The Analysis of Coordinate-recorded Merge-Sort based on the Divide-and-Conquer Method

Menghan Yan, Wenqian Shang, Mengyi Zhang
Communication University of China
School of Computer Science
Beijing, China
e-mail:524942428@163.com

***Abstract*—Many available algorithms are structurally recursive and can invoke the typical algorithm itself once or even more times to solve tightly related sub-problems. All of these algorithms follow the major principle called Divided-and-Conquer method, which first divides initial problem into several items same in goals but have a smaller scale. The Merge Sort uses this ideology to compass complexity and accelerate processing time. The coordinate-recorded merge sort algorithm continues the main idea, lets the processing problems get larger gradually, but directly works out solutions of the smallest sub-problems at the very start, in other words, the dividing process becomes useless.**

Keywords—divide-and-conquer method; merge sort; time complexity; coordinate recorded

I. INTRODUCTION

The Divided-and-Conquer method [1][2] is applied in solving the complex problems by firstly separating them into several items with a smaller scale, then researching each of them up to getting the specific solutions that can be returned back to the upper gradation. Merging [3] the resolutions of every sub-problem together is the last stage to find the final answer.

Randomness [4] is a kind of superior characteristic in finding out the best solution especially in sorting algorithm era. It often happens that concrete distribute information of inputs is blurry and even unbeknown, which probably can be an impediment in analyzing the average case. The insertion sort [5] will not highlight the special features of any queue waited to be treated even the numbers inside of the queue have been already ordered. What exactly will happen in a queue with many ordered numbers is that insertion sort will compare each element with every item in the sub-array already sorted, which can be seen as the worst case.

This kind of worst situation reveals an upper bound towards arbitrary inputs. The average case is usually as bad as the worst situation and it`s more valuable to calculate the mean of the time consuming expenses. Using more abstract and simplified processes can make insertion sort more easier to be analyzed such as applying constant c_i representing all the time cost that can be neglected of each sentence. $\theta(n^2)$ [6][7] expresses the time cost in the worst situation and constants a、b and c inside depend on sentence cost c_i. But what we really interest in is the items with greatest weight, for instance, in this formula, an^2 is the most important element in a sense including fastest increment speed.

If an algorithm has a lower growth rate compared with another one when used to solve the same problem, the former can be regarded as more efficient than the last one, especially the complexity of the problem waited to be solved is in a high level. The divide-and-conquer method compasses the height of the scale so that the speed can be accelerated prominently.

II. THE MERGE-SORT

A. The Pattern of Merge Sort

Merge sort algorithm completely comply with the operations that the divide-and-conquer method does.

① Divide: Divides a sequence having elements inside into 2 sub-sequences which respectively includes n/2 elements.

② Solve: Using merge sort arranges 2 sub-sequences recursively.

③ Merge: Merging 2 sub-sequences that already ordered into the final answer.

Everytime the length of sequence needed to be ordered develops into 1,the recursive operation starts to “rise”. There will be no work to be finished when the sequence can not be splitted anymore.

B. The Analysis of the Traditional Merge-Sort

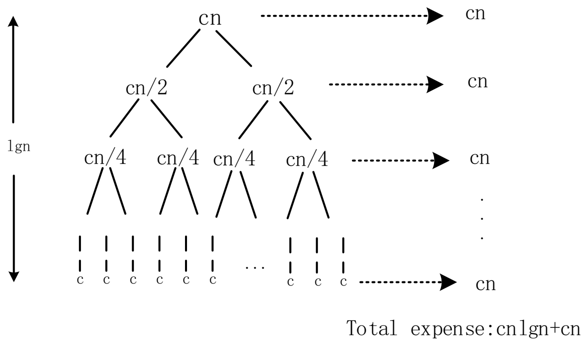
Supposing $T(n)$ represents the running time of a problem to be solved, and n represents the scale of the problem. If the scale is small enough it`s viable to get final result in a constant time recorded as $\theta(1)$ ,ignoring some coefficients inside.But it`s necessary and valuable to do some decomposing work and to turn the initial queue into a sub-queues,of which each element is 1-bth of the initial one.Supposing the time expense that decomposing problems needs is D(n),and merging operations needs is C(n),the final recursive formula can be expressed as followed.

$$T(n)=\begin{cases}\theta(1) & n\le c\\ aT(n/b)+D(n)+C(n) & else\end{cases} \tag{1}$$

Assuming that the scale of original problem is a power of 2,the analysis based on recursive formula will be effectively simplified.Every step of decomposing processes produces 2 sub-queue with the scale of n/2. $T(n)$ in this case can be expressed as followed.

$$T(n)=\begin{cases}\theta(1) & n=1\\ 2T(n/2)+\theta(n) & n\succ 1\end{cases} \tag{2}$$

The Master Theorem can be used to proof the solution of formula (2) is $T(n)=\theta(n\lg n)$ .In the worst situation,time cost $\theta(n\lg n)$ is definitely better than $\theta(n^2)$.



(3)

The element in the uppermost layer is divided into 2 smaller items, which are also divided using the same way. With the scale of the problem down to 1,solving every absolute sub-problem will cost c unit time.

The traditional proper order after applying recursive ideology in dealing sequencing problems is always from top to bottom, and then regarding the solutions of every layer as the inputs returned back to the upper layer until touching the ceil of the original queue again.

The following contents introduces the preudocode implementation of the **traditional** algorithm.

Algorithm 1.1 *MERGE-SORT*(A,p,q,r)

Input: A: an array not be sorted
p,r: the first and the last position of every array to be sorted
q: the middle position splits the array

Output: sorted array A

```
1  n1=q-p+1 , n2=r-q
2  let L[1...n1+1] and R[1...n2+1]be new arrays
3  for i = 1 to n1
4      L[i]=A[p+i-1]
5  for j=1 to n2
6      R[j]=A[q+j]
7  L[n1+1]=∞ , R[n2+1]=∞
8  i=1 , j=1
9  for k=p to r
10     if L[i]≤R[j]
11        A[k]=L[i]
12        i=i+1
13     else A[k]=R[j]
14        j=j+1
```

III. COORDINATE-BASED MERGE-SORT

The coordinate-based merge-sort starts to dispose the sequence of numbers directly from the “bottom”, and the scale of sub-queues to be merged is stationary corresponding to each layer of the recursion tree. It can be easily understood that the scale of every element at the relatively first layer is twice as big as the second layer`s. So it`s possible to establish an absolute region called ***GENERATOR*** that particularly be used to merge 2 sub-queues and generate a final result waiting to be submitted to the higher layer.

Algorithm 2.1 *GENERATOR*(A1,A2)

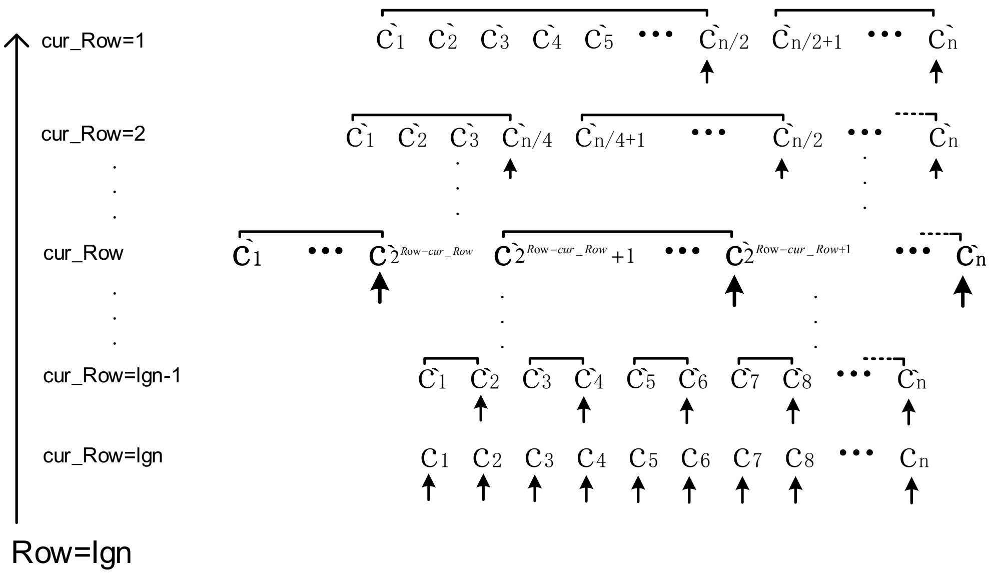
Input:A1,A2: in which the numbers have been already sorted
Output:A_{12}: combination of A1 and A2 including a serious of orderly fegures

```
1  let A12[1... A1.length …A1.length+A2.length] be new array
2  while i≠A1.length-1 or j≠A2.length-1
3    if A1[i] <A2[j]
4        A12[k]=A1[i]
5    else
6        A12[k]=A2[j]
7    Return A12
```

Algorithm 2.1 presents how 2 already sorted arrays merge into one final array in which the numbers are in the ascending order. Choose the smallest number between $A_1[i]$ and $A_2[j]$ as $A_{12}[k]$ (lines 3-6),and finally return array A_{12}.

Each input array size will become doubled with layer increasing and in the recursion tree where there is a strong correlation between the overall height, current layer and the scale of the queue unit.

As shown in the picture (4), the arrows pointing to some special numbers present the final figure of every absolute unit and also prevail the very starting. We can set 2 pointers immobilized at the first 2 places of the entire row that always indicate the impending inputs being transported into the GENERATOR. The upper layer will not start to work until there is no left *time* in the calculagraph.


cur_Row=1
cur_Row=2
cur_Row
cur_Row=lgn-1
cur_Row=lgn
Row=lgn
C`1 C`2 C`3 C`4 C`5 ••• C`n/2 C`n/2+1 ••• C`n
C`1 C`2 C`3 C`n/4 C`n/4+1 ••• C`n/2 ••• C`n
C`1 ••• C`2^Row-cur_Row C`2^Row-cur_Row +1 ••• C`2^Row-cur_Row+1 ••• C`n
C`1 C`2 C`3 C`4 C`5 C`6 C`7 C`8 ••• C`n
C1 C2 C3 C4 C5 C6 C7 C8 ••• Cn

Algorithm 2.2 *Array_Producer*(A)

Input: unordered numbers within the original array A

Output: ordered numbers within the original array A`

```
1  $cur\_Row=Row=\lg n$ , $gap\_Value=2^{Row-cur\_Row}$
2  while $cur\_Row \neq 0$
3      $time=2^{cur\_Row-1}$ , $cur\_time=1$
4      let $A_1[1\ldots gap\_Value+1]$ and $A_2[1\ldots gap\_Value+1]$ be
       new arrays
5      let $Coord[1\ldots 2^{cur\_Row}]$ be new array
6      $Coord[0]=gap\_Value-1$
7      for i=1 to $Coord.length-1$
8          $Coord[i]=Coord[i-1]+gap\_Value$
9      $first\_Hand=Coord[0]$, $second\_Hand=Coord[1]$
10     While $cur\_time \leqslant time$
11         $A_1[A_1.length-1]=\infty$ , $A_2[A_2.length-1]=\infty$
12         $sub\_S = GENERATOR(A1,A2)$
13         for i=0 to $sub\_S.length-1$
14             $S[first\_Hand-gap\_Value+1+i]=sub\_S[i]$
15         $first\_Hand=first\_Hand+2*gap\_Value$
16         $second\_Hand=second\_Hand+2*gap\_Value$
```

Let array *Coord*[] records the positional information, from where we acquire appropriate messages applied to decide which and how many numbers should be selected to be transported into the ***GENERATOR***, and of which the scale will be diminishing regularly with the layer increasing.

Supposing the inputs are n numbers, "*Row*" presents the total height of the recursion tree , "*cur_Row*" presents the order number of the layer being processed and *gap_Value* presents the quantity of numbers in every single unit. They have a mutual connection as follows.

$$Row=\lg n$$

cur_Row is *Row* down to 1

$$gap\_Value=2^{Row-cur\_Row}$$

The length of the array that records the positional information is 2^{cur_Row}.

The variable "*time*" in preudocode implementation presents how many times method ***GENERATOR*** will be invoked in the current layer and meanwhile "*cur_time*" represents the current order inside the entire frequency.

The analysis of the Coordinate-based merge-sort

The coordinate-based merge-sort algorithm leaves out the step of splitting the original array into smaller ones and directly process the leaf nodes at the very start.In every layer,totoal numbers will be conpared once in the

GENERATOR which process costs $\theta(n)$ .And there is a total of $\lg n$ *(Row)* layers . Every element will be processed by ***GENERATOR*** $\lg n$ times so that the entire time is $\theta(n)$.

IV. Conclusion

Merge Sort is an efficient, general-purpose, comparison based sorting algorithm in computer science. Most implementation preserves the input order of equal elements in the sorted output. Coordinate-recorded merge sort also uses the basic ideology of divide-and-conquer method that was invented by John von Neuman in 1945. The traditional merge sort is a top-down implementation which starts from the entire array and then gradually splits the original problem into unitary items. The coordinate-recorded merge sort just starts with the smallest unit that can not be divided and then return respective result up to the higher layer. The time cost is still $\theta\ (n)$.

Acknowledgment

This paper is partly supported by “the Excellent Young Teachers Training Project (the second level, Project number: YXJS201508)”, “Key Cultivation Engineering Project of Communication University of China (Project number: 3132016XNG1606 and 3132016XNG1608)”, “Cultural technological innovation project of Ministry of Culture of P.R.China (Project number: 2014-12)”, and partly supported by “The comprehensive reform project of computer science and technology, department of science and Engineering”. The research work was also supported by “Chaoyang District Science and Technology Project (CYXC1504)”.

References

[1] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein. Introduction to Algorithm 3th ed. 2013

[2] Qi Guo,Bo-Wei Chen, Feng Jiang, Xiangyang Ji, Sun-Yuan Kung. Efficient Divide-And-Conquer Classification Based on Feature-Space Decomposition. ArViv:1501.07584[cs.LG] Thu.29 Jan 2015

[3] S.Todd IBM United Kingdom Scientific Centre,Neville Road,Peterlee,
Durham SR8 IBY,England,UK. Algorithm and Hardware for a Merge Sort Using Multiple Processors. IBM Journal of Research and Development (vol:22,Issue:5) Sept.1978

[4] Gregory J.Chaitin. Randomness and Mathematical Proof. Scientific American 232,No 5(May 1975) pp. 47-52

[5] Sultanullah Jadoon,Salman Faiz Solehria. Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm:a Comparative Study.
International Journal of Electrical & Computer Sciences IJECS-IJENS
Vol:11 No:02.

[6] A.Davidson, D.Tarjan, M.Garland. J.D.Owens. Efficient parallel merge
sort for fixed and variable length keys. Innovative Parallel Computing (InPar),2012

[7] Svante Carlsson,Christos Levcopoulos,Ola Petersson. Sublinear merging
and natural merge sort. International Symposium SIGAL ’90
Tokyo,Japan, August 16-18,1990 Proceedings

[8] J. Shlens. A Tutorial on Principal Component Analysis. Center for Neural Science,New York University,Salk Institute for Biological Studies La Jolla, CA 92037 arXiv:1404.1000[cs.LG]

[9] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[10] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[11] K. Elissa, “Title of paper if known,” unpublished.