CPE 360 HW3

//==========Question 1==========

Time Complexities:

Using the following function:

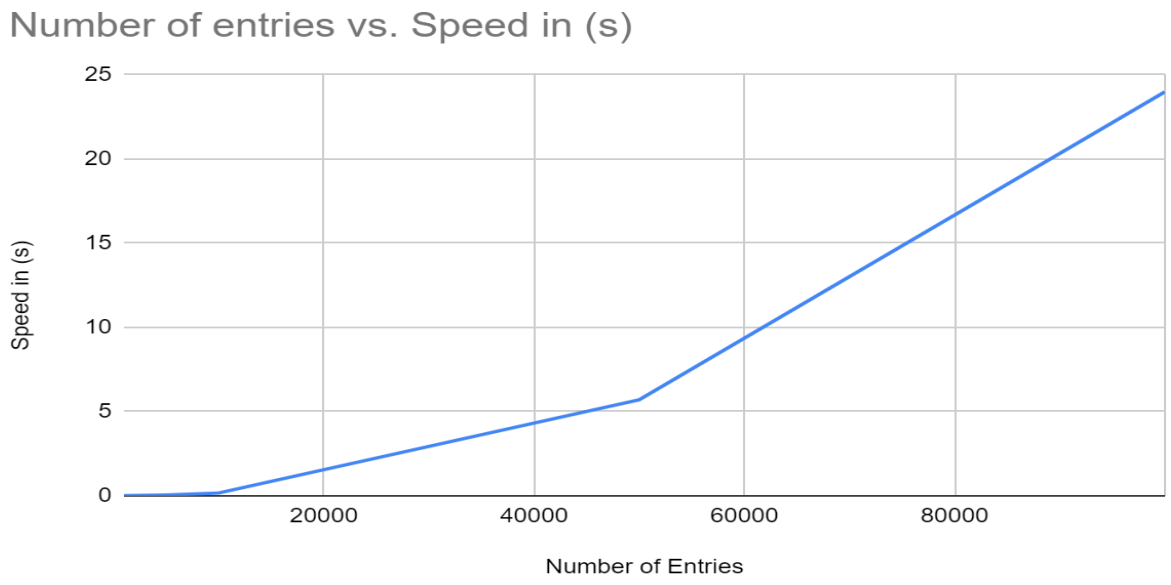
```
void measureTime(void (*sortFunc)(int[], int), int arr[], int n)
{
    clock_t start = clock(); // Start timer
    sortFunc(arr, n); // Call the sorting function
    clock_t end = clock(); // End timer

    // Calculate and print the time taken by the sorting function
    double timeTaken = (double)(end - start) / CLOCKS_PER_SEC;
    cout << "Time taken: " << timeTaken << " seconds" << endl;
}
```

```
measureTime(bubbleSort, arr, n);
```

I was able to gather 10 data points to create a graph of each sorting algorithm. My computer is running an AMD ryzen 7 and swap speed is about 0.5745323741 nanoseconds.

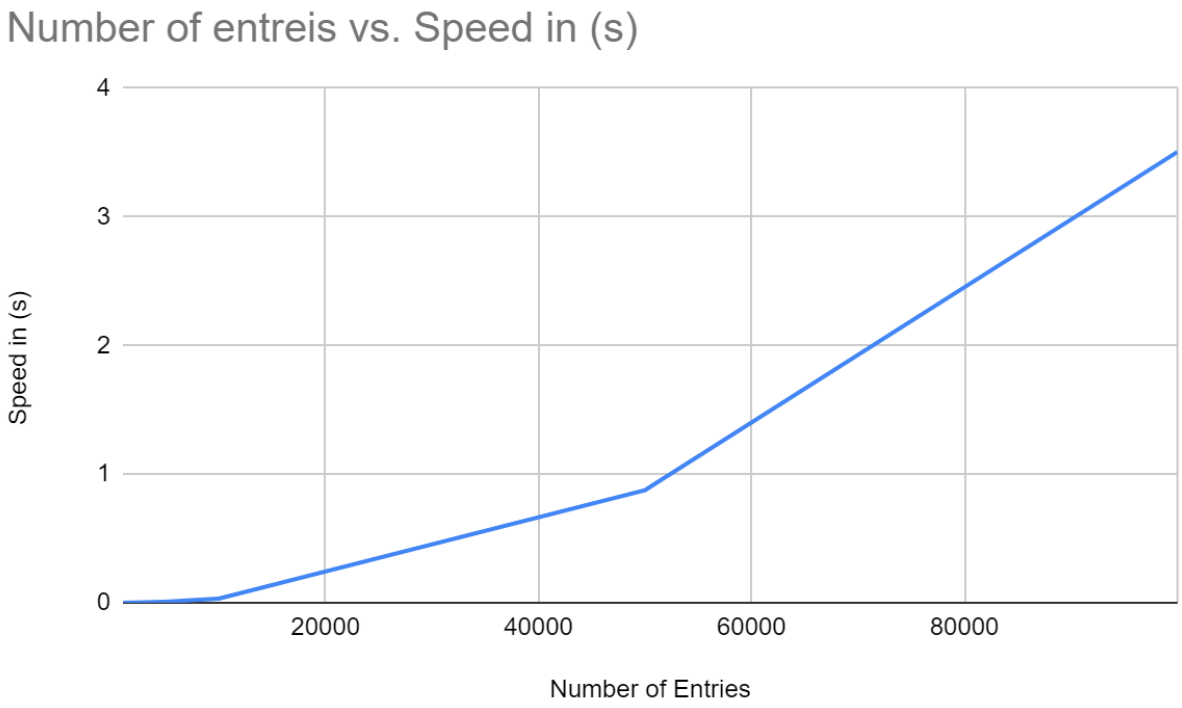
Bubble sort:
Graph:



Data:

Speed in (s)	Number of entries
0.0023985	1000
0.041	5000
0.165	10000
5.708	50000
23.985	100000

Insertion Sort:
Graph:

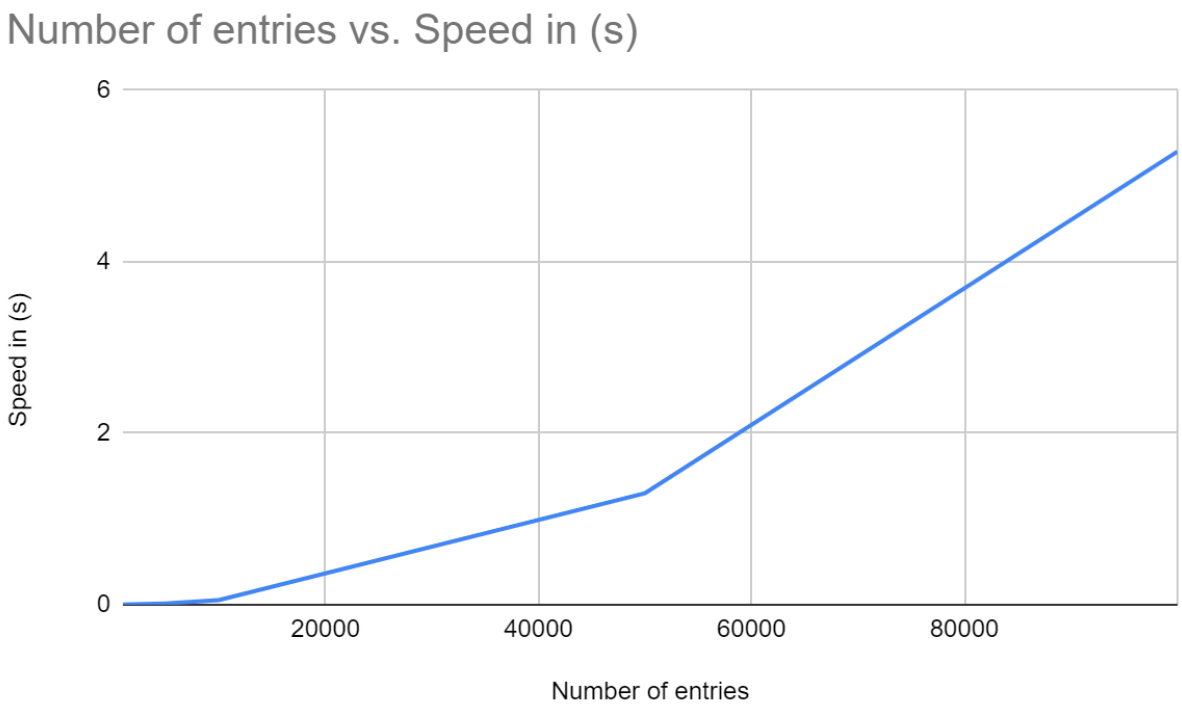


Data:

Speed in (s)	Number of entries
0	1000
0.008	5000
0.032	10000
0.875	50000
3.505	100000

Selection Sort:

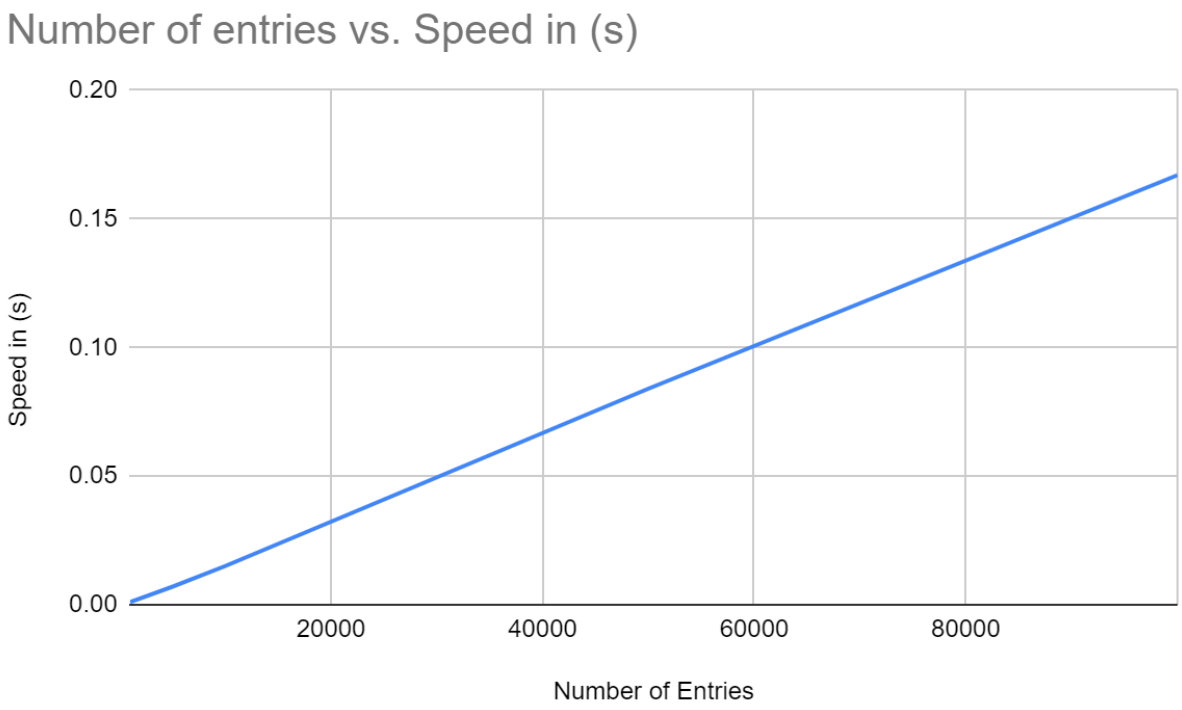
Graph:



Data:

Speed in (s)	Number of entries
0.001	1000
0.014	5000
0.0529	10000
1.2999	50000
5.28399	100000

Merge Sort:
Graph:

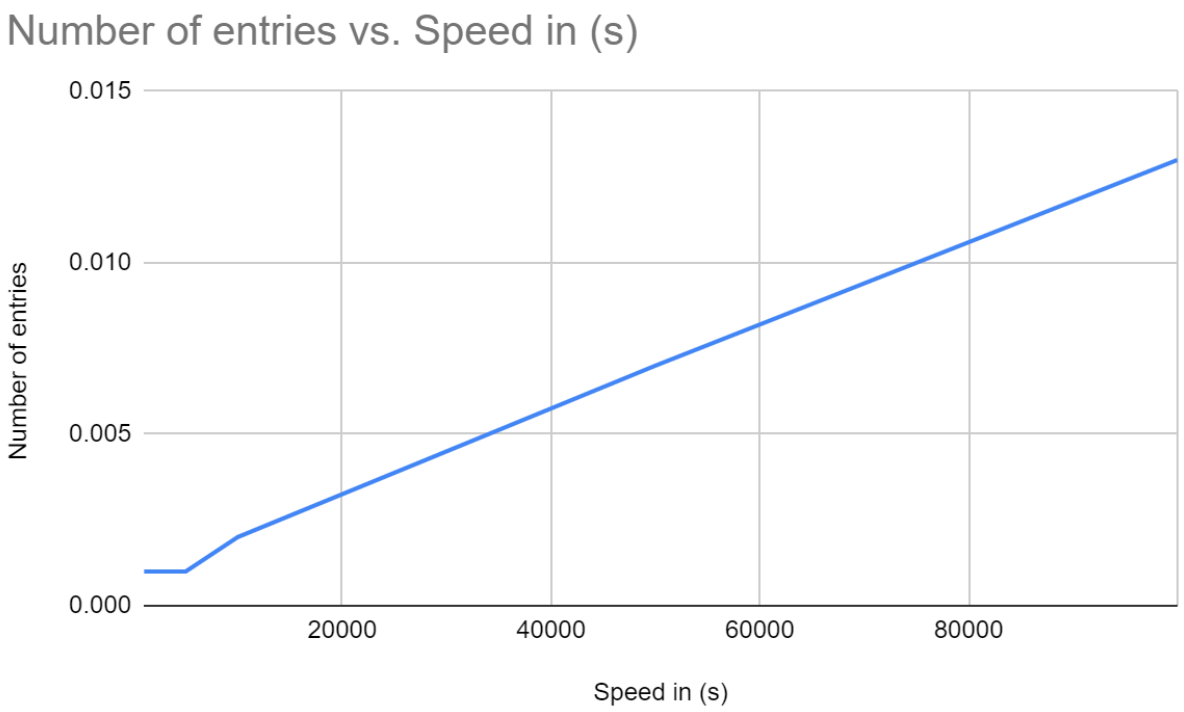


Data:

Speed in (s)	Number of entries
0.001	1000
0.007	5000
0.015	10000
0.084	50000
0.167	100000

Radix Sort:

Graph:



Data:

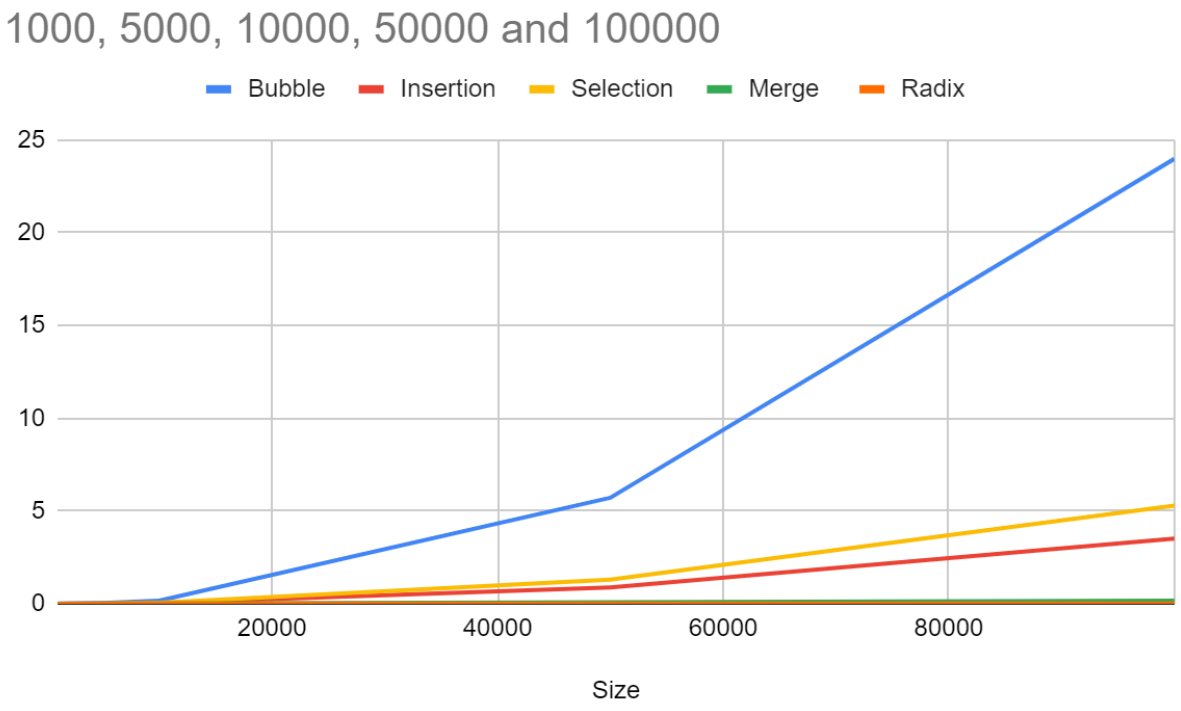
Speed in (s)	Number of entries
0.001	1000
0.001	5000
0.002	10000
0.007	50000
0.013	100000

All:

Data:

Size	Bubble	Insertion	Selection	Merge	Radix
1000	0.0023985	0	0.001	0.001	0.001
5000	0.041	0.008	0.014	0.007	0.001
10000	0.165	0.032	0.0529	0.015	0.002
50000	5.708	0.875	1.2999	0.084	0.007
100000	23.985	3.505	5.28399	0.167	0.013

Graph:



//==========Question 2==========

Based of the graphs:

Bubble sort: O(n^2)

Insertion sort: O(n^2)

Selection sort: O(n^2)

Merge sort: O(nLogn)

Radix sort: O(an) if sorted o(n)

//==========Question 3==========

```
std::vector<int> items;

    for (int i = 0; i < 8000000; i++) {

        items.push_back(rand());

    }
```

For this section I have switched my arrays to vectors because an array can only up to 1,000,000 values an a vector can hold 2^32

Test	1	2	3	4	5	6	7	8	9	10	Average
Merge Sort for 8 million	14.659	14.601	14.605	14.666	14.95	14.43	14.383	14.385	14.787	15.019	14.6485
Radix for 8 million	1.104	1.146	1.923	1.102	1.393	1.381	1.845	1.245	1.225	1.197	1.3561

For an unsorted array the only sorting algorithms that would even measure were radix and merge showing that radix is the clear winner, the other ones can not do 8 million entries because they are n^2 and i would probably be dead by time they sorted an array of 8 million.

//==========Question 4 ==========

Test	1	2	3	4	5	6	7	8	9	10	Average
Bubble	0.025794	0.02536	0.024656	0.025939	0.025618	0.025981	0.024594	0.024505	0.025717	0.024853	0.0253017
Insertion	0.04024	0.047868	0.041274	0.042996	0.04064	0.041841	0.041807	0.047565	0.040315	0.042311	0.0426857
Selection	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
Merge	12.669	12.221	12.237	12.44	12.18	12.521	12.488	12.379	12.294	12.302	12.3731
Radix	1.3017	1.269	1.275	1.274	1.231	1.292	1.249	1.263	1.246	1.303	1.270

	43	168	628	57	99	269	842	702	872	677	9461

Bubble sort would be the fastest based on the data because in its best cases which is fully ordered array it would be O(n)