Radix Sort

Sreevatsa Nukala

March 1, 2020

Post AP Data Structures and Algorithms - Mr. Bradley

1 Introduction

In this project, I implemented a radix sort, in which the number of bins used can be modified. I also implemented a standard quick sort to compare runtimes against. I used 4 data sets to test runtimes, 2 found online, 2 randomly generated. One of the data sets found online contain expenditure on food in Spanish households, the other contains the number of kids born per day in every state from 1969 to 1978. Each of the randomly generated data sets has 1000000 values, but one has values up to 4 digits and the other up to 9 digits. The goal is to find out when radix sort is the better sorting method to use.

2 Data

Table 1: Runtimes of Radix Sort vs. Quick Sort for FoodExpenditure.csv

FoodExpenditure.csv		
No. of Bins(Radix)	**Radix Runtime(s)**	**Quick Runtime(s)**
2	32.4	10.8
3	23.2	11.1
4	18.8	10.9
5	16.7	11.3
6	16.8	11
7	13.7	10.5
8	19.9	10.7
9	17.5	11.5
10	13.1	11.3
11	11.6	10.9
12	12.1	10.6
13	12.3	11.4
14	12.3	11.1
15	12.2	10.8

Figure 1: Runtimes of Radix Sort vs. Quick Sort for FoodExpenditure.csv

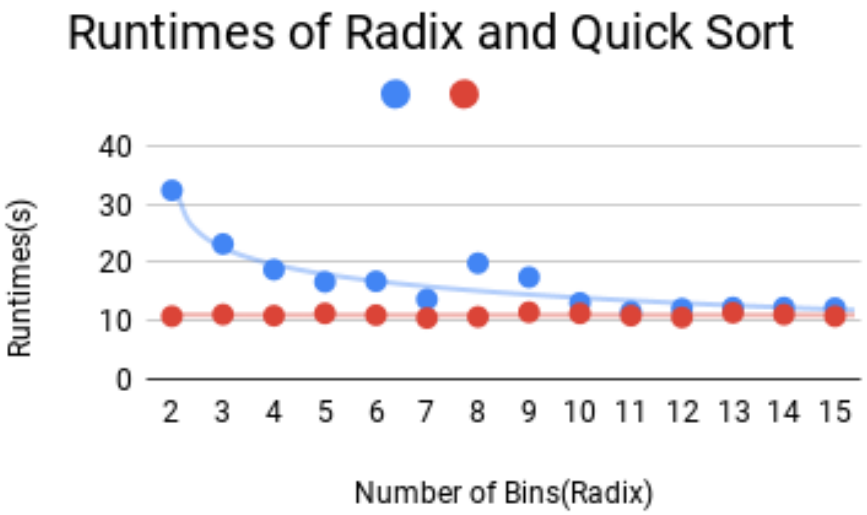


Table 2: Runtimes of Radix Sort vs. Quick Sort for Birthdays.csv

Birthdays.csv		
No. of Bins(Radix)	**Radix Runtime(s)**	**Quick Runtime(s)**
2	19.87	160.61
3	12.63	167.58
4	10.93	160.44
5	9.22	160.01
6	9.49	160.53
7	7.56	160.23
8	7.58	159.37
9	7.53	159.85
10	7.83	158.95
11	7.82	160.46
12	7.79	161.26
13	5.84	160.78
14	5.94	160.26
15	5.86	159.96

Figure 2: Runtimes of Radix Sort vs. Quick Sort for Birthdays.csv

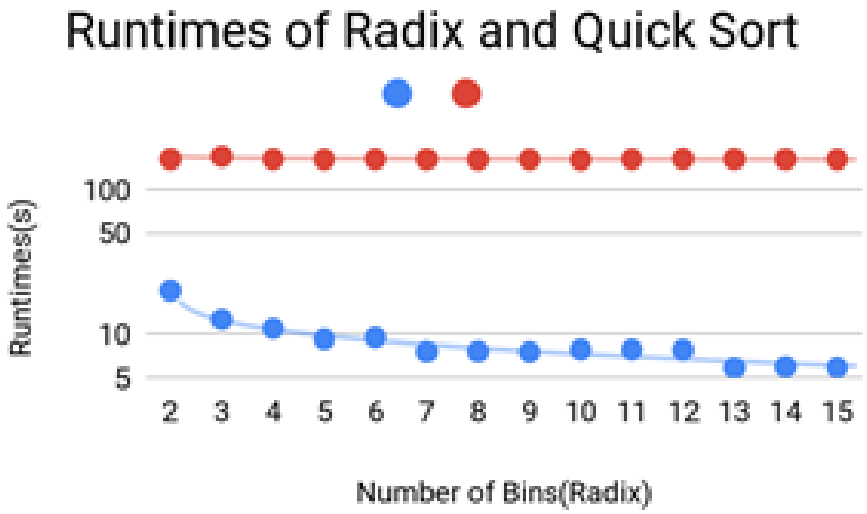


Table 3: Runtimes of Radix Sort vs. Quick Sort for Random Small Values

Randomly Generated Data (Small Values)		
No. of Bins(Radix)	**Radix Runtime(s)**	**Quick Runtime(s)**
2	21.6	22.5
3	17.3	21.2
4	12.3	21.5
5	11.7	23.4
6	10.3	21.4
7	10.5	19.7
8	9.7	20.3
9	7.4	20.5
10	8.7	20.8
11	5.6	21.2
12	5.2	22.6
13	6.4	21.8
14	5.7	22.3
15	5.9	22.1

Figure 3: Runtimes of Radix Sort vs. Quick Sort for Random Small Values

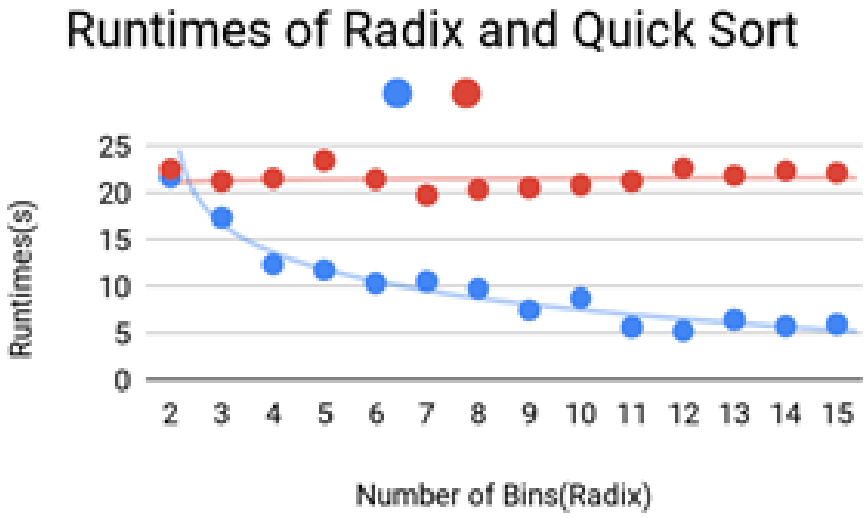
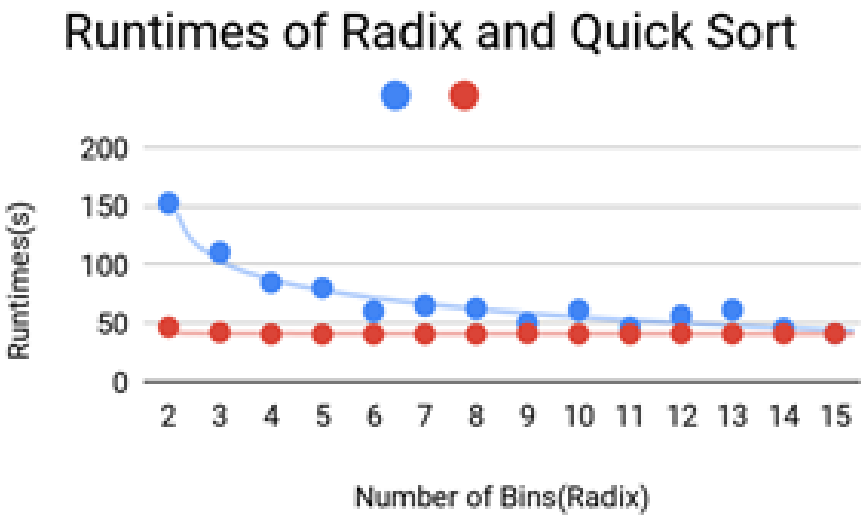


Table 4: Runtimes of Radix Sort vs. Quick Sort for Random Large Values

Randomly Generated Data (Large Values)		
No. of Bins(Radix)	**Radix Runtime(s)**	**Quick Runtime(s)**
2	152.57	46.6
3	110.48	42.23
4	84.56	40.66
5	79.99	40.5
6	60.11	40.52
7	64.92	40.62
8	62.39	40.56
9	50.26	41.38
10	61	40.66
11	46.45	40.68
12	56.33	41.1
13	61.3	40.86
14	45.64	41.09
15	41.08	41.14

Figure 4: Runtimes of Radix Sort vs. Quick Sort for Random Large Values



3 Conclusion

In this project, I implemented a radix sort, in which the number of bins used can be modified. I also implemented a standard quick sort to compare runtimes against. I used 4 data sets to test runtimes, 2 found online, 2 randomly generated. One of the data sets found online contain expenditure on food in Spanish households, the other contains the number of kids born per day in every state from 1969 to 1978. Each of the randomly generated data sets has 1000000 values, but one has values up to 4 digits and the other up to 9 digits. The goal is to find out when radix sort is the better sorting method to use. In all the data we see that the runtime of the radix sort decreases logarithmically as the number of bins increases. In FoodExpenditures.csv, large data values with up to 9 digits exist, as well as in the data set of large, randomly generated values. In both of these, we see that quick sort runs faster, but radix sort displays asymptotic behavior near the run time the quick sort ran at. For the other two data sets, we see that radix sort runs faster because the data sets only contain data values that have a maximum of 4 didgits. No optimal bin number seems to exist based on this data, but as the number of bins gets increasingly larger, the less drastic a change we see in runtime. We can conclude that it is better to use radix sort with small data values.