http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (3): 424-427, 2007
ISSN 1812-5638
© 2007 Asian Network for Scientific Information

Comparative Analysis of Some Pivot Selection Schemes for Quicksort Algorithm

[1]Aminu Mohammed and [2]Mohamed Othman
[1]Department of Computing Science, Faculty of Information and Mathematical Sciences,
University of Glasgow, G12 8RZ Glasgow, UK
[2]Department of Communication Technology and Networks,
Faculty of Computer Science and Information Technology,
University Putra Malaysia, 43400 UPM Serdang, Selangor, Darul Ehsan, Malaysia

Abstract: Data sorting is an intriguing problem that has attracted some of the most intense research efforts in the field of computer science for both its theoretical importance and its use in many applications. Quicksort is widely considered to be one of the most efficient sorting techniques, which depends on an appropriate pivot selection technique for its performance. In this study, a sequential quicksort was implemented using six different pivot selection schemes for minimizing the execution time of quicksort algorithm. The schemes were tested together using integer array data type. From the results obtained, median-of-five without random index selection scheme minimizes the execution time of quicksort algorithm by about 23-35% as compared to the other techniques. The results also indicated that there is an overhead associated with random index selection and this can affect the performance of a particular method. Thus, this factor needs to be considered in selecting an optimal pivot selection scheme.

Key words: Data sorting, partitioning, pivot selection scheme, quicksort algorithm

INTRODUCTION

Sorting is the use of computer to put files in order. It is one of the most intensively studied problems in computing science. It continues to consume sizable fraction of time on mainframe and personal computers, accounting for roughly one quarter to half of their cycles (Brest *et al*., 2000). Sorting has continued to be an interesting theoretical and difficult practical problem. Many sorting algorithms have been proposed (Sedgewick, 1977; Hoare, 1961).

Although there is no internal sorting that is best for every situation, the quicksort algorithm introduced by Hoare (1961, 1962) is widely accepted as the most efficient internal sorting technique. Quicksort sorts a list of keys A[1], A[2],.., A[m] recursively by choosing a key p in the list as a pivot key around which to rearrange the other keys in the list. Ideally, the pivot key is near the median key value in the list, so that it is preceded by about half of the keys and followed by the other half. The keys of the list are rearranged such that for some n, A[1], A[2],.., A[n] contain all the keys with values less than p and A[n+1], A[n+2],.., A[m] contain all the keys with values greater than or equal to p. The elements A[1], A[2],.., A[n] are called the left sub list and the elements A[n+1], A[n+2],.., A[m] are the right sub list. Thus, the original list is partitioned into two sub lists where all the keys of the left sub list precede all the keys of the right sub list. After partitioning, the original problem of sorting the entire list is now reduced to the problem of sorting the left and right sub lists independently. Quicksort is then applied recursively to each of these sub lists until the sub list consist of just a single item. Not only is this algorithm simpler than many other sorting algorithms, but empirical (Sedgewick, 1977; van Emden, 1970) and analytical (Knuth, 2005) studies shows that quicksort can be expected to be up to twice as fast as its nearest competitors, with expected time complexity of O (*nlogn*) and a worst case of O (n^2).

There is a continuous demand for greater computational speed from a computer system than currently possible (Moh *et al*., 1999). Areas requiring great computational speed include numerical modelling and simulation of scientific and engineering problems. Such problems often need huge repetitive calculation on large amount of data to give valid result. Thus, it is appropriate to study the effect of pivot selection schemes on the performance of quicksort algorithm.

Overview of pivot selection techniques: There are several ways to make the worst case for a quicksort algorithm very unlikely in practical situations. Instead of using the

Corresponding Author: Aminu Mohammed, Department of Computing Science,
Faculty of Information and Mathematical Sciences, University of Glasgow, G12 8RZ Glasgow, UK

first element in the list as the partitioning element, one may use some other fixed element, like the middle element. This helps some, but simple anomalies can still occur. Using a random partitioning element will virtually prevent the anomalous cases from happening in practical sorting situations, but random number generation can be relatively expensive and does not reduce the average running time of the rest of the algorithm (Singleton, 1969).

Median-of-three method: The best choice of pivot would be the median of the array; unfortunately this is hard to calculate and would slow down quicksort considerably. The median of three modifications (Hoare, 1962) will actually improve the average performance of the algorithm while at the same time making the worst case unlikely to occur in practice (Weiss, 2000).

In this method, samples of size three are used at each particular stage. Primarily, sampling provides insurance that the partitioning elements don't consistently fall near the ends of the sub lists. To make the worst case unlikely the method uses the first, middle and the last elements as samples and the median of those three as the partitioning element. Using this method clearly eliminates the bad case for sorted input and actually reduces the running time of quicksort by about 5% (Hoare, 1962; Singleton, 1969).

Median-of-five with random index selection method: As larger sample size gives better estimates of the median, many researchers had made attempts to improve on the running time of quicksort by proposing different pivot selection techniques. Thus, median-of-five method was used in Brest *et al*. (2000) and Cerin (2002). The method uses a sample size of five elements, i.e., the first, middle, last and two other elements randomly picked through a random number generation function between the first and last elements. This technique gives a better load balancing and reduces the execution time of quicksort by more than 5%, but there is an overhead associated with random number generation. A snapshot of this method is presented in Fig. 1.

Median-of-five without random index selection method: In (Mohammed and Othman, 2004), a new pivot selection scheme for quicksort algorithm was proposed, which does not involve any use of random index selection. This method also uses a sample size of five elements as in median-of-five with random index selection scheme. The five elements are selected as follows: first, middle, last and other two are at position $\lceil (\text{low} + \text{high})/4 \rceil$ and $\lceil 3*(\text{low} + \text{high})/4 \rceil$, where low and high are indexes of the first and last elements in the original array. Thus, it

```
Median51 Scheme (int A[ ], int low, int high)
{
int V[5];
V[0] = low;
V[1] = high;
V[2] = (low + high) / 2;
V[3] = (low + high) / 4;
V[4] = 3((low + high) / 4);

for ( int i = 0; i < 5: i ++ )
for ( int j = 0; j < 4: j ++ )
if (A[V[j]] > A[V[j + 1]])
{
int temp = V[j];
V[j] = V[j + 1];
V[j + 1] = temp;
}
return V[2];
}
```

Fig. 1: A snapshot of median50 scheme

```
Median51 scheme (int A[ ], int low, int high)
{
int V[5];
V[0] = low;
V[1] = high;
V[2] = (low + high) / 2;
V[3] = (low + high) / 4;
V[4] = 3((low + high) / 4);

for ( int i = 0; i < 5: i ++ )
for ( int j = 0; j < 4: j ++ )
if (A[V[j]] > A[V[j + 1]])
{
int temp = V[j];
V[j] = V[j + 1];
V[j + 1] = temp;
}
return V[2];
}
```

Fig. 2: A snapshot of median51 scheme

forms a new array of five elements, which will be used by the algorithm. It then returns the middle element after sorting the new array. The returned middle element is the median of those five elements and is used as the partitioning element. Thus, it eliminates the use of random index selection.

The snapshot of the new pivot selection scheme is presented in Fig. 2. All the five elements are selected explicitly as shown in the snapshot. The first five elements are picked from the unsorted array and then sorted using sample sort. At the end the median of the five elements i.e., V[2]; is returned and will be used as the pivot element.

Median-of-seven without random index selection method: This method selects seven elements, all without random selection. Thus, the number of

elements to be considered for selecting the pivot is increased. The elements are selected as follows: low, high, middle, $\lceil (low + high)/3 \rceil$, $\lceil 2*(low + high)/3 \rceil$, $\lceil 5*(low + high)/6 \rceil$ and $\lceil (low + high)/6 \rceil$. The elements are sorted and the middle element is returned.

Median-of-seven with random index selection method: This method just like the previous method selects seven elements from the original array indexes. The first five elements are selected as follows: low, high, $\lceil (low + high)/2 \rceil$, $\lceil (low + high)/4 \rceil$ and $\lceil 3*(low + high)/4 \rceil$. While the other two elements are selected randomly using a random number generator functions as shown in Fig. 1.

Median-of-nine without random index selection method: In this method, the number of elements to be selected has increased by two. It does not employ random selection. Thus, all the nine element are explicitly selected as follows: low, high, $\lceil (low + high)/2 \rceil$, $\lceil (low + high)/8 \rceil$, $\lceil (low + high)/4 \rceil$, $\lceil 3*(low + high)/8 \rceil$, $\lceil 5*(low + high)/8 \rceil$ and $\lceil 7*(low + high)/8 \rceil$. Thus, more elements are involved in the selection process.

Median-of-nine with random index selection method: This scheme employs random selection, in which two elements are selected randomly using the random generating function used in Median-of-seven with random index selection. The other seven elements are explicitly selected as follows: low, high, $\lceil (low + high)/2 \rceil$, $\lceil (low + high)/6 \rceil$, $\lceil (low + high)/3 \rceil$, $\lceil 2*(low + high)/3 \rceil$ and $\lceil (5*low + high)/6 \rceil$.

Implementation: The algorithm was implemented in C++ programming language on a PC-cluster machine. Our implementation was sequential and thus the parallel capability of the PC-cluster was not utilized. The purpose of using a PC-cluster as a normal PC was to utilize its larger array size capability. The maximum array size that can be declared on the PC-cluster was two million elements. In the experiment, integer array data type is used. The data were randomly generated and 40 independent measurements were performed in each case. Thus, we operated on average values.

The implementation was based on three scenarios. In the first scenario, all the six schemes were run with the same quicksort algorithm using an array size of 100 to 330 thousand elements. Based on the first scenario, the best three schemes in terms of execution time were taken and implemented separately using an array size of 250 to 600 thousand elements. The idea was to make further analysis by increasing the array size. From the second scenario, the best two of the three schemes were implemented separately using an array size of 700 thousand to 1 million elements, in order to make further informed performance analysis. In all the three implementation scenarios, the array was generated randomly by a function in the program.

RESULTS AND DISCUSSION

The experimental results of the implemented sequential quicksort algorithm using the six pivot selection schemes are presented. Three different scenarios are analyzed. The first scenario compares the six techniques together, while the second compared the best three techniques of the first scenario and in the third scenario the best two techniques of the second scenario were compared. Table 1 shows the execution time of the six partitioning methods for the first scenario. The Median50, Median70 and Median90 represent median-of-five, median-of-seven and median-of-nine partitioning methods without random index selection, respectively. While Median51, Median71 and Median91 represent median-of-five, median-of-seven and median-of-nine partitioning methods with random index selection, respectively. The second scenario results are presented in Table 2 which shows the execution time of the three methods. The third scenario results are presented in Table 3 which shows the execution time of the two methods.

The results of Table 1 shows that, three of the methods i.e., median-of-seven with and without random selection and median-of-nine without random selection, have almost the same execution time for all the array sizes. Median-of-nine with random selection was the worse in term of speed. While the least in terms of execution time

Table 1: Average execution time (sec) of the six schemes

Array size	Median 50	Median 51	Median 70	Median 71	Median 90	Median 91
100K	0.16	0.21	0.31	0.32	0.32	0.51
150K	0.25	0.34	0.47	0.47	0.49	0.77
200K	0.34	0.47	0.65	0.66	0.66	1.06
250K	0.44	0.60	0.84	0.84	0.86	1.35
300K	0.53	0.72	1.04	1.05	1.05	1.66
350K	0.58	0.80	1.16	1.18	1.18	1.83

Table 2: Average execution time (sec) of the best three schemes of the first scenario

Array size	Median 50	Median 51	Median 70
250K	0.44	0.66	0.84
300K	0.53	0.72	1.03
350K	0.62	0.86	1.22
400K	0.72	0.99	1.43
450K	0.82	1.13	1.63
500K	0.91	1.26	1.84
550K	1.01	1.44	2.06
600K	1.12	1.55	2.28

Table 3: Average execution time (sec) of the two best schemes of the second scenario

Array size	Median 50	Median 51
700K	1.60	2.15
750K	1.75	2.34
800K	1.91	2.42
850K	2.05	2.62
900K	2.18	2.71
950K	2.35	2.82
1 M	2.54	2.95

(fast in terms of speed) is median-of-five without random selection, then followed by median-of-five with random selection. Thus, the scheme in Mohammed and Othman (2004) was faster and therefore minimizes the execution time of quicksort algorithm sequentially by at least 31%.

Results of Table 2 shows that median-of-seven without random index selection has the worst performance, while the scheme in (Mohammed and Othman, 2004) has the best execution time and was about 35% faster than the two schemes.

Table 3 clearly shows the performance of the best two methods. Like the first two scenarios Tables, median-of-five without random index selection presented in Mohammed and Othman (2004) performs better than the other scheme presented in Brest *et al*. (2001)

CONCLUSIONS

In this study, pivot selection methods in quicksort algorithm and experimental study of their performance were presented. Table 1, 2 and 3 show that median-of-five without random index selection was faster than the other five schemes presented. Thus, the execution time of quicksort algorithm was minimized by about 23-30%. It is evident that the larger the array, the more critical a decision will be in selecting the appropriate or optimal median-of-N pivot selection method. Likewise, our result indicated that, there is an overhead associated with random index selection and this can affect the performance of a particular pivot selection scheme.

The work is constrained by our inability to generate larger array size that can be in tens or hundreds of million in sizes. Databases and lists in real world problems do not exist only in integer data type form, thus the need for comparing the schemes using different data types.

ACKNOWLEDGMENTS

We are grateful to the authorities of University Putra Malaysia for providing the research facilities and Usmanu Danfodiyo University, Sokoto, Nigeria for the study fellowship.

REFERENCES

Brest, J., A. Vreze and V. Zumer, 2000. A Sorting Algorithm on Pc Cluster. In: Proceedings of the 2000 ACM Symposium on Applied Computing, Como, Italy.

Cerin, C., 2002. An out-of-core sorting algorithm clusters with processors at different speed. In: Proceedings of the IEEE Parallel and Distributed Processing Symposium, 15-19 April 2002, Ft. Lauderdale, FL, USA.

Hoare, C.A.R., 1961. Partition: Algorithm 63: Quicksort: Algorithm 64: and Find: Algorithm 65, Communication of ACM, 4: 321-322.

Hoare, C.A.R., 1962. Quicksort. Computer J., 5: 10-15.

Knuth, D.E., 2005. The Art of Computer Programming, Vol. 3: Sorting and Searching Addison-Wesley, Mass., USA.

Mohammed, A. and M. Othman, 2004. A new pivot selection scheme for quicksort algorithm, suranaree. J. Sci. Technol., 11: 211-215.

Moh, S., S. Kim, M. Lee, C. Yu and D. Han, 1999. A new parallel quicksort with efficient processor allocation and minimal communication. In: Proceedings of SIG on Parallel Processing System conference, 10-11 September 1999, Seoul, Korea.

Sedgewick, R., 1977. Quicksort with Equal Keys. Siam J. Comput., 6: 240-287.

Singleton, R.C., 1969. Algorithm 347: An efficient algorithm for sorting with minimal storage. Commun. ACM, 12: 185-187.

Van Emden, M.H., 1970. Increasing the efficiency of quicksort. Commun., ACM, 13: 563-567.

Weiss, M.A., 2000. Data Structure and Algorithm Analysis in C++, Addison-Wesley Publishing, Longman, Inc, Reading.