07.01.2020

Algorithm Analysis Project

Comparison of Radix Sort and Rank Sort with Multithreading

ALPEREN ÇINAR

110510048

***Rank Sort***

Rank Sort is an algorithm that used to sort an array. It is something like mix between selection and insertion sort. It compares one element with every other elements to find its rank. After, found the rank it is inserted to a new array in the proper position. The time complexity for the rank sort is calculated as *O(n2)* for all cases(worst case, best case, average case). In Figure 1, an example for rank sort is shown.

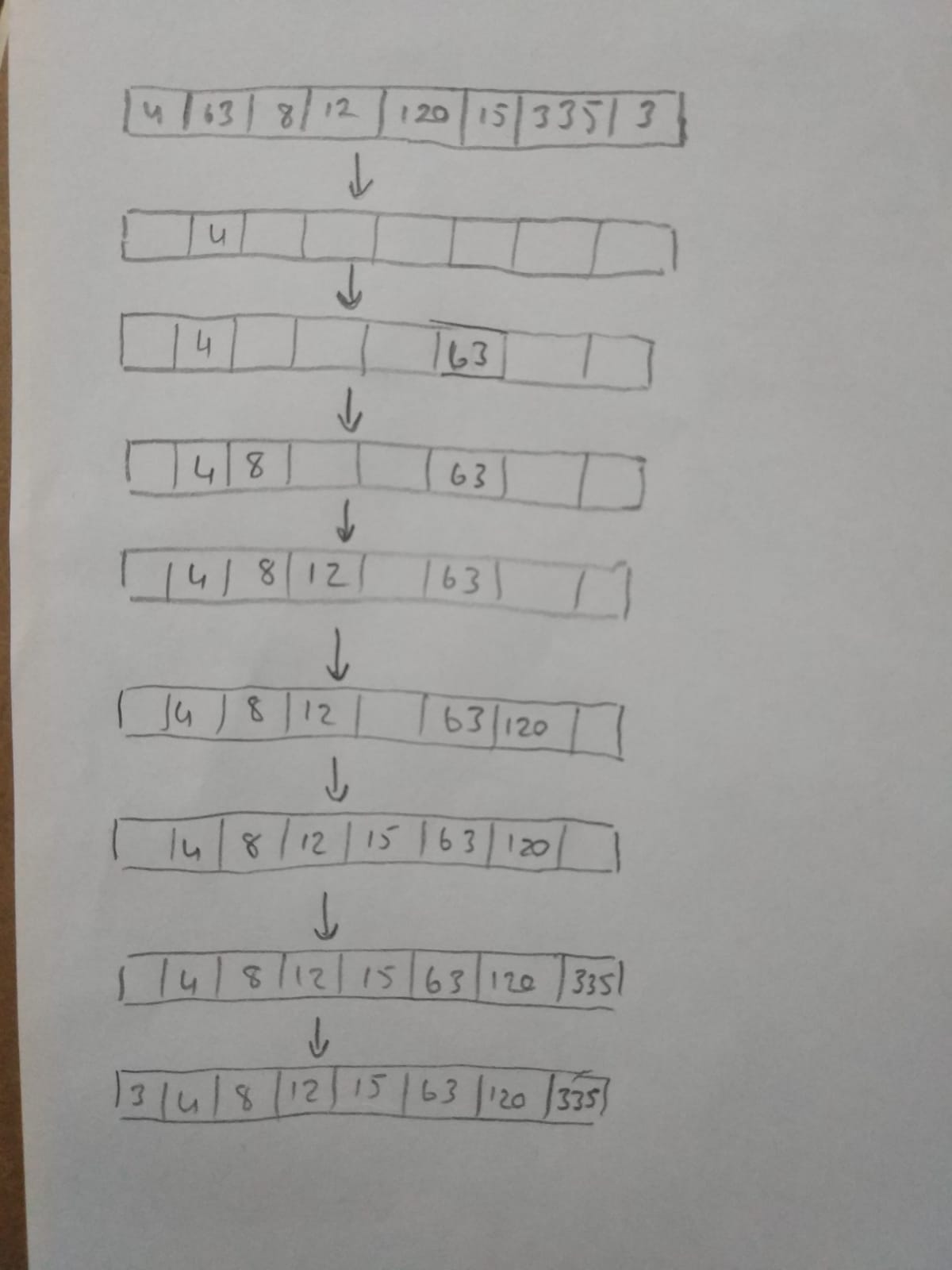


Figure 1: The Rank Sort Example

***Radix Sort***

It is another sorting algorithm, which sorts elements in linear time. Radix is the base of numeral system. For instance, the radix for decimal system is 10. Radix sort puts elements to buckets, which are based on parts of the key for the element. It generally starts with least significant key. After that, the element is put to a new array. This repeats until it is done for all possible keys. An extra sorting algorithm is used to sort the keys. Generally, bucket and counting sorts are used for this step. Radix sorts has two different version. First one is least significant key that uses the least significant key as first point to sort. The other one is most significant key which uses most significant key as starting point to sort and move downwards.

The complexity of radix sort depends the maximum number of digits (max) and the base of numerical system (k). The complexity of radix sort is calculated as *O(max\*(n+k))* for all cases. In Figure 2, an example of radix sort is shown.

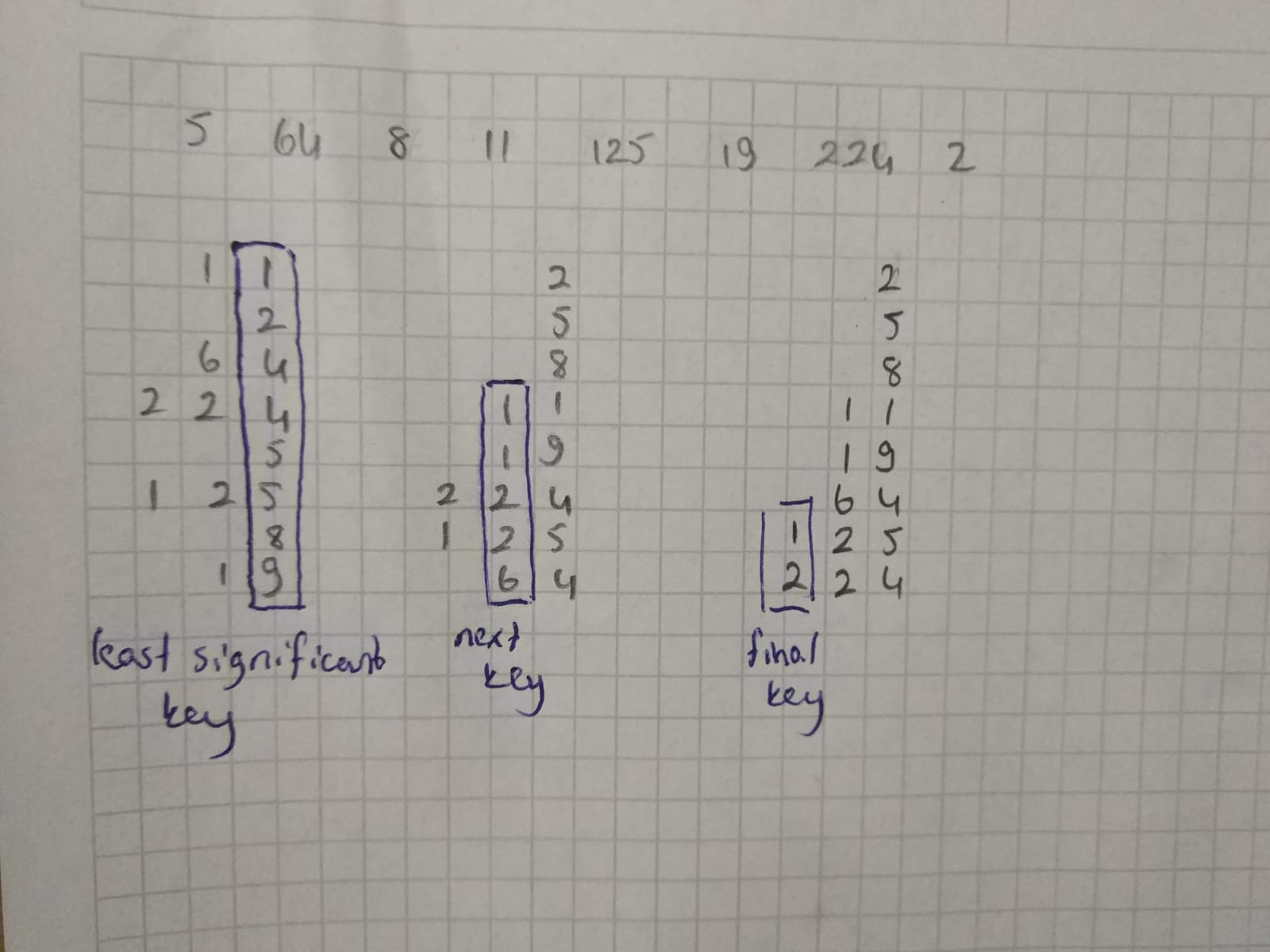


Figure 2: Radix Sort Example

***Example Input Output:***

In the codes both of sorting algorithms are executed. They are tested with an dataset which has 16 elements. The results are shown in the following lines.

For Radix Sort:

*Array before sort:*

*917 234 117 843 559 254 26 607 820 909 313 511 475 952 659 105*

*Array after sort:*

*26 105 117 234 254 313 475 511 559 607 659 820 843 909 917 952*

For Rank Sort:

*Array before sort:*

*917 234 117 843 559 254 26 607 820 909 313 511 475 952 659 105*

*Array after sort:*

*26 105 117 234 254 313 475 511 559 607 659 820 843 909 917 952*

***Multithreaded:***

Both of the algorithms are executed with single thread, 2 threads and 4 threads for different array sizes (16, 256, 4096, 65536, 1048576, 16777216) and execution times are calculated. The datasets with different sizes are in also project file with codes. The results are shown in the following tables.

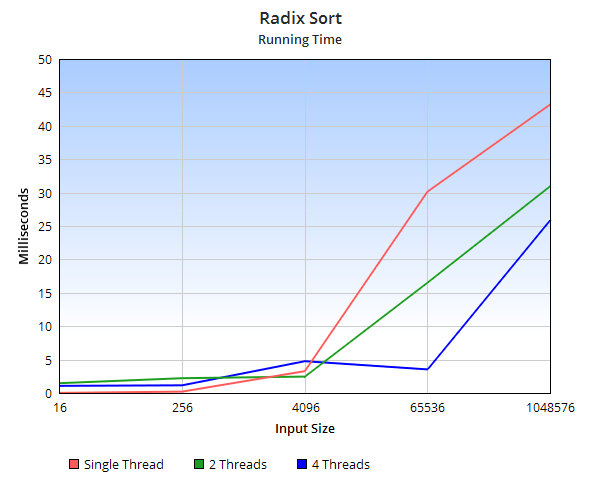
|  |  |  |  |
| --- | --- | --- | --- |
| **Array Size** | **Single Thread** | **2 Threads** | **4 Threads** |
| 16 | 0,0168 | 1,4694 | 1,0687 |
| 256 | 0,2077 | 2,2157 | 1,1519 |
| 4096 | 3,2732 | 2,458 | 4,7591 |
| 65536 | 30,1689 | 16,5407 | 3,5296 |
| 1048576 | 43,2246 | 30,986 | 25,8926 |
| 16777216 | 669,7813 | 598,582 | 433,6944 |

Table1: Running Times in Radix Sort (millisecond)

|  |  |  |  |
| --- | --- | --- | --- |
| **Array Size** | **Single Thread** | **2 Threads** | **4 Threads** |
| 16 | 0,0119 | 1,9687 | 1,0419 |
| 256 | 2,2512 | 2,5045 | 1,4831 |
| 4096 | 62,8282 | 73,5063 | 52,7687 |
| 65536 | 13307,72 | 7156,982 | 4620,332 |

Table2: Running times for Rank Sort (millisecond)

According to these results the following graphs are drawn.



Graph 1: Running Time Radix Sort

***Observations:***

Graph 2: Running times for rank sort

First, in the rank sort graphs since the range of running times are too big (0.01199-13307.72) the running times for small array sizes are not clear in the Graph 2 to compare but the values are also shown in the Table 2.

As shown in the graphs, when the array sizes are small like 16 or 256 single thread algorithm runs faster than multithreaded algorithms for radix sort. However, when the size of array bigger than 4096 multithreaded programs execute faster than single threaded.

On the other hand, 4 threads algorithms generally run faster than the others for rank sort but if the array size is smaller than 4096 single thread run faster than 2 threads. If the array size is bigger than 4096, the fastest is with 4 threads and slowest is with single thread. Also, in radix sort after 65536, the computer could not finish to execute program.

As a result, as shown in the tables, radix sort is much faster than rank sort and multi-threaded algorithms are faster than single thread algorithm for large array sizes.

***Application Areas:***

According to Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, David E. Culler, Joseph M. Hellerstein, and David A. Patterson, while reading disk, a bucket sort with a partial radix sort can be used to reduce the in-memory sort time for one million records. Moreover, Terdiman P. (2000) reported that radix sort can be used in computer graphics. In addition, Joshi A (2019) claimed that radix sort generally used in big data analysis. For example, English words for dictionary. In addition, eBay allows sorting listings with current bid amount leveraging radix sort. To sum up, radix sort is an algorithm that can be used in different areas such as to utilize hardware and in some websites.

Unlike radix sort, rank sort is not fast enough for big data. That is why it is not efficient to use it with big data such as databases of applications and websites. However, we can use it to find closest pair, frequency distribution, selection and element uniqueness as the other sorting algorithms. Moreover, it can be used to display Google PageRank results and identify statistical outliers since this kind of applications do not use big data.

***Improvements***

Several methods can be implied to improve radix sort. According to Liu X. and Deng Y. (2014), some hardware solutions can be applied to improve bucket sort. In addition, we can have preliminary check for initial array to understand number of bytes to sort. When the first byte is sorted, the shift operation can be ignored to earn time. In addition, if one of the buckets has all the elements, it is not necessary to sort this byte so this can be skipped. Moreover, we can use basic math to improve algorithm, if array contains some negative values, these values can be at the end. For 32 bit, integer values there are 4 iterations at max. Therefore, the algorithm can switch the purpose of two arrays, which are the helper array and original array to reduce copying overhead at the end of each iteration. If this kind of improvements can be applied for radix sort, it will be more efficient to use radix sort in application areas such as big data and hardware because it will be faster.

In the literature, there are no study about to improve rank sort algorithm. However, the multi-processor and multithreading can be used to improve rank sort as proved in this project. In addition, like every other application the hardware is an important point to improve an algorithm.

***References***

1. Andrea C. Arpaci-Dusseau , Remzi H. Arpaci-Dusseau , David E. Culler , Joseph M. Hellerstein , David A. Patterson (2020). *High-Performance Sorting on Networks of Workstations*. *Citeseerx.ist.psu.edu*. Retrieved 6 January 2020, from <http://citeseerx.ist.psu.edu/viewdoc/versions?doi=10.1.1.141.2971>
2. Joshi A. (2019) “The Radix Sort Algorithm” taken from:

<https://www.quora.com/What-are-real-life-examples-of-radix-sort>

1. Terdiman P.(2000) “Radix Sort Tevisited” retrieved from:

<http://codercorner.com/RadixSortRevisited.htm>

1. X. Liu and Y. Deng, "Fast Radix: A Scalable Hardware Accelerator for Parallel Radix Sort," 2014 12th International Conference on Frontiers of Information Technology, Islamabad, 2014, pp. 214-219.  
   doi: 10.1109/FIT.2014.48  
   URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7118402&isnumber=7118353>