Sertaç Derya 22003208 Section 3

Question 1

a) c=7 and n_0=$2\sqrt{2}$. After n_0=$2\sqrt{2}$, $7n^4$ is always bigger than f(n).

b)

I) Selection Sort:

[5, 3, 2, 6, 4, 1, 3, 7]->[5, 3, 2, 3, 4, 1, 6, 7]->[1, 3, 2, 3, 4, 5, 6, 7]->[1, 3, 2, 3, 4, 5, 6, 7]->
[1, 3, 2, 3, 4, 5, 6, 7] -> [1, 2, 3, 3, 4, 5, 6, 7] -> [1, 2, 3, 3, 4, 5, 6, 7]->[1, 2, 3, 3, 4, 5, 6, 7]

II) Merge Sort:

[3, 5, 2, 6, 4, 1, 3, 7]-> [3, 5, 2, 6, 4, 1, 3, 7]-> [2, 3, 5, 6, 4, 1, 3, 7]-> [2, 3, 5, 6, 1, 4, 3, 7]->
[2, 3, 5, 6, 1, 4, 3, 7]-> [2, 3, 5, 6, 1, 3, 4, 7]-> [1, 2, 3, 3, 4, 5, 6, 7]

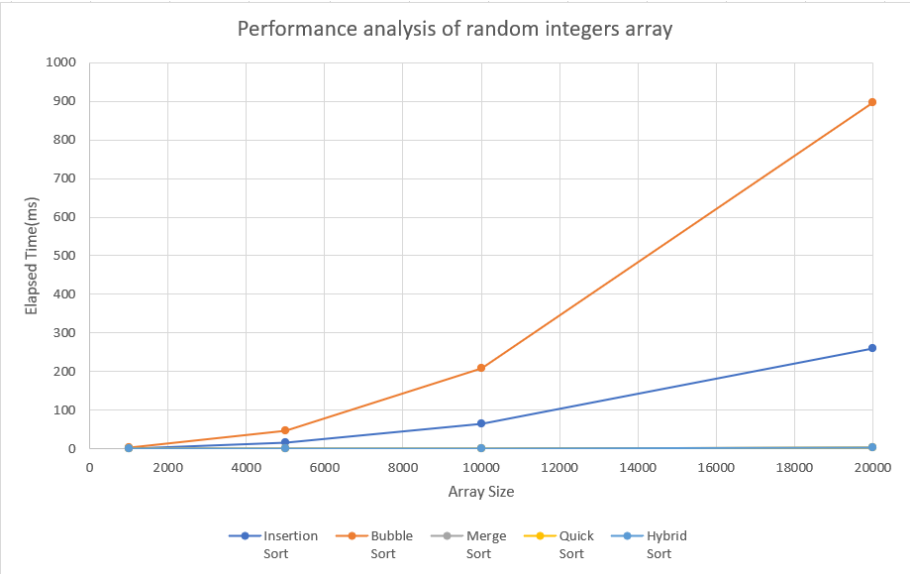
III) Quick Sort:

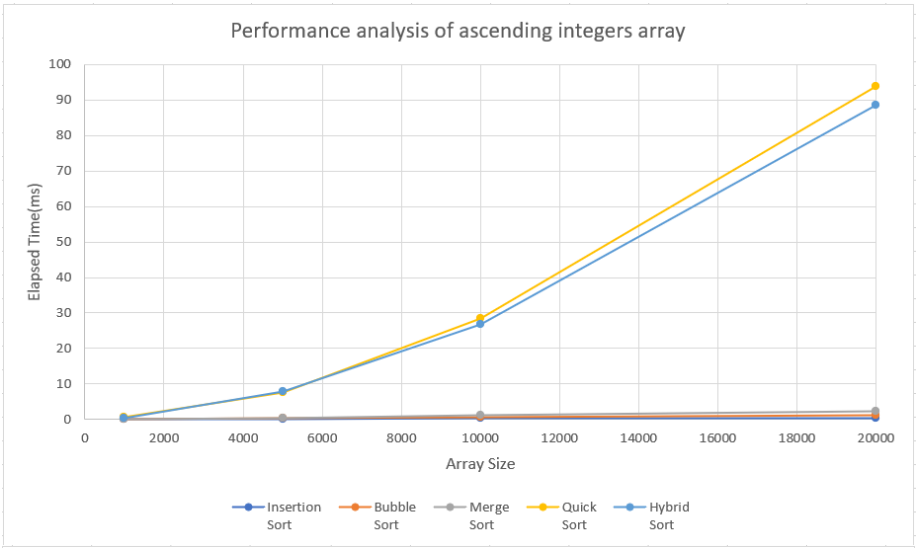
[5, 3, 2, 6, 4, 1, 3, 7]-> [5, 3, 2, 6, 4, 1, 3, 7]-> [1, 3, 2, 6, 4, 5, 3, 7]-> [1, 2, 3, 6, 4, 5, 3, 7]->
[1, 2, 3, 6, 4, 5, 3, 7]-> [1, 2, 3, 6, 4, 5, 3, 7]-> [1, 2, 3, 3, 4, 5, 6, 7]

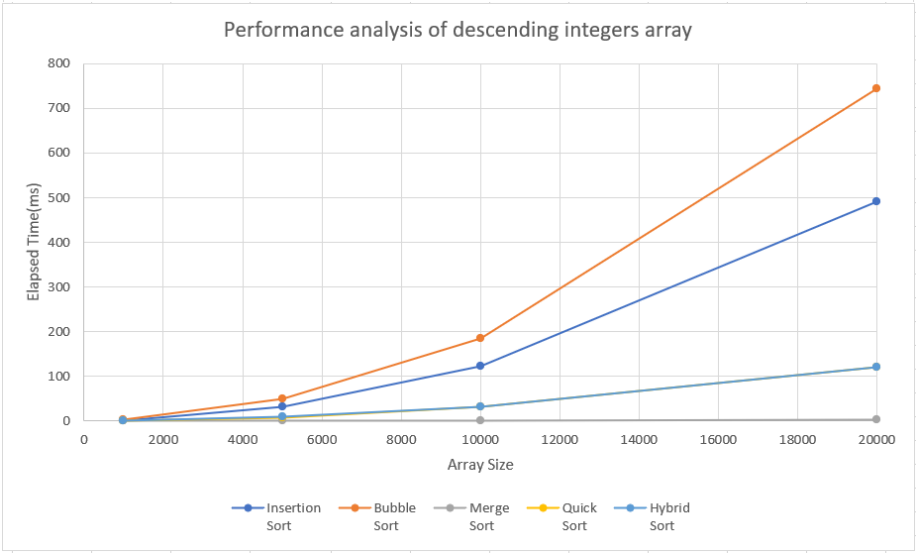
c) $T(n) = 2T(n-1) + n^2 \rightarrow T(n) = 2(2T(n-2) + (n-1)^2) + n^2 \rightarrow$
$T(n) = 2(2(2(\dots(2(2T(1) + 2^2) + 3^2)\dots) + (n-2)^2) + (n-1)^2) + n^2 \rightarrow$
$T(n) = 2^{n-1} + 2^n + 3^2.2^{n-2} + 4^2.2^{n-3} + \cdots + (n-1)^2.2 + n^2$
$from\ this\ result\ the\ bound\ can\ be\ found\ as\ O(2^n).$

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	0,9022	1,6961	0,355	0,0709	0,0814	254383	498759	8715	11565	14834	254383	760152	19952	18099	21423
R5K	15,5493	46,4468	0,5774	0,4063	0,4624	6378081	12493584	55182	66149	83354	6378081	19119246	123616	106389	123285
R10K	64,813	209,6797	1,2962	0,9451	1,0622	24905286	49993872	120407	161520	195649	24905286	74685861	267232	207477	241506
R20K	260,6615	896,7452	2,5833	1,9256	2,2443	1E+08	2E+08	260980	327821	395632	1E+08	3E+08	574464	529179	594954
A1K	0,0224	0,047	0,1094	0,5225	0,427	2750	8955	5804	180297	180701	2750	5253	19952	4914	4350
A5K	0,1036	0,2294	0,4377	7,4953	7,7765	17449	59922	35701	3832569	3835756	17449	37350	123616	27498	25935
A10K	0,236	0,5305	1,1197	28,5438	26,8147	37544	129909	76714	14029836	14037904	37544	82635	267232	57762	56859
A20K	0,4217	1,068	2,3501	93,9174	88,588	80202	279895	164265	53750190	53769002	80202	180609	574464	118254	120633
D1K	1,6169	2	0,0726	0,4718	0,4691	498273	499500	5641	136649	137179	498273	1491822	19952	225876	225522
D5K	32,593	49,9002	0,3898	8,3135	8,7439	12487900	12497497	34685	2727125	2731999	12487900	37448703	123616	4555434	4555854
D10K	123,7309	185,2051	0,8399	31,9897	31,2698	49972439	49994999	75040	10224437	10236353	49972439	1,5E+08	267232	17253426	17256195
D20K	490,7798	744,867	1,9152	120,4754	119,6463	2E+08	2E+08	160720	39493902	39521042	2E+08	6E+08	574464	67254843	67264158







c) These results show that throughout all the array types, **merge sort** performed the most reliably as its average case and worst case run times are the same. But for the ascending and descending array types, bubble and insertion sort seemed to perform a small amount better. **For the insertion and bubble sort**, randomized array performance was the worst out of all sorting types, and between the two, insertion sort was the faster one because the number of comparisons and data moves was lower. When sorting ascending arrays, the two algorithms performed much faster as the array was nearly sorted and the algorithms performed closer to their best case. When sorting descending arrays the algorithms performed poorly again and the average case result could be seen for both algorithms. Between these algorithms the sort times were almost the same because the time complexities are similar for average cases. **For Quick and Hybrid sort**, they were the fastest algorithms for sorting randomly created arrays. Quick sort was a tiny amount faster, but there wasn't a noticeable difference. However, while sorting ascending and descending arrays these algorithms performed poorly because these are considered edge cases and partition algorithm doesn't work as efficiently for these types of arrays and the complexity becomes $O(n^2)$. For ascending arrays, these algorithms were the worst out of all the algorithms and for descending arrays, they were better than insertion and bubble sort but were much slower than merge sort.