Compiling and executing the table.cpp file produces the following output (each algorithm is tested 30 times for each value of n — you can adjust this number):

```
[antikytechanism:HW2 KoreyHuskonen$ ./a.out

n         Bubble          Insertion       Selection       Merge           Quick           Count           Radix           Heap
10        5372            372             559             786             633             1136            1073            796
100       37999           10584           20863           11476           9521            3073            7499            13852
500       624916          181876          355430          58047           48579           11840           39382           77230
5000      82577017        18577507        33199774        762674          666205          121631          480072          1069883
25000     2161997656      427096331       767775311       4175642         3698242         570791          2278451         5872256
Trials: 30
antikytechanism:HW2 KoreyHuskonen$
```

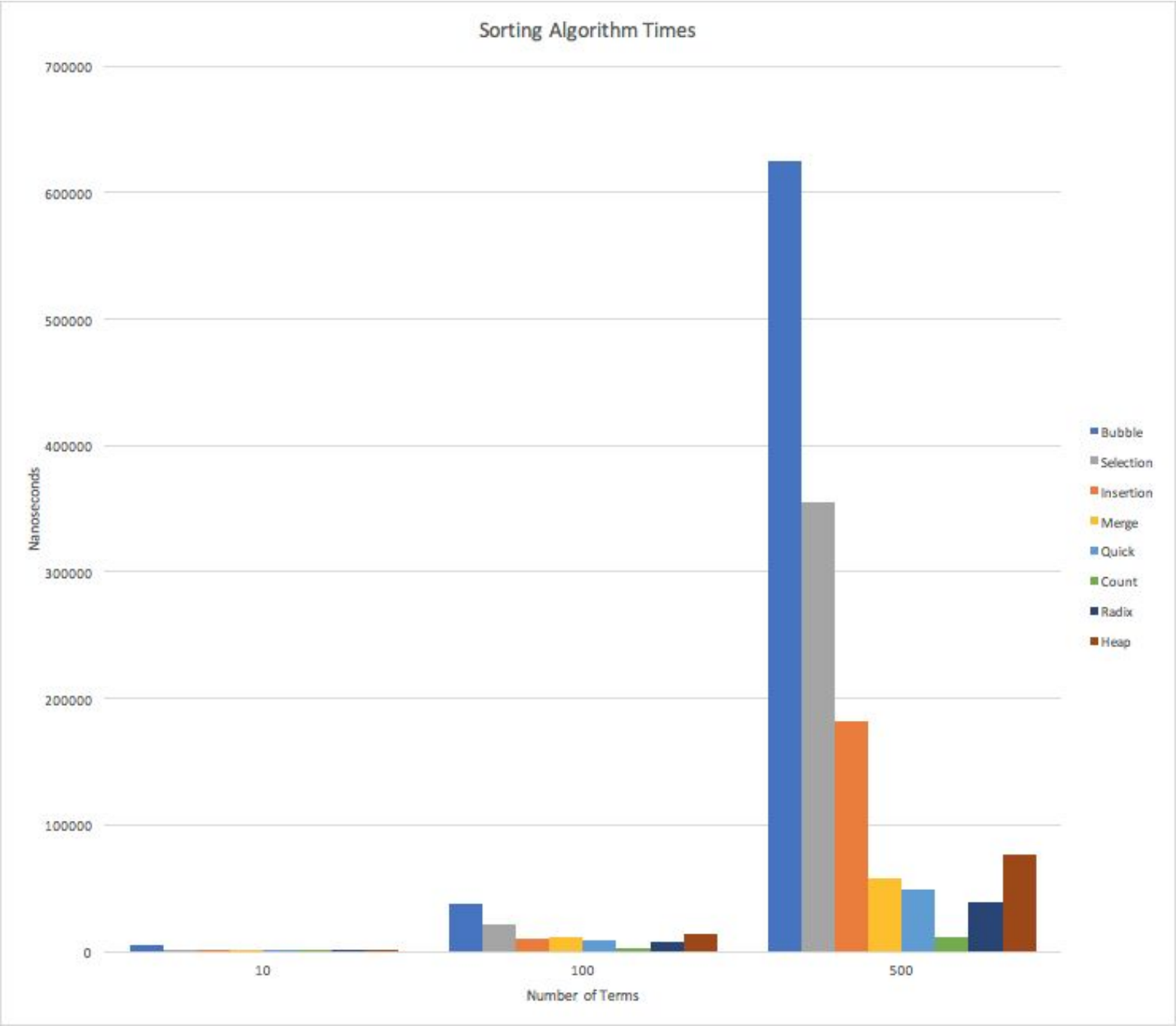
n	10	100	500	5000	25000
Bubble	5372	37999	624916	82577017	2161997656
Insertion	372	10584	181876	18577507	427096331
Selection	559	20863	355430	33199774	767775311
Merge	786	11476	58047	762674	4175642
Quick	633	9521	48579	666205	3698242
Count	1136	3073	11840	121631	570791
Radix	1073	7499	39382	480072	2278451
Heap	796	13852	77230	1069883	5872256

As expected, the $O(n^2)$ algorithms — bubble, insertion, and selection — performed much worse than the O(nlogn) algorithms for large values of n. Insertion and selection sort were quicker than the O(nlogn) algorithms for n = 10, as expected. The only result that surprised me was selection sort. I thought it'd be quicker than insertion sort, but insertion sort beat it every time.

Of course, count sort performed the best since it is O(n+k) where k was only at most 25000. So I switched the possible range from n to n^2 ($k = n^2$) and this happened:

```
n         Bubble          Insertion       Selection       Merge           Quick           Count           Radix           Heap
10        9315            360             585             832             638             1365            1177            812
100       36254           10223           20572           11332           9067            66572           9323            12980
500       622702          191001          353913          60936           49696           1373541         63381           76642
5000      80124263        18037081        31907568        773756          675323          165257059       786587          1032947
25000     2285089754      445835613       802305311       4581050         4043270         4582456340      4344980         6097550
Trials: 20
```

Count sort blew up, and here we see the advantage radix sort has over count sort. Unfortunately, both count sort and radix sort only work for integers, so it appears that the best general purpose sorting algorithm is quick sort.



I only graphed the first 3 sizes to avoid problems with the scale