Title: Algorithm Analysis & Sorting
Author : İlhami Uluğtürkkan
ID: 22102546
Section : 2
Homework : 1
Description : Questions 1 and 3

Question 1:

A-) Show that $f(n) = 6n^4 + 9n^2 - 8$ is $O(n^4)$ by specifying the appropriate c and n_0 values in Big-O definition.

Answer: I know that the definition of Big-O is " T(n) = O(f(n)) if there are positive constants c and n_0 such that $T(n) <= c*f(n)$ when $n>=n_0$" from CS202 slides. In this case $T(n) = 6n^4 + 9n^2 - 8$ and $f(n) = n^4$.
When I put 7 for c. $7n^4 >= 6n^4 + 9n^2 - 8$.
From there $n^4 - 9n^2 + 8 >= 0$, and $(n^2 - 1) * (n^2 - 8) >= 0$.
So $7n^4$ is bigger than $6n^4 + 9n^2 - 8$ on (-infinity, 1] and [3, +infinity).
Therefore c=7 and $n_0 = 3$.

B-) Trace the below mentioned sorting algorithms to sort the array [5, 3, 2, 6, 4, 1, 3, 7] in ascending order. Use the array implementation of the algorithms as described in the textbook and show all major steps (after each sort pass for instance).

Answers:

1-) Selection Sort:

Initial Array: [5, 3, 2, 6, 4, 1, 3, 7]
After 1st Track: [5, 3, 2, 6, 4, 1, 3 **|** 7]
After 2nd Track: [5, 3, 2, 3, 4, 1 **|** 6, 7]
After 3rd Track: [1, 3, 2, 3, 4 **|** 5, 6, 7]
After 4th Track: [1, 3, 2, 3 **|** 4, 5, 6, 7]
After 5th Track: [1, 3, 2 **|** 3, 4, 5, 6, 7]
After 6th Track: [1, 2 **|** 3, 3, 4, 5, 6, 7]
After 7th Track: [1 **|** 2, 3, 3, 4, 5, 6, 7]

2-) Merge Sort:

Initial Array: [5, 3, 2, 6, 4, 1, 3, 7]
Divide: [5, 3, 2, 6] | [4, 1, 3, 7]
Divide: [5, 3] | [2,6] || [4, 1] | [3, 7]
Divide: [5] | [3] || [2] | [6] ||| [4] | [1] || [3] | [7]
Merge: [3, 5] || [2] | [6] ||| [4] | [1] || [3] | [7]
Merge: [3, 5] | [2, 6] ||| [4] | [1] || [3] | [7]
Merge: [2, 3, 5, 6] ||| [4] | [1] || [3] | [7]
Merge: [2, 3, 5, 6] ||| [1, 4] || [3] | [7]
Merge: [2, 3, 5, 6] || [1, 4] | [3, 7]
Merge: [2, 3, 5, 6] | [1, 3, 4, 7]
Merge: [1, 2, 3, 3, 4, 5, 6, 7]

3-) Quick Sort:

Pivot: 5 **Initial Array:** [5, 3, 2, 6, 4, 1, 3, **7**]
1st Sort Pass: [5, 3, 2, 6, 4, 1, 3, **7**] **New Pivot:** 3
2nd Sort Pass: [3, 2, 1, **3**, 5, 6, 4, **7**] **New Pivot:** 4
3rd Sort Pass: [3, 2, 1, **3, 4,** 5, 6, **7**] **New Pivot:** 6
4th Sort Pass: [3, 2, 1, **3, 4, 5, 6, 7**] **New Pivot:** 1
5th Sort Pass: [**1,** 3, 2, **3, 4, 5, 6, 7**] **New Pivot:** 2
6th Sort Pass: [**1, 2, 3, 3, 4, 5, 6, 7**]

C-) Find the asymptotic running times in big O notation of $T(n) = 2T(n-1) + n^2$, where $T(1) = 1$ by using the repeated substitution method. Show your steps in detail.

Answer:

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

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

…

$T(n) = 2^n*T(n-n)+[\sum_{i=0}^{n-1} 2^i] * n^2$

$T(n) = 2^n*T(0)+[2^n-1]*n^2$

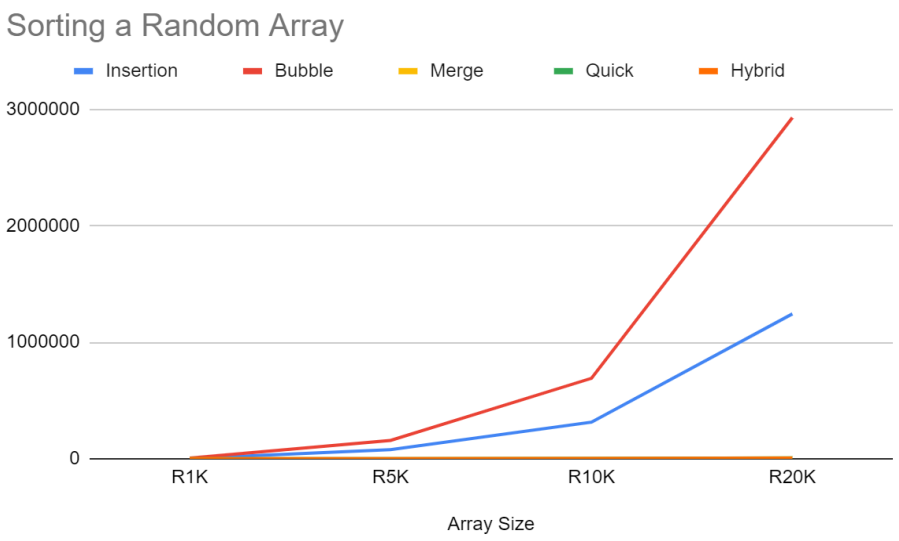
$T(n) = 2^n n^2-n^2$

Big O Notation: $O(2^n n^2)$

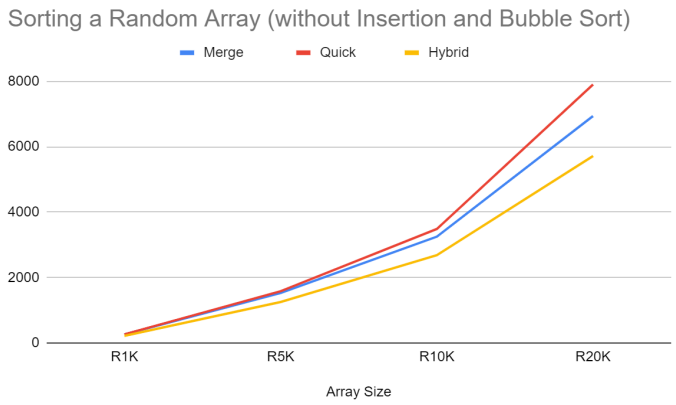
Question 3:

	Elapsed Time(ms)					Number of Comparisons					Number of Data Moves				
Array	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	3189	5516	263	264	217	255998	500094	8704	10365	14128	768994	797994	19952	668	22343
R5K	78059	157692	1527	1581	1253	6305425	12502290	55217	68880	87701	18921275	18916275	123616	3372	116914
R10K	313939	691349	3250	3486	2685	25224415	49997979	120433	157654	190826	75683245	75673245	267232	6683	281865
R20K	1243686	2928583	6938	7902	5717	100491769	200009370	260984	354094	416240	301495307	301475307	574464	13304	602744
A1K	56	111	194	809	1260	2202	9965	5848	149584	138306	7606	6606	19952	680	411419
A5K	318	734	1142	16145	24289	14677	64935	35942	3295182	2708518	49031	44031	123616	3473	8100792
A10K	659	192460	2328	60742	60020	31253	35540111	77179	12555086	12573167	103759	121473	267232	6932	65053
A20K	1378	1082324	4904	238978	237076	67376	200029998	165217	49797626	49835523	222128	322116	574464	13846	135165
D1K	6127	7194	198	1844	1011	497225	501500	6675	126743	134590	1492675	1491675	19952	679	234946
D5K	153394	176147	1100	36903	14240	12481890	12507498	39994	2635615	1971404	37450670	37445670	123616	3397	3305908
D10K	614324	717794	2337	129417	65256	49960491	50014986	84868	9138049	9159796	149891473	149909193	267232	6757	15676680
D20K	2461656	2878222	4876	527027	266132	199916814	200029995	161963	37186560	37231392	599770442	599870424	574464	13544	64007136

Sorting with random array:



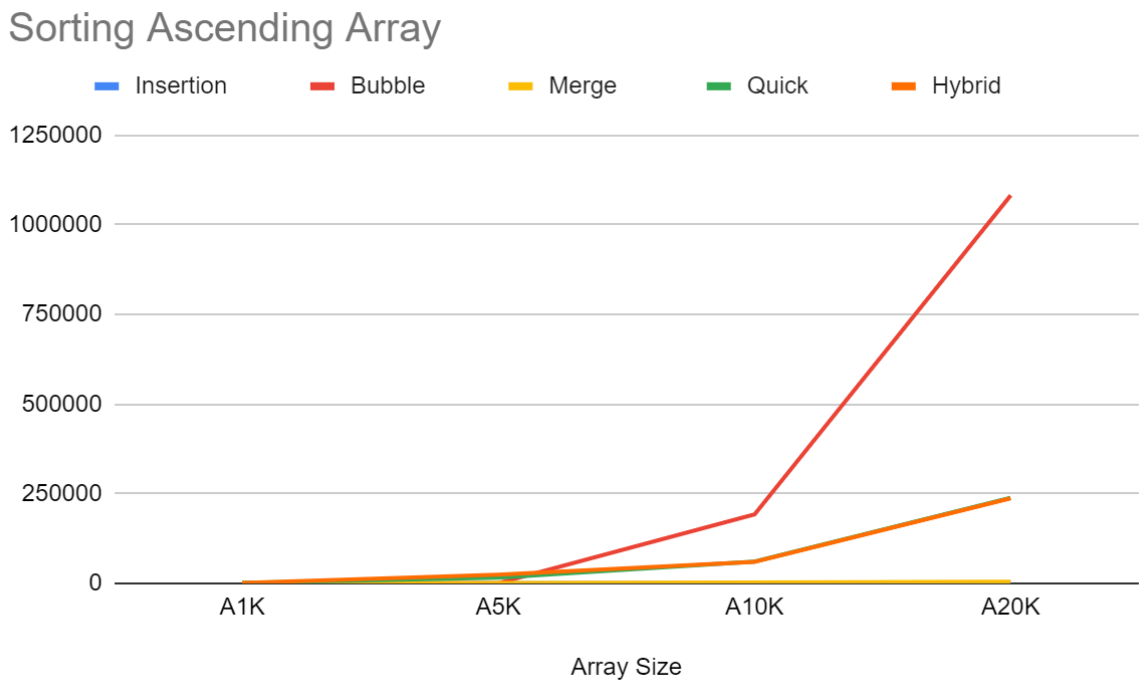
To see merge, hybrid and quicksort easily:

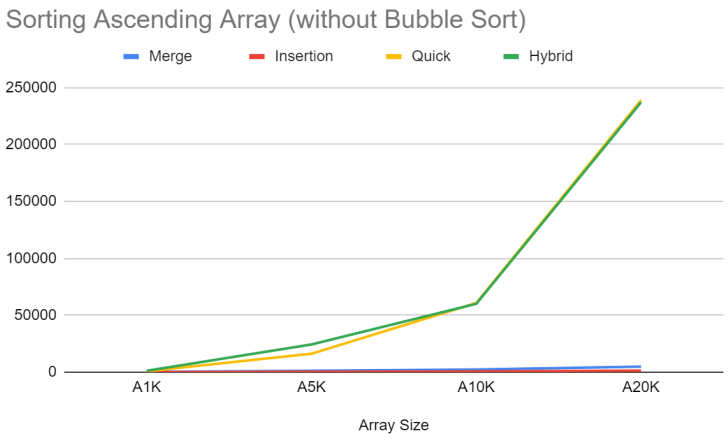


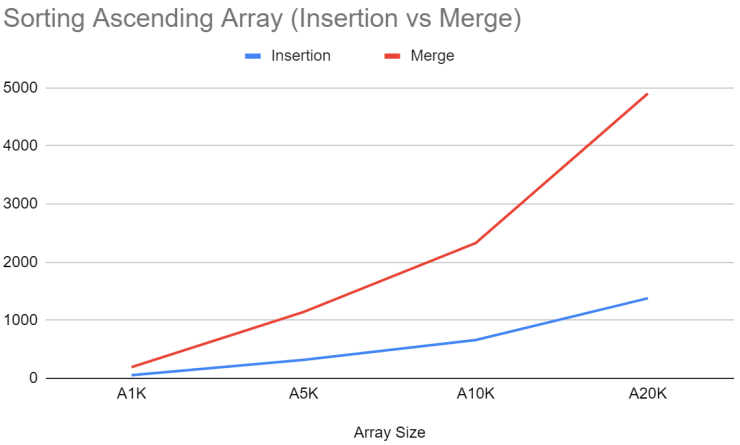
Analyze of Random Array Sorting (Above graphs):

From all these graphs I see that the algorithms' big o notations are accurate. It is not easy to see it with a small number of array elements. With random arrays it is more logical to use hybrid, quick or merge sort algorithms because those take much less time (as we can predict from average case big o notations of those algorithms).

Sorting an Ascending Array:





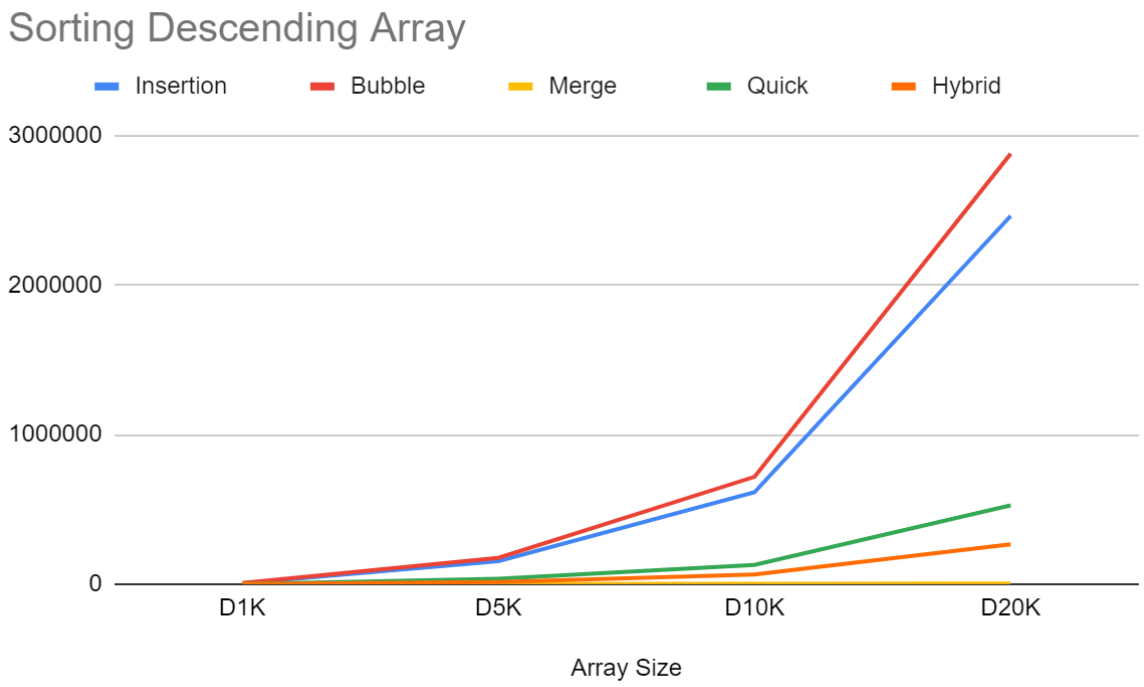


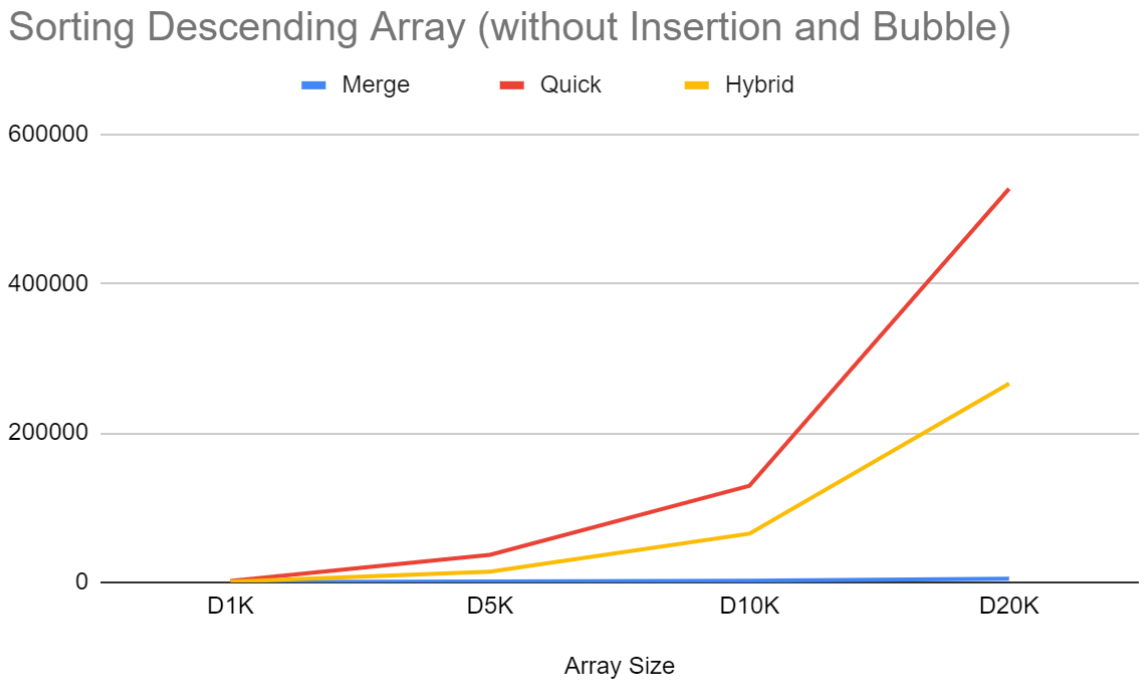
Analyze of Ascending Array Sorting(Above graphs):

From the first graph I can see that it is not logical to use bubble sort algorithms for ascending arrays. That is because this kind of array is a **worst case scenario** for bubble sort and hybrid sort uses bubble sort partially. For insertion sorting this kind of array is a **best case scenario** so it takes much less time than others. Quicksort and hybrid sort(which mostly acts like quicksort) require more time

than random array sorting. But it is because I don’t use the median of medians method during quicksort. Since these algorithms choose unoptimized pivots and require much more time than normal.

Sorting a Descending Array:





Analyze of Descending Array Sorting (Above Graphs):

For insertion sort and bubble sort, this kind of array is the **worst case scenario.** Because these algorithms always have to track all array elements every time to swap the smallest element with first elements but small elements are at the end of the array.

For merge sort it is okay as always. Because merge sort is amazing. Its average case and worst case's big o notations are the same.

It is bad for quick sort to have a sorted array because usually chosen pivot is inefficient in that kind of array. Because of that hybrid sort is inefficient (it is partially a quicksort algorithm).