Algorithms Homework 5 Code Report

Ryan Rau [redacted]

March 20 2020

1 Problem

Given 10 million integers between the range of 0 and $2^{32} - 1$ which algorithm should be used to efficiently sort those values. Quick sort or radix sort?

2 Solution

2.1 Algorithm

For the quick sort part, I started with implementing a version of quick sort based off the following pseudo code that was found in the book.

```
Partition(A, p, r)
    x = A[r]
    i = p - 1
    for j = p to r - 1
        if A[j] <= x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[j]
    return i + 1

Quicksort(A, p, r)
    if p < r
        q = Partition(A, p, r)
        Quicksort(A, p, q - 1)
        Quicksort(A, q + 1, r)
```

With that as a base, I then made slight modifications so that it would work in Java. Testing with small data sets until I got it fully working.

For the radix sorting part, I started with the code that I wrote for Homework 4 which was based off the following pseudo code.

```
RADIX-SORT(A, d)
    for i to d
        use a stable sort to sort array A on digit i

COUNTING-SORT(A, B, k)
    let C[0..k] be new array
    for i = 0 to k
        C[i] = 0
    for j = 1 to A.length
        C[A[j]] = C[A[j]] + 1
    for i = 1 to k
        C[i] = C[i] + C[i - 1]
    for j = A.length downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

With that as a base, I began to modify it to sort 4 base 256 digits allowing for a time complexity closer to

$$\Theta(4(n + 255))$$

2.2 Time Complexity

The average case time complexity for quick sort is $\Theta(nlgn)$ with a worst case of $\Theta(n^2)$. For radix sort give that the number can be split into a 4 digit base 256 number the time complexity should be $\Theta(4(n + 255))$. Given a n of 10 million on paper radix sort on average should be quicker.

2.3 Testing

To test my sorting algorithms I made it so that my code would create 3 sets of data and then sort each of those sets 3 times using both quick and radix sort, recording the times each time. I also wrote a bit that would allow for a data file to be entered and sorted recording those times as well. When testing I ran the code on both my MacBook and on turing and the differences in there times will be listed in the next section.

When testing the code you can either specify a input and output text file which will be sorted and display the running times for each of the sorting methods. If no files are specified the program will create 3 data files and sort them with both sorting methods displaying the run times.

The following are example commands for how to run my program.

Will sort given input file and store the sorted result to output recording the sorting time.

```
javac Homework5.java
java Homework5 input.txt output.txt
```

NOTE provided text file needs be of length of 10 million and have values within the specified range for the project

Will generate 3 random data files and sort them recording the times.

```
javac Homework5.java
java Homework5
```

2.4 Results

Below are the time results for the provided data0.txt file along with 3 other randomly generated data files. All times are in milliseconds.

data0.txt, the file that was provided

Quick Sort		
Run	Turing Time	MacBook Time
1	1172	1024
2	1148	982
3	1165	992
Average:	**1162ms**	**999ms**

Radix Sort		
Run	Turing Time	MacBook Time
1	1459	1069
2	1412	1127
3	1488	1067
Average:	**1453ms**	**1087ms**

data1.txt, randomly generated data

Quick Sort		
Run	Turing Time	MacBook Time
1	1115	1133
2	1141	1107
3	1147	1115
Average:	**1134ms**	**1118ms**

Radix Sort		
Run	Turing Time	MacBook Time
1	1487	1204
2	1452	1064
3	1413	1069
Average:	**1451ms**	**1112ms**

data2.txt, randomly generated data

Quick Sort		
Run	Turing Time	MacBook Time
1	1136	1112
2	1136	1118
3	1131	1124
Average:	**1134ms**	**1118ms**

Radix Sort		
Run	Turing Time	MacBook Time
1	1450	1067
2	1461	1058
3	1427	1091
Average:	**1446ms**	**1072ms**

data3.txt, randomly generated data

Quick Sort		
Run	Turing Time	MacBook Time
1	1127	1109
2	1118	1110
3	1112	1112
Average:	**1119ms**	**1110ms**

Radix Sort		
Run	Turing Time	MacBook Time
1	1394	1087
2	1393	1054
3	1386	1074
Average:	**1391ms**	**1072ms**

Overall averages

Sort Type	Turing Time	MacBook Time
Quick	1137ms	1086ms
Radix	1435ms	1085ms

2.5 Conclusion

In my implementation, going off the results from turing, quick sort was faster given 4 different sets of random data. That said, results provided by my Macbook with the same data sets, showed a different story with both sorting algorithms being more or less the same with radix being slightly faster on average. I believe that the reason for this is for differences in how resources are allocated for java on my machine and turing. But based on the data provided from turing, I would say that quick sort would be the better option when it comes to sorting 10 millions numbers that range from 0 to $2^{32} - 1$.

3 Resources

The radix sort was a slight modification of the radix sort that I created for Homework 4