Radix Sort Project

Joshua Tensuan

March 1, 2020

1 Introduction

Quicksort is an efficient divide and conquer sorting algorithm which runs in $\mathcal{O}(log(N))$ sort. It works by selecting a pivot point and placing all the other elements of the array around this pivot (put elements less than the pivot on the left and vice versa). There are many ways to choose a pivot but for this project, we just used the last elemnt as the pivot.

Radix sort is an integer sorting algorithm. It avoids comparison by distributing elements into bins. Overall time complexity is $\mathcal{O}((N+b)\log_b(k))$ where N is the number of elements, b is the number of bins, and k is the max number in the array. For every sorted array, we optimized the number of bins, however in real application, this may not always be possible.

In this project, we first implement both quicksort and radix sort. Then, we take 4 databases and compare runtimes of each algorithm, in hopes of finding advantages and disadvantages of each sort. Finally, we conclude with when to use each sorting algorithm.

2 Procedure

We used four datasetas in the project to test each algorithm. One dataset we used implemented the 2019 AP Physics C: Electricity and Magnetism score distribution to generate numbers 1-5 [4]. This was designed to test the sorting algorithms with a dataset with very few unique numbers. Another dataset we used is the public worker income of Baltimore workers in the 2019 fiscal year [2]. This was designed to test efficiency in a dataset with completely unique large real world values. The next dataset we used is a hyperspectral and soil moisture dataset from an experiment that dealt with varying types of soil (sorted based off soil moisture) [3]. This was designed to test each algorithm's efficiency at sorting mostly unique (very few duplicates) large real data. Finally, the last dataset we used utilized the SAT score distribution (generated with a random normal distrubiton) [1]. This was designed to test efficiency in sorting data in a confined set with many duplicates. Every sort was ran 50 times to reduce chance variation.

3 Results

For every dataset, we manually optimized the bin size for radix sort. We did this by going through a handful of bin sizes and finding a minimum sorting time. Table 1 shows an example of the bin optimization process.

Table 1: Sample of bin optimization on the SAT Dataset. We chose a bin size of 150.

bin size	25	50	75	100	125	150	175	200	225
time (s)	19.838	13.854	12.970	12.254	12.236	12.266	12.024	12.298	12.62

We did this bin optimization process for every dataset. We used bin sizes of 155 for sorting the public worker incomes of Baltimore, 180 for the soil moisture dataset, and 9 for the AP Physics C scores. The bin size of 9 is unusual for the AP Physics C scores data as it is above the max number (5), but it is still faster than any other bin size so we still use it. Table 2 shows the results for sorting times for each dataset with quicksort and radix sort.

Table 2: Comparison of quicksort and radix sort runtimes for every dataset.

	AP Physics C Scores	*SAT Score*	*Soil Moisture*	*Public Worker Incomes*
Quicksort time (s)	169.96	29.05	56.5	6.01
Radix sort time (s)	12.024	4.52	86.9	7.02

Radix sort ran faster than quicksort for the AP Physics C scores and SAT scores datasets. On the other hand, quicksort ran faster than radix sort for the soil moisture and Baltimore public worker incomes dataset. For the times radix sort were quicker, radix sort was much faster than quick sort (7 times faster in sorting SAT scores and 13 times faster in sorting AP Physics C scores). In comparison, quicksort ran only marginally faster than radix sort (1.17 times faster in sorting public worker incomes and 1.5 times faster in sorting soil moisture times). Overall, radix sort excelled at sorting datasets with fewer unique numbers and smaller values while quicksort excelled at sorting datasets with large values.

4 Conclusion

In conclusion, radix sort is best at sorting smaller numbers and datasets with fewer unique numbers. Quicksort out sorted radix sort when there were larger numbers and more unique numbers. However, quicksort out sorting radix sort with larger numbers only occured with incredibly large values in a dataset, and most of the time with large numbers (on the order of 10^6), radix sort still did better. However, in terms of practical sorting, the difference in times between radix sort and quicksort even for these large datasets was largely negligible with only a few seconds separating the two. Further, radix sort requires the optimization of bin size which may not always be possible in the real world. On top of this, radix sort is limited to only integers (and possibly strings with a radix trie) while quicksort allows sorting between floating point numbers, which is very important in real world application. Finally, in terms of practical use, radix sort requires more memory as an auxiliary array/vector is needed for every iteration. Looking at this from a numerical standpoint, radix sort has a worst case space complexity of $\mathcal{O}(w + N)$ where w is the number of bits required to store each key, compared to quicksort's worst case space complexity of $\mathcal{O}(log(N))$. Compiling all this together, quicksort is the best standard sorting algorithm for general application, and radix sort should only be used in special situations when there is a small amount of unique integers in the dataset, and when system memory usage is not a big issue.

References

[1] College Board. *SAT Suite of Assessments Annual Report*. `https://reports.collegeboard.org/pdf/2019-total-group-sat-suite-assessments-annual-report.pdf`. 2019.

[2] Mayor's Office. "Baltimore City Employee Salaries FY2019". In: *Open Baltimore* (2019). URL: `https://data.baltimorecity.gov/City-Government/Baltimore-City-Employee-Salaries-FY2019/6xv6-e66h`.

[3] Felix M. Riese and Sina Keller. *Hyperspectral benchmark dataset on soil moisture*. `https://github.com/felixriese/hyperspectral-soilmoisture-dataset`. 2018. DOI: `10.5281/zenodo.1227837`.

[4] Shannon Willoughby. "Chief Reader Report on Student Responses: 2018 AP Physics C: Electricity and Magnetism Free-Response Questions". In: *College Board* (2018). URL: https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap18-chief-reader-report-physics-c-em.pdf.