/**
* Title: Algorithm analysis & Sorting
* Author : GHULAM AHMED
* ID: 22101001
* Section : 2
* Homework : 1
* Description : description of your code */

CS 202 Assignment 1

Question 1

a) f(n) = 6n^4 + 9n^2 - 8
f(n) < c*n^4 for c = 7 and n >= 1
Therefore, f(n) = O(n^4)

b)

b)

Selection Sort :	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
	1	2	3	3	4	5	6	7
Merge Sort :	5	3	2	6	4	1	3	7
	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
Quick Sort	5	3	2	6	4	1	3	7
	3	2	1	3	5	4	6	7
	2	1	3	3	5	4	6	7
	1	2	3	3	5	4	6	7
	1	2	3	3	4	5	6	7

c)
$T(n) = 2T(n-1) + n^2$
$T(n-1) = 2T(n-2) + (n-1)^2$
Substituting T(n-1) into T(n):
$T(n) = 2[2T(n-2) + (n-1)^2] + n^2$

$T(n) = 4T(n-2) + 2(n-1)^2 + n^2$
$T(n-2) = 2T(n-3) + (n-2)^2$
Substituting T(n-2) into T(n):
$T(n) = 4[2T(n-3) + (n-2)^2] + 2(n-1)^2 + n^2$
$T(n) = 8T(n-3) + 4(n-2)^2 + 2(n-1)^2 + n^2$

$T(n) = 2^kT(n-k) + k(n-k+1)^2 + (k-1)(n-k+2)^2 + ... + 2(n-1)^2 + n^2$
When n-k = 1, using T(1) = 1,
$T(n) = 2^{(n-1)} + n^4/4 - n^3/2 + n^2/4$

Therefore, the asymptotic running time in big O notation is **$O(2^n)$**, since the exponential term $2^{(n-1)}$ dominates the polynomial term $n^4/4$ for large values of n.

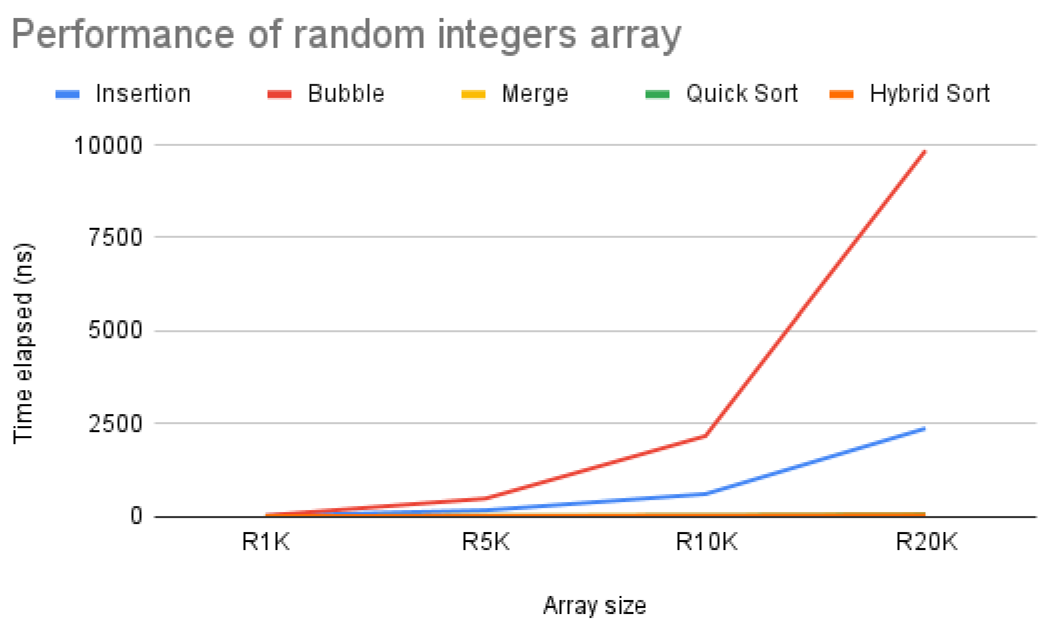
Question 3

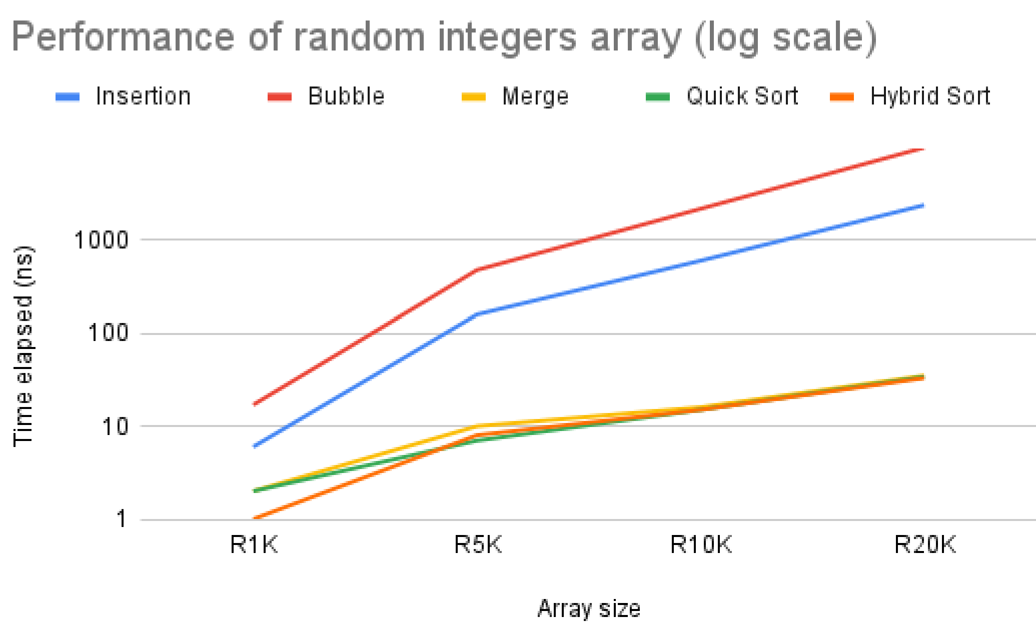
Results

Array	Elapsed Time (ns)					Number of comparison					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	6	17	2	2	1	259171	499500	17668	18558	17562	260170	777513	19952	20155	20155
R5K	158	474	10	7	8	6322159	12497500	111998	125147	120152	6327158	18966477	123616	132892	132892
R10K	597	2161	16	15	15	25086666	49995000	244055	298524	288529	25096665	75259998	267232	314078	314078
R20K	2366	9851	35	34	33	99896806	199990000	528028	594202	574205	99916805	299690418	574464	625153	625153
A1K	7	19	1	2	1	248987	499500	17699	20416	19425	249986	746970	19952	21963	21963
A5K	177	523	7	7	7	6230238	12497500	112024	128422	123427	6235237	18690717	123616	136125	136125
A10K	688	2390	15	15	15	24688830	49995000	244123	277940	267945	24698829	74066532	267232	293384	293384
A20K	2442	9565	32	33	33	100740976	199990000	528176	642123	622127	100760975	302222934	574464	673118	673118
D1K	6	18	1	1	1	238766	499500	17670	18273	17276	239765	716298	19952	19804	19804
D5K	161	506	8	8	7	6270156	12497500	111967	127944	122947	6275155	18810468	123616	135644	135644
D10K	648	2218	15	15	15	25241701	49995000	244006	290567	280570	25251700	75725103	267232	306082	306082
D20K	2454	9949	33	35	36	100498476	199990000	528185	674169	654173	100518475	301495428	574464	705005	705005

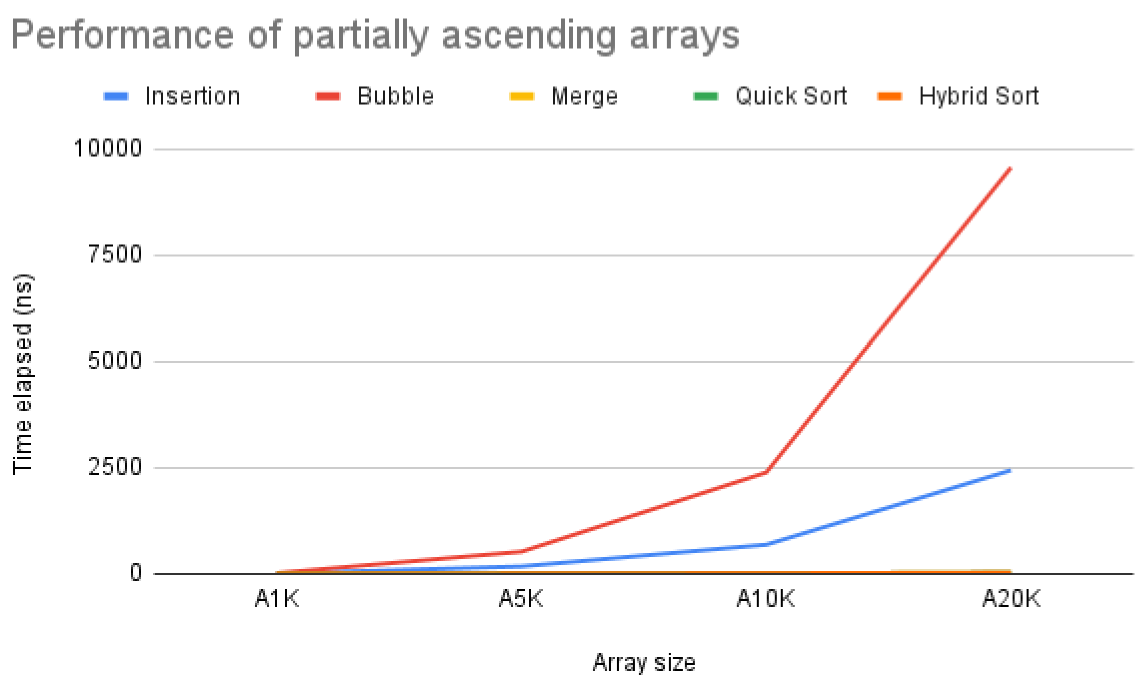
Performance charts

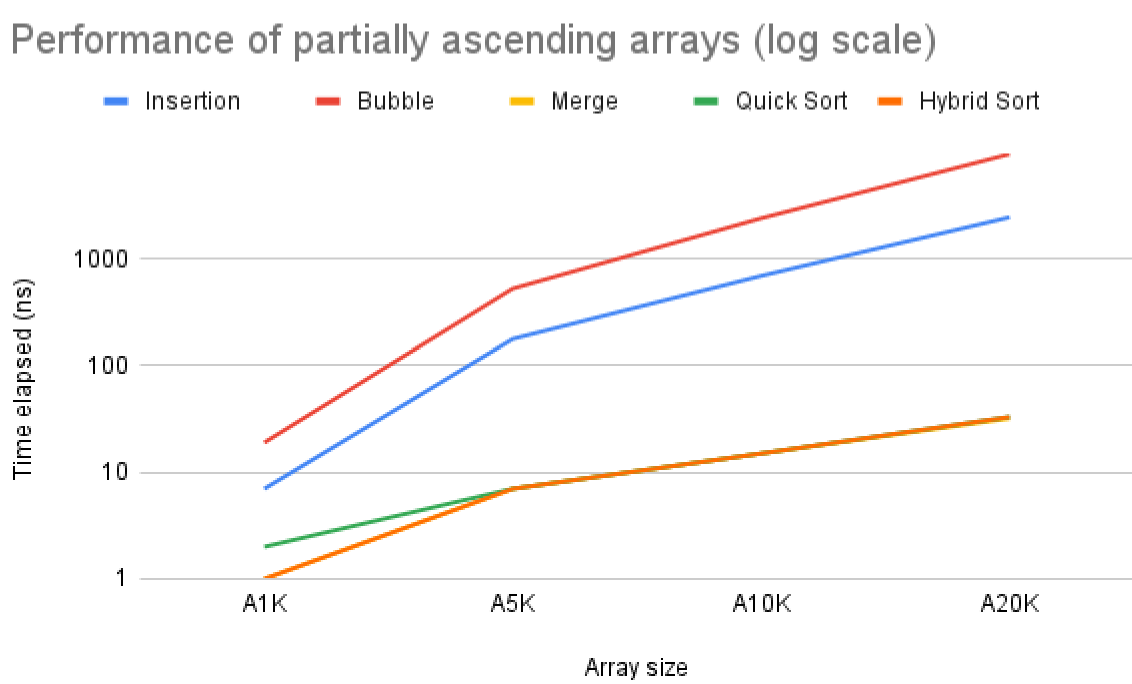
Random integers array



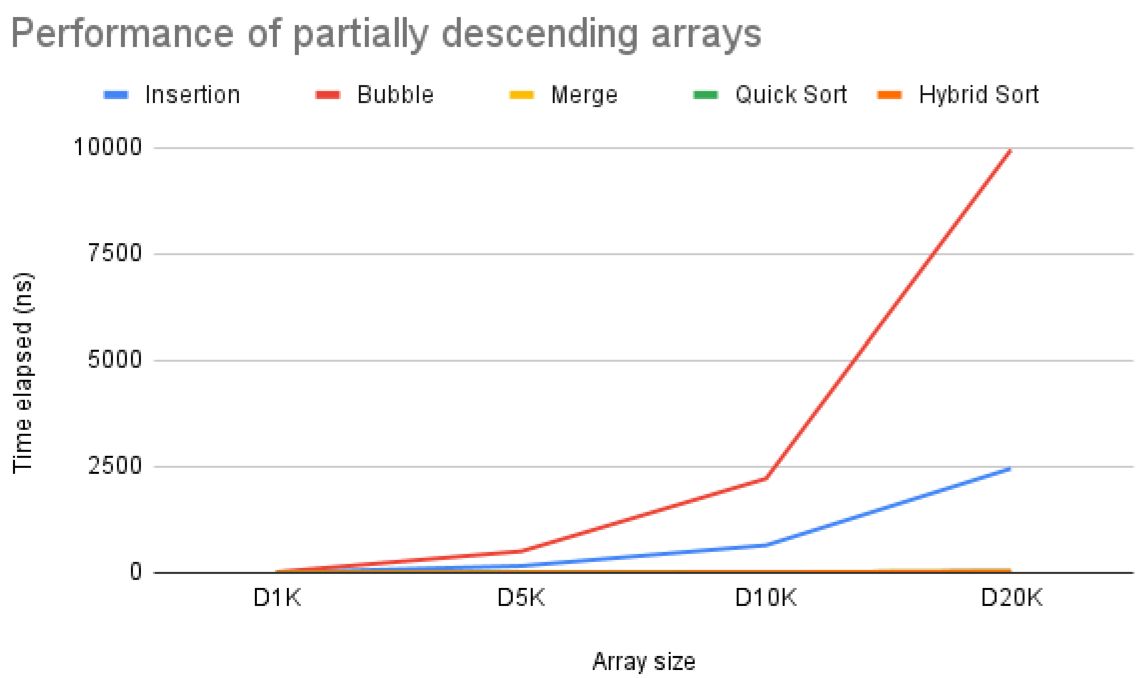


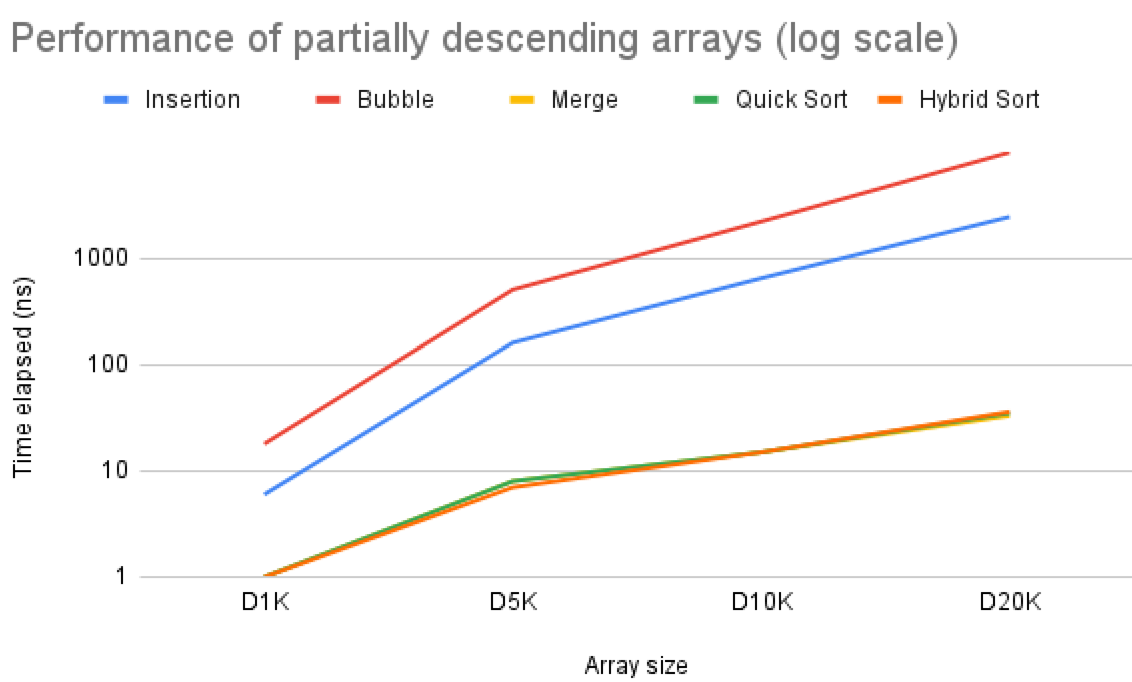
Partially ascending arrays





Partially descending arrays





Analysis

Bubble sort performs worst in all aspects (elapsed time, comparisons, data moves). For random integer arrays, hybrid sort is the best performer as it combines the efficiency of bubble sort for small array sizes and divide and conquer efficiency of quick sort for large inputs. For partially ascending arrays, merge and hybrid sort outperform quicksort for small array sizes, but the performance is almost equal for larger sizes. For partially descending arrays merge sort, quick sort, and hybrid have nearly the same performance in terms of time required. In terms of comparisons made, bubble and insertion sort were very inefficient, so their performance can be ignored. On the other hand, merge sort had the least number of comparisons, followed by hybrid sort and quicksort. For comparisons, bubble sort, and insertion sort have a comparison complexity of $O(n^2)$, making them the least efficient of the five algorithms. Quicksort, merge sort, and hybrid sort have a comparison complexity of $O(n \log n)$, making them more performance efficient than insertion and bubble sort. In terms of data moves, bubble sort, and insertion sort have a data move complexity of $O(n^2)$, making them the least efficient of the five algorithms. Quicksort and hybrid sort have a data move complexity of $O(n \log n)$, making them more efficient than insertion and bubble sort. Merge sort has a data move complexity of $O(n)$, making it the most efficient of the five algorithms. Quicksort and hybrid had the same number of data moves as they shared similar algorithms. Overall, when analyzing the runtimes, number of comparisons, and number of data moves for the five sorting algorithms, hybrid sort was found to be the most efficient sorting algorithm. It also made fewer comparisons and data moves than the other four algorithms.