Malaysian Journal of Computer Science, Vol. 13 No. 2, December 2000, pp. 16-20

ENHANCED DIVIDE-AND-CONQUER ALGORITHM WITH 2-BLOCK POLICY

Mustafa Mat Deris,
Mohd Pouzi Hamzah
Faculty of Science and Technology
Dept. of Computer Science and Information Technology
College University Terengganu
21030 Kuala Terengganu
Terengganu, Malaysia
email: mustafa@uct.edu.my
mph@uct.edu.my

Ali Mamat
Faculty of Computer Science and Information Technology
University Putra Malaysia
43400 UPM Serdang
Malaysia
email: ali@fsas.upm.edu.my

ABSTRACT

The number of comparisons involved in searching minimum and maximum elements from a set of data will determine the performance of an algorithm. A Divide-and-Conquer algorithm is the most efficient algorithm for searching minimum and maximum elements of a set of data of any size. However, the performance of this algorithm can still be improved by reducing the number of comparisons of certain sets of data. In this paper a 2-block (2B) policy under the divide-and-conquer technique is proposed in order to deal with this problem. On the basis of this policy, the divide-and-conquer algorithm is enhanced. It is shown that the performance of the proposed algorithm performs equally at par when compared with the established algorithm of data size of power of two and better when compared with data size of not a power of two.

Keywords: ***Algorithm, Divide-and-Conquer, Performance, Recursive***

1.0 INTRODUCTION

One of the areas that is substantially important with storage I/O problems is an algorithm, which involves the technique of searching and computation [1, 2, 3, 6]. An algorithm can provide a statisfactory solution to a problem if it is efficient and produces a correct answer. One measure of an algorithm's efficiency is the time used by a computer in solving a problem when input values are of a specified size [5]. The time taken by an algorithm can be expressed in terms of the number of operations used by the algorithm for a certain number of input values.

For the case of searching minimum and maximum elements from a set of data, the time complexity of the algorithm can be expressed in terms of the number of comparisons. Therefore, the smaller the number of comparisons needed, the better the algorithm is. Hence, the performance of the algorithm increases with the decrease in the number of comparisons.

Until now, Pohl's algorithms are the most popular algorithms for searching minimum and maximum elements from a set of data. Pohl has established two different algorithms [4] to cater two different forms of data size. The first algorithm is for the data size in a power of two, i.e., it has a form of $n = 2^k$, where k is a positive integer. The second one is for the case of $n \neq 2^k$. According to Pohl, the algorithm for $n = 2^k$ is still the best algorithm and the algorithm for $n \neq 2^k$ is still a very efficient algorithm [4]. The number of comparisons in the algorithm for $n = 2^k$ is the same as the one expressed by the formula given by Rosen [5]. However, the algorithm for the case of $n \neq 2^k$ requires a lot of comparisons due to the large number of recursive calls for splitting the data set. Therefore, that algorithm does not perform well because a large number of comparisons are needed. In this paper, an enhanced algorithm by means of a 2-block (2B) policy is proposed in dealing with a large number of comparisons used in divide-and-conquer technique. The 2B policy ensures that the algorithm is applicable in both cases, i.e., for the case of $n \neq 2^k$ as well as $n = 2^k$. Prior to this, the formula for calculating the number of comparisons based on the enhanced algorithm will be presented. We then use this formula to compute the performance or efficiency of the algorithm for different number of elements and hence make a comparison to the existing formula based on its performance.

The rest of the paper is organized as follows: in section 2, a general formula for expressing the number of comparisons is presented. The new formula to compute the number of comparisons by means of the 2B policy when a data set is of any size is derived in section 3. In section 4, the enhanced divide-and-conquer algorithm is then presented. The performance comparison between the enhanced algorithm and Pohl's algorithm is also described. Finally, some concluding remarks are presented in section 5.

2.0 THE NUMBER OF COMPARISONS

Many recursive algorithms take a problem with a given input and divide it into one or more smaller problems. This reduction is successively applied until the solutions of the smaller problems can be found quickly. These procedures are called divide-and-conquer algorithms [1, 5]. Consider the following algorithm for locating the minimum and

maximum elements of a sequence $a_1, a_2, \ldots, a_n$. If n=1, then a_1 is the minimum and the maximum. If n > 1, split the sequence into two sets, where each set contains n/2 elements when n is even. If n is odd then one of the sets has one element more than the other. For example, if n = 7, one set has 3 elements and the other has 4 elements. The problem is reduced to finding the minimum and maximum of each of the two smaller sets. The solution to the original problem results from the comparison of the separate minima and maxima of the two smaller sets to obtain the overall minimum and maximum.

Let $f(n)$ be the total number of comparisons needed to find the minimum and maximum elements of the set with n elements. A problem of size n, where n is a power of two can be reduced into two problems of size n/2, using two comparisons, one to compare the minima of the two sets and the other to compare the maxima of the two sets. This gives the divide-and-conquer recurrence relation:

$$f(n) = f(\lfloor n/2 \rfloor) + f(\lfloor n/2 \rfloor) + 2$$
$$= 2f(\lfloor n/2 \rfloor) + 2$$

For any positive integer n, the above recurrence relation can be express as:

$$f(n) = f(\lfloor n/2 \rfloor) + f(n - \lfloor n/2 \rfloor) + 2 \quad \ldots \quad (1)$$

where $(\lfloor n/2 \rfloor)$ means the largest integer that is less than or equal to a. Equation (1) can be deduced recursively and it is (Appendix 1.1 for the derivation):

$$= \sum_{i=0}^{k} \left(\binom{k}{i} f\left(\sum_{j=0}^{i} (-1)^{i-j} \binom{i}{j} \lfloor n/2^{k-j} \rfloor \right) \right) + 2\sum_{j=0}^{k-1} 2^j \quad \ldots \quad (2)$$

For the case where n is a power of 2, then $n - \lfloor n/2 \rfloor = \lfloor n/2 \rfloor$. Therefore, from equation (2), $f(n)$ (Appendix 1.2 for the derivation);

Equation (3) is the same as discussed by Rosen [5].

$$= 2^k f(1) + 2\sum_{j=0}^{k-1} 2^j \quad \ldots \quad (3)$$

3.0 THE 2B POLICY AND SOLUTION METHOD

In what follows, the terms an array of data and a set of data will be used interchangeably. An array of integers of size n will be split recursively using the divide-and-conquer technique until an array with two elements when $n = 2^k$, or an array containing 1 element when $n \neq 2^k$ is obtained. The initial conditions for $f(2) = 1$ and $f(1) = 0$ are used, when $n = 2^k$ and $n \neq 2^k$ respectively. For the case of $n = 2^k$, Pohl has given the algorithm such that the number of comparisons is the same as the equation (3) where the minimum number of elements in an array of integers is 2. In other words, no recursive calls will be required for n = 2. Therefore equation (3) should be rewritten as follows:

$$f(n) = 2^{k-1} f(2) + 2\sum_{j=0}^{k-2} 2^j, \quad for \quad k \geq 2. \quad \ldots \quad (4)$$

However, for the case of $n \neq 2^k$, a number of recursive calls will be required until an array contains 1 element. Therefore, Pohl's algorithm [4] will cause a lot of comparisons due to a large number of recursions. This is due to the fact that, if n is the number of data in a data set, then there will be n base cases, each with no comparison, and n-1 recursive calls, each of which is followed by two comparisons, hence a total of 2(n-1) comparisons. For instance, let n=5, then the number of recursive calls is 4 and the number of comparisons is 8. This can be shown graphically as in Fig. 1.

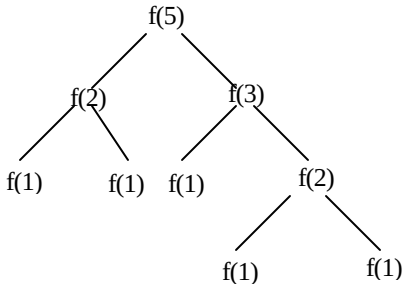

Fig. 1: Graphical representation for n = 5

We will hence introduce a policy, which reduces the number of recursive calls. In this policy, the smallest size of an array is 1. However, an array with size 2 will not be further split or blocked. Therefore, the minimum size of sub arrays is either 1 or 2. When n = 2 or n = 1, no recursive call is required. This policy is called 2block (2B) policy. Due to the initial conditions of n, the range of n should be $2^{k+1} < n < 2^{k+2}$, where k is a positive integer. Thus, the present formula from equation (2) can be deduced as:

$$f(n) = \sum_{i=0}^{k} \left(\binom{k}{i} f\left(\sum_{j=0}^{i} (-1)^{i-j} \binom{i}{j} \lfloor n/2^{k-j} \rfloor \right) \right) + 2\sum_{j=0}^{k-1} 2^j$$

where $k = \lfloor \log_2 n - 1 \rfloor$.

By applying equation (5), for the case of n =5, the number of recursive calls is 2, and the number of comparisons is 6.

$$\begin{cases} 2^k f(2) + (n - 2^{k+1}) f(1) + 2(n-1) - 2^{k+1} & for \quad n \leq 3 * 2^k \ldots \quad (5a) \\ (n - 2^{k+1}) f(2) + (2^{k+2} - n) f(1) + 2^{k+2} - 2 & for \quad n > 3 * 2^k \ldots \quad (5b) \end{cases}$$

Property 1: Equation (5) is also applicable for the case of $n = 2^k$.

Proof: To prove the property (2) it must be shown that equation (5) holds for the case of $n = 2^k$.

First, consider equation (5a). Since $2^k \leq 3^* 2^{k-1}$, then by substituting k with k-1, we have,

$$\begin{aligned} f(n) &= 2^{k-1} f(2) + (2^k - 2^k) f(1) + 2(2^k - 1) - 2^k \\ &= 2^{k-1} f(2) + 0 f(1) + (2^k - 2) \\ &= 2^{k-1} f(2) + 2(2^{k-1} - 1) \\ &= 2^{k-1} f(2) + 2 \sum_{j=0}^{k-2} 2^j, \qquad \text{for } k \geq 2 \end{aligned}$$

Similarly, since $2^k > 3^* 2^{k-2}$, we substitute k with k-2 in equation (5b). Hence

$$\begin{aligned} f(n) &= (2^k - 2^{k-1}) f(2) + (2^k - 2^k) f(1) + (2^k - 2) \\ &= 2^{k-1} f(2) + 0 f(1) + 2(2^{k-1} - 1) \\ &= 2^{k-1} f(2) + 2 \sum_{j=0}^{k-2} 2^j, \qquad \text{for } k \geq 2 \end{aligned}$$

This property is the same as in equation (4).

Property 2: The number of comparisons for equation (5) can be derived explicitly as below:

$$f(n) = \begin{cases} 2n - 2^k - 2 & \text{for } n \leq 3^* 2^k \quad (6a) \\ n + 2^{k+1} - 2 & \text{for } n > 3^* 2^k \quad (6b) \end{cases}$$

Proof: The original divide-and-conquer requires the following numbers of comparisons and recursions: Let n be a number of data in a data set. If n is a power of two, there will be n/2 base cases, each with one comparison, and n/2-1 recursive calls, each with two comparisons, for a total of n+n/2-2 comparisons [4]. If n is not a power of two, the total number of comparisons are 2(n-1) as mentioned in the earlier section.

From property (1), the coefficient of $f(2)$ and $f(1)$ are the number of base cases for $f(2)$ and $f(1)$ respectively.

Therefore, from equation (5a), for the case of $f(2)$, there will be 2^k base cases, each with one comparison, and $2^k - 1$ recursive calls, each with two comparisons, for a total of $2^k + 2(2^k - 1) = 3^* 2^k - 2$ comparisons. For the case of $f(1)$, there will be $n - 2^{k+1}$ base cases, each with no comparison, and $n - 2^{k+1}$ recursive calls, each of which is followed by two comparisons, hence a total $2(n - 2^{k+1})$ comparisons.

Thus,

$$\begin{aligned} f(n) &= (3^* 2^k - 2) + 2(n - 2^{k+1}) \\ &= 2n - 2^k - 2. \end{aligned}$$

Similarly, from equation (5b), for the case of $f(2)$, there will be $n - 2^{k+1}$ base cases, each with one comparison, and $n - 2^{k+1} - 1$ recursive calls, each with two comparisons, for a total of $n - 2^{k+1} + 2(n - 2^{k+1} - 1) = 3(n - 2^{k+1}) - 2$ comparisons. For the case of $f(1)$, there will be $(2^{k+2} - n)$ base cases, each with no comparison, and $2^{k+2} - n$ recursive calls, each of which is followed by two comparisons, hence a total $2(2^{k+2} - n)$ comparisons.

Thus,

$$\begin{aligned} f(n) &= 3(n - 2^{k+1}) - 2 + 2(2^{k+2} - n) \\ &= n + 2^{k+1} - 2 \end{aligned}$$

Therefore, equation (6) holds.

4.0 THE PROPOSED ALGORITHM AND PERFORMANCE COMPARISON

As an implementation of the 2B policy, the algorithm, which guarantees the least number of comparisons expressed by equation (5), is presented below. This algorithm contains two base cases, namely when number of elements (n) in an array is 1 and 2 respectively. The algorithm is described as follows:

```
/* the size of the array a is n  */
 divide-and-conquer:
  if( n =1)
       min = max = a[0]
  else
  if( n =2)
    find min_max (a[0] , a[1])

  else
   divide-and-conquer (a,n/2, min,max)
    divide-and-conquer (a+n/2, n-n/2,min,max)

    find_min (min1,min2)
    find_max(max1,max2)
end.
```

Fig. 3: Divide-and-Conquer Algorithm using 2B policy

A comparison of performance between the 2B policy algorithm and Pohl's algorithm based on the number of comparisons to get the minimum and maximum elements from the sets of data is given in Table 1. The present algorithm, which applies to the present formula, will be known as model 1 and the existing algorithms (for $n = 2^k$ and $n \neq 2^k$) given by Pohl [4] will be known as model 2. The performance of model 1 compared to model 2 for the case of $n = 2^k$ is the same due to the same number of comparisons. For instance, let n = 16, from Pohl's algorithm, there will be a total of 16+ 16/2 - 2 = 22 comparisons. Similarly, by considering equation (5a), since $16 \leq 3^* 2^3$,

$$f(16) = 2^3 f(2)+(16-2^{3+1})f(1)+ 2(16-1)-2^{3+1}$$
$$= 8\, f(2) + 0f(1) + 30-16$$
$$= 8 + 14 = 22$$

However, for the case of $n \neq 2^k$ the performance is different, i.e., equation (6) has a smaller value than 2(n-1). This can be proved by the following proposition.

Proposition 1:
Let n be the number of data in a data set, if $f(n)$ as in (6), then $f(n) < 2(n-1)$.

Proof:
First, from equation (6a), since $2^k > 1$, where k is a non-negative integer,
then,

$$2n-2^k-2 < 2n-2$$
$$= 2(n-1).$$

Secondly, from equation (6b), since $2^{k+1} < 3.2^k < n$,
then,

$$n+2^{k+1}-2 < 2n-2 = 2(n-1).$$

Therefore, it is clear that the number of comparisons, using the 2B policy algorithm is smaller than the number of comparisons given by Pohl.

Table 1 shows the performance results obtained for $n \neq 2^k$. The processing time taken for model 1 and model 2 were running under the PC computer system with the 16.0 MB RAM memory.

Table 1: Performance comparison between model 1 (m1) and model 2 (m2). Number of elements varies from 1000 to 7000

No of elements (n)	No. of recursive calls		No of comparisons f(n)		Processing time (sec)	
	m1	m2	m1	m2	m1	m2
1000	511	999	1510	1998	3	6
2000	1023	1999	3022	3998	6	12
3000	1975	2999	4974	5998	11	17
4000	2047	3999	6046	7998	12	24
5000	2951	4999	7950	9998	18	30
6000	3951	5999	9950	11998	23	36
7000	4095	6999	11094	13998	24	41

It was found that model 1 shows a lower value of the number of comparisons when compared to model 2 for the case of the number of elements from a set of data is not in a power of 2. This is due to the least number of recursions of model 1 when compared to the number of recursions of model 2. For instance, when n = 7000, the number of recursions of formula 2 is 6999 whereas the number of recursions given by model 1 is 4095 making the number of comparisons of model 1, 21.05% lower than that of model 2, and also, the processing time of model 1 is 41.5% smaller than that of model 2. Hence the performance of model 1 is better than that of model 2 since the number of comparisons from model 1 are smaller than that of model 2 (Fig. 5 – Fig. 7).

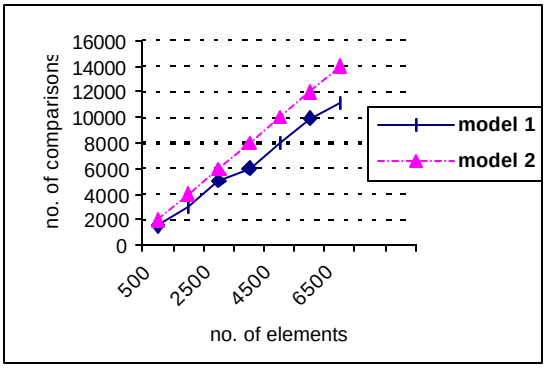

Fig. 5: Graph number of comparisons versus number of elements

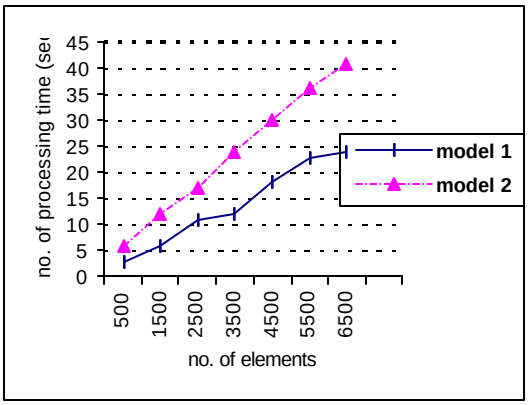

Fig. 6: Graph number of recursions versus number of elements

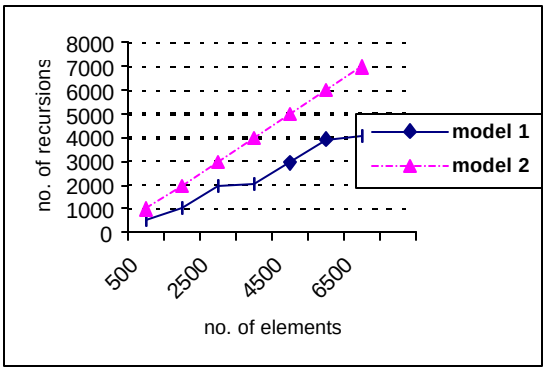

Fig. 7: Graph number of processing time versus number of elements

5.0 CONCLUSION

We have proposed a so-called 2B policy for the formula of calculating the number of comparisons in searching the minimum and maximum elements in a set of data of size n. As an implementation of the 2B policy, the divide-and-conquer algorithm was enhanced and its performance has been compared with that of Pohl's algorithm. It was observed that the enhanced divide-and-conquer algorithm developed in this paper under the 2B policy has performed equally at par when compared with Pohl's algorithm for data size in power of two and better when compared with data size not in a power of two.

6.0 REFERENCES

[1] J. V. Aho, J. E. Hopcroft and J. D Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.

[2] D. A. Bader, D. R. Helman, J. Jaja, "Practical Parallel Algorithms for Personalized Communication and Integer Sorting", *ACM Journal*, Vol. 1, No. 3, 1996.

[3] G. A. Gibson, J. S. Vitter and J. Wilkes, "Strategic Directions in Storage I/O Issues in Large-Scale Computing", *ACM Computing Survey*, Vol. 28, No. 4, December 1996.

[4] A. Kelley and I. Pohl, *C by Dissection, The Essentials of C Programming*, 2nd ed. Benjamin/ Cummings Pub. Company, 1992, p. 412.

[5] K. H. Rosen, *Discrete Mathematics and Its Applications*, 3rd ed, McGraw-Hill, Inc, 1995, p. 326.

[6] D. E. Vengroff, J. S. Vitter, "I/O-Efficient Algorithms and Environments", *ACM Computing Survey*, Vol. 28, No. 4, December 1996.

BIOGRAPHY

Mustafa Mat Deris received his M.Sc. in computer science from University of Bradford, England, in 1989. He is currently a lecturer in the Department of Computer Science, College University Terengganu, with interested in computer system performance and database.

Mohd Pouzi Hamzah obtained his M.Sc. in App. Science from University of Glassgow, Scotland, in 1987. Currently he is a lecturer in the Department of Computer Science and Information Technology, College University Terengganu, with interested in database.

Ali Mamat is currently a lecturer in computer science at University Putra Malaysia, Serdang. He obtained his Ph.D. in computer science from the University of Bradford, England, in 1992. His research interests include databases and logic programming.

APPENDIX 1.1

Let $f(n) = f(\lfloor n/2 \rfloor) + f(n-\lfloor n/2 \rfloor) + 2$ be the equation as in section (2).

where $\lfloor n \rfloor$ means the largest integer that is less than or equal to n. The above equation can be deduced recursively as follows:

$$= [f(\lfloor n/2^2 \rfloor) + f(\lfloor n/2 \rfloor - \lfloor n/2^2 \rfloor) + 2] + [f(\lfloor n/2 \rfloor - \lfloor n/2^2 \rfloor) + f(n-2\lfloor n/2 \rfloor + \lfloor n/2^2 \rfloor) + 2] + 2.$$

$$= f(\lfloor n/2^2 \rfloor) + 2f(\lfloor n/2 \rfloor - \lfloor n/2^2 \rfloor) + f(n-2\lfloor n/2 \rfloor + \lfloor n/2^2 \rfloor) + 2(3).$$

$$\vdots$$

$$= f(\lfloor n/2^k \rfloor) + kf(\lfloor n/2^{k-1} \rfloor - \lfloor n/2^k \rfloor) + \frac{k(k-1)}{2!} f(\lfloor n/2^{k-2} \rfloor - 2\lfloor n/2^{k-1} \rfloor + \lfloor n/2^k \rfloor) + \ldots + f(n-k\lfloor n/2 \rfloor + \frac{k(k-1)}{2!}\lfloor n/2^k \rfloor - \ldots + (-1)^k \lfloor n/2^k \rfloor) + 2\sum_{j=0}^{k-1} 2^j$$

$$= \sum_{i=0}^{k} \left(\binom{k}{i} f\left(\sum_{j=0}^{i} (-1)^{i-j} \binom{i}{j} \lfloor n/2^{k-j} \rfloor \right) \right) + 2\sum_{j=0}^{k-1} 2^j$$

APPENDIX 1.2

As the equation in Appendix 1.1, for the case where n is a power of 2, then $n-\lfloor n/2 \rfloor = \lfloor n/2 \rfloor$. Therefore,

$$f(n) = \sum_{i=0}^{k} \left(\binom{k}{i} f\left(\sum_{j=0}^{i} (-1)^{i-j} \binom{i}{j} 2^j \right) \right) + 2\sum_{j=0}^{k-1} 2^j$$

$$= f(1)\left[1 + k + \frac{k(k-1)}{2!} + \ldots + 1\right] + 2\sum_{j=0}^{k-1} 2^j$$

$$= f(1) + kf(-1+2) + \frac{k(k-1)}{2!} f(1-2(2)+4)) + \ldots + f(1) + 2\sum_{j=0}^{k-1} 2^j$$

$$= 2^k f(1) + 2\sum_{j=0}^{k-1} 2^j$$