T(n/2^2) + c*n/2^2] + c*n/2$

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

$= 2^2 [2T(n/2^3) + c*n/2^3] + c*n$

$= 2^3T(n/2^3) + 3c*n/2 \dashrightarrow$

$= 2^mT(n/2^m) + mc*n/2$ where m is the time which is decreasing by n/2 and if we say it

Reaches the base case 1 in the time

$c*n/2^m = 1 \rightarrow n = 2^m \rightarrow logn = m \rightarrow$ then $2^mT(n/2^m)$ becomes constant.

$= c_2*n + 3*logn* c*n/2$

$= O(nlogn)$

c- We choose indexSoFar 8 and then we check if there is larger than this value. By incrementing by 1 we swap the values in Selection Sort.

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

We swap the pairs with comparison from the beginning and have the sorted in the end always.

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

→ 4 5 1 8 6 2 3 /9 → 4 1 5 8 6 2 3 9 → 4 1 5 6 8 2 3 9 → 4 1 5 6 2 8 3 9 → 4 1 5 6 2 3 8 9 → 1 4 5 6 2 3 8 9 → 1 4 5 2 6 3 8 9 → 1 4 5 2 3 /6 8 9 → 1 4 2 5 3 6 8 9 → 1 4 2 3 /5 6 8 9→ 1 2 4 3 5 6 8 9 → 1 2 3 4 5 6 8 9

Question 3:

In question 3, as Windows operating did not allow me to have floating number in “Elapsed Time” calculation, I had the results with integer type even if I wrote float in my code. I compared the results as the data I have then.

	Elapsed Time (in milliseconds)			
Arrays	Insertion Sort	Merge Sort	Quick Sort	Hybrid Sort
R1K	2	0	0	0
R7K	170	40	0	0
R14K	700	80	0	0
R21K	1590	130	10	10
A1K	0	0	0	0
A7K	0	40	370	370
A14K	0	90	1480	1480
A21K	0	130	3330	3330
D1K	0	0	0	0
D7K	360	40	230	230
D14K	1390	90	950	950
D21K	3140	130	2130	2130

Elapsed Time Analysis: For theoretical results, there are some major differences between them. However, When we looked at the major differences between insertion and the others, insertion sorting time shows us that for a large number of arrays, it works way slower than the other sorting algorithms except if the array is ascending ordered. For Merge Sort and Quick Sort, the big difference appeared for ascending ordered array. For merge, it did not take that much to execute compare to Quicksort because Quick Sort again processes the array in choosing the pivot and comparing again from the end. So it took more time than Merge Sort. The difference between Quick Sort and Hybrid Sort in my data because of the reason I have given in above under Question 3. The range of difference is so small because the only difference is size comparison. Other than it is expected that they will be closed to each other. The only major result I have achieved in this table is that for the random array, Quick Sort is much faster than MergeSort because the random array is the worst case for both of them but Quick Sort is faster (n^2) than Merge Sort (nlogn).

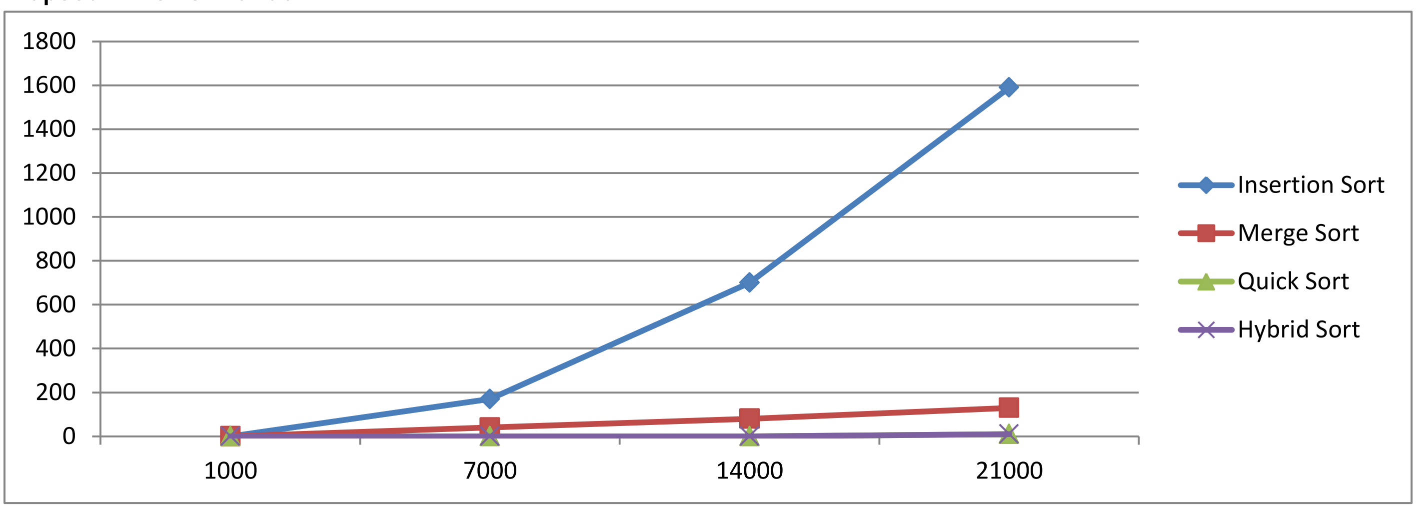
	The Number of Data Moves			
Arrays	Insertion Sort	Merge Sort	Quick Sort	Hybrid Sort
R1K	725838	20951	20394	20394
R7K	36262044	186615	141726	141726
R14K	146640939	401231	279493	279493
R21K	327845547	627463	396740	396740
A1K	0	20951	1505493	1505493
A7K	0	186615	73538493	73538493
A14K	0	401231	294076993	294076993
A21K	0	627463	661615493	661615493
D1K	1498500	20951	755493	755493
D7K	73489500	186615	36788493	36788493
D14K	293979000	401231	147076993	147076993
D21K	661468500	627463	330865493	330865493

The Number Of Data Moves Analysis: For Quicksort and Merge Sort in worst cases, the move numbers are close to each other in the theoretical matter. From n^2 rough calculation and from my data, the comparison would have the true result. In the table when we look at merge R14K and R21K and also Quick Sort same lines, it is the half rate of each other which implies the theoretical result. Quick Sort has the most move count in D21K except A21K.

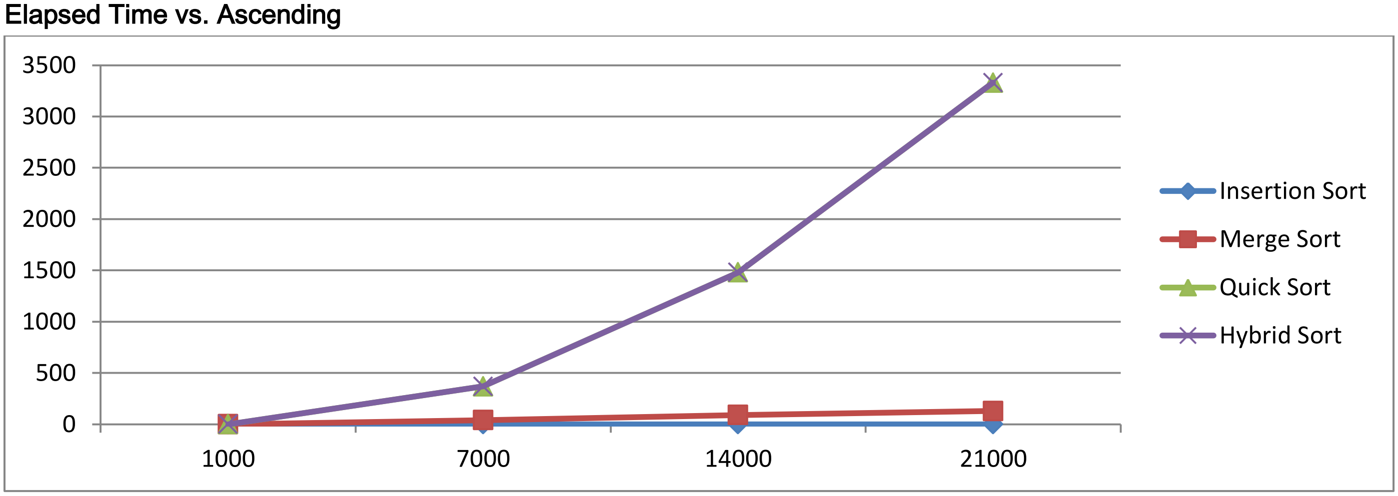
	Key Comparison			
Arrays	Insertion Sort	Merge Sort	Quick Sort	Hybrid Sort
R1K	242946	9165	18626	18626
R7K	12094348	72677	335234	335234
R14K	49121313	152545	1163023	1 163023
R21K	328056547	238744	2456075	2456075
A1K	999	10088	999000	999000
A7K	6999	92360	48993000	48993000
A14K	13999	198720	195986000	195986000
A21K	20999	313016	440979000	440979000
D1K	999999	4932	749000	749000
D7K	48999999	43628	36743000	36743000
D14K	195999999	94256	146986000	146986000
D21K	440999999	146724	330729000	330729000

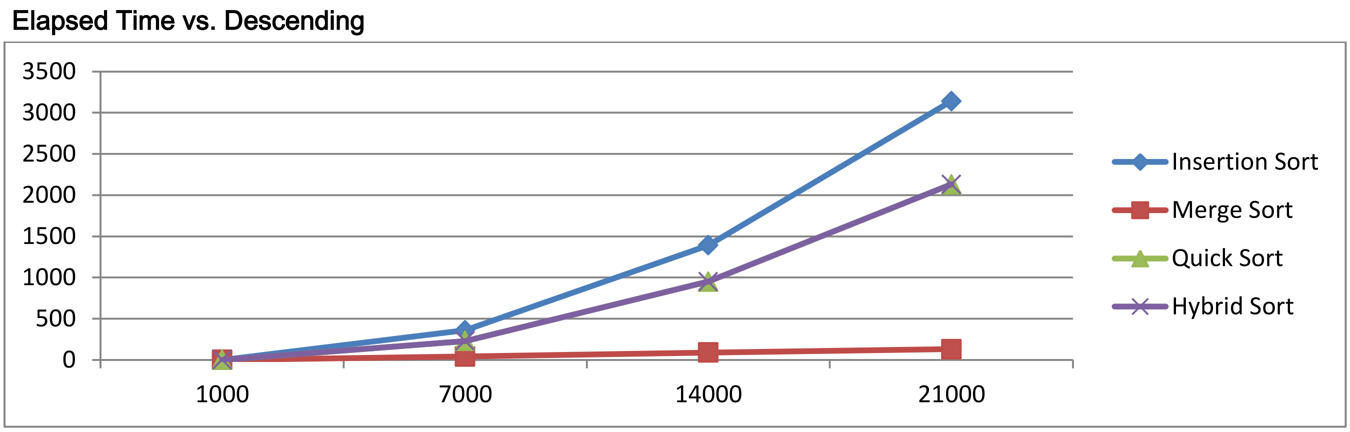
Key Comparison Analysis: In theoretical of Insertion Sort, for the random array it was expected worst-case scenario. In the worst case the number would be the biggest. When we compare with other sorting algorithms, it is the largest. In a rough calculation, for 21000 arrays size, it would be 441*106. It is very close to this number in my table for R21K. In ascending ordered it was expected that it will make the comparison as much as array size -1 in Insertion Sort. It is the same in the table. Although there is not much difference in Insertion Sort, I should focus on Merge and Insertion Sort. For Merge Sort, the best case is either descending or ascending ordered list sorting. When comparing A data and D data of Merge with Quick, it is smaller. Worst case of QuickSort is ascending or descending ordered list sorting. So in comparing Quicksort for a different type of arrays, the random array is least comparison made one. So theoretically and my data are close mostly.

Elapsed Time vs. Random



Elapsed Time vs. Ascending





- In general, one of the conditions for preferences is that if we have an array which is partially or mostly sorted, we choose insertion sort. It is chosen mostly if there are subproblems that are considered small to sort and solve. Also, it does not require additional memory which is more than a constant amount. For merge sort, the array properties do not matter. It does what it does even if the array is mostly sorted. However, the level of sorting in the array does matter. If it is not sorted at all, randomly chosen and etc. , merge sort is chosen to sort.
- Considering the array sizes that I experimented with the code, insertion sort is not preferred over any of them. Since they are not as small as I talked in the last entry. Besides memory, the size of the array and sorting state, the qualities of the computer we are using to sort out the list is a factor. It just depends on the hardware, speed and memory situation.
- About insertion sort vs quicksort, in quicksort, there are fewer swaps than merge sort even if it depends on the implementation. The swaps are changing the way to approach as CPU and memory usage wise in a bad way. So in some points, we can choose quicksort over merge sort. However, again insertion sort is more efficient to use in small arrays because even if it is not sorted at all, the quicksort will take time and memory to run recursion and all partionining issues.
- Depending on whether we want to see the difference in memory wise or speed. If there is ascending or descending arrays, it is better to choose merge sort. Their strategy is similar but as I concluded after the data I have obtained, merge sort saves time in this cases.
- Hybrid Sort includes both insertion sort and quick sort. It takes the advantages of both insertion and quicksort. As I experienced that insertion sort is more efficient in small array cases, hybrid array chooses insertion for the array size less than 10. And over that particular size of the array, it applies quicksort. Thus, yes hybrid sort has an advantage over quick sort by being more adaptable. However, it wastes time in comparing if the array size is small or big so it might take more time than quicksort in this case.