Islam, M.R.; Rana, S.M.M.; Hossain, M.M. and Nusrat, W. 2006. In-place merging and external sorting algorithm with no additional disk space. *Khulna University Studies*, 7(1): 1-12.

Khulna University Studies, 7(1): 1-12, 2006

REVIEW ARTICLE ENGINEERING & TECHNOLOGY

IN-PLACE MERGING AND EXTERNAL SORTING ALGORITHM WITH NO ADDITIONAL DISK SPACE

Md. Rafiqul Islam*, S. M. Masud Rana, Md. Monir Hossain and Wahida Nusrat

Computer Science and Engineering Discipline, Khulna Univesrsity, Khulna 9208, Bnagladesh

KUS –04/06-251104

Manuscript received: November 25, 2004; Accepted: May 7, 2006

Abstract: The performance of external sorting mainly depends on its I/O and time complexities. In this paper we represent a more efficient external sorting algorithm with no additional disk space. This algorithm uses Quick sort to produce runs in the first phase. It uses special in-place merging technique in the second phase. The algorithm excels in sorting a huge file, which is larger than the available internal memory of the computer. The algorithm creates no backup file for manipulating huge records. Usually the external sorting algorithm needs additional disk space in merging phase. This algorithm saves disk space, since it does not use any backup file or additional disk space. Here we have briefly reviewed the external sorting algorithms, which do not use additional disk space. The I/O and time complexities of the proposed algorithm are analyzed and compared with that of other similar algorithms. We have found that the I/O complexity of the proposed algorithm is less than that of other similar algorithms (algorithms which do not need additional disk space) with the same time complexity.

Key words: external sorting, runs, quick sort, algorithm, in-place merging

Introduction

External sorting is required when the number of records to be sorted is larger than the computer can grasp in its high-speed internal memory. The most common external sorting algorithm still uses the Merge Sort as described by Knuth (1973). In balanced two-way merge, runs (sorted records which can fit into the internal memory) are distributed over two external files and another two external files are used to hold the merged runs of the previous external files. After each merge the length of runs becomes double. When all the runs are merged into a single run, the process stops. The key drawback of this process is that it requires extra disk space. Dufrene and Lin (1992) proposed an algorithm in which no external file is needed; only the original file is used. Islam *et al.* (2003) proposed an algorithm that uses no extra disk space and faster than the algorithm proposed by Dufrene and Lin, but the output complexity is more than that of proposed by Dufrene and Lin. Here we proposed an external sorting algorithm with no additional disk space, which uses special in-place merging technique. The basic idea of our algorithm is based on the algorithm proposed by Dufrene and Lin (1992) and Islam *et al.* (2003). We have analyzed the I/O and time complexities of the proposed algorithm and the algorithm proposed by Dufrene and Lin algorithm as well as the algorithm proposed by Islam *et al.* (2003). We have shown that the I/O complexities of the proposed algorithm is less than that of Islam *et al*. (2003) algorithm as well as Dufrene and Lin algorithm and the time complexity of the proposed algorithm is less than that of Dufrene an Lin algorithm.

*Corresponding author. Tel: +88-(041)-720171-3/Extn. 344; e-mail: <dmri1978@yahoo.com>

DOI: https://doi.org/10.53808/KUS.2006.7.1.0406-E

There are many external sorting algorithms (Dufrene *et al.*, 1992; Islam *et al.*, 2003; Leu *et al.*, 2000; Wegner *et al.* 1989; Nodine *et al.*, 1995). However, two algorithms such as, algorithm proposed by Dufrene and Lin (1992) and the algorithm proposed by Islam *et al.* (2003) do not need additional disk space. On the other hand these two algorithms were designed based on generalization of bubble sort. Since the proposed algorithm is similar to these two algorithms, so we review the algorithms below.

An Efficient External Sorting Algorithm with no Additional Disk Space

This algorithm was proposed by Dufrene and Lin in 1992 (Dufrene *et al.*, 1992). They proposed an external sorting algorithm, which require no extra disk space. In this algorithm the external file is divided into equal sized blocks, which are approximately one half of the main memory (RAM). This is shown in Fig. 2.1.

Now,

The size of the main memory = M.

Block size, $B = M / 2$.

The size of external file = N.

Number of blocks in external file, $S = N / B$.

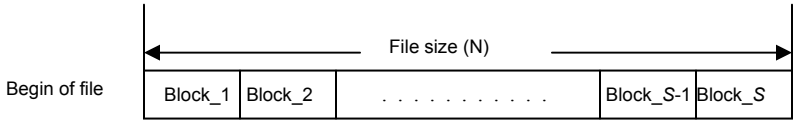

Fig. 2.1. External file after splitting into blocks.

At the first iteration Block_1 and Block_*S* are read into the lower half and upper half of memory array respectively. These two blocks are then sorted using Quick sort. The records of the lower half of the main memory contains the lowest sorted records of Block_1 and Block_*S* and the records of upper half of memory array are returned to Block_*S* area of external file. Now Block_*S*-1 comes into the upper half and the process continues. The loop terminates when the last block, Block_2 has been processed. Then Block_1 contains the lowest ordered records for the entire file. This approach is shown in Fig. 2.2.

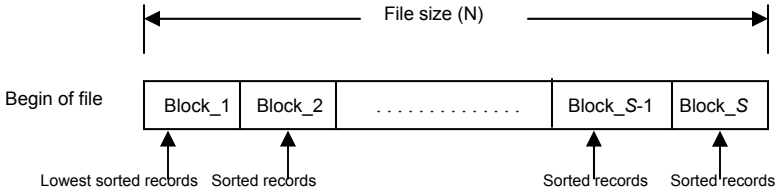

Fig. 2.2. External file after first phase.

The next iteration starts with Block_2. After this pass, as in the case of Quick sort, the size of the file is decreased by one block. The last two blocks to be processed are Blocks_*S*-1 and Block_*S*, upon the completion of which the entire file is sorted.

A New External Sorting Algorithm with no Additional Disk Space

This algorithm was proposed by Islam *et al.* (2003). It is also the generalization of internal Bubble sort. The algorithm works in two phases. In the first phase, this algorithm works as the algorithm proposed by Dufrene and Lin. That is, Block_1 and Block_*S* are read into lower half and upper half of the memory array respectively and they are sorted using Quick sort. This phase terminates when Block_2 is read into the upper half of memory array and sorted with the remaining records in the lower half of the memory array. After this, the algorithm switches to its second phase. In this phase, Block_*S*-1 and Block_*S* are read into the lower and upper half of the memory array respectively. Then the algorithm uses the special merging process. The used merging process is a special one because; the merging is accomplished in two steps. In the first step, merging is applied to sort the records (as both halves of memory array contain sorted records) of the lower and upper

half of memory array and the sorted records are written simultaneously in the position of Block_*S*-1 in the external file until the block is full. In the second step, the remaining records in the lower and upper half of memory array are again merged and the sorted records are written from the beginning of the upper half of memory array simultaneously. Now the upper half of the memory array contains the highest ordered records of Block_*S* and Block_*S*-1. Fig. 2.3 illustrates this approach.

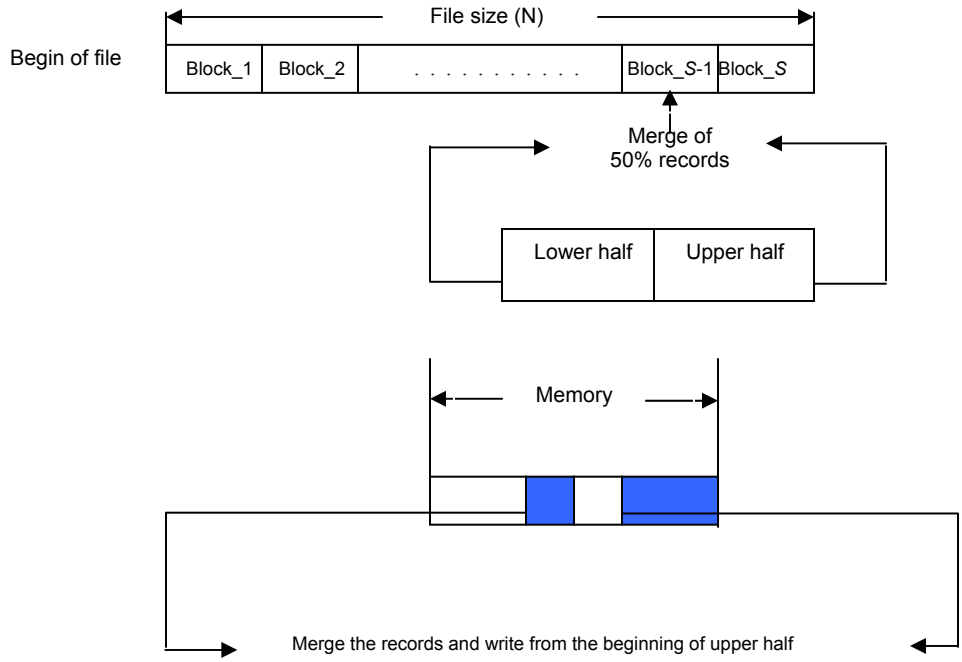

Fig. 2.3. Sorting by special merging technique

After this, Block_*S*-2 is read into lower half of memory array. In this way, when the last block, Block_2 has been processed, the upper half of memory array contains the highest sorted records of the entire file and they are written in the position of Block_*S* in the external file. The next iteration starts with Block_*S*-2 and Block_*S*-1 to be read into the lower and upper half of memory array respectively. At the end of this iteration, upper half of memory array contains the highest records among the blocks Block_2, Block_3, . . . , Block_*S*-1 and they are written in the position of Block_*S*-1 in the external file. After each pass, as in the case of the Bubble sort, the size of the external file is decreased by one block. The last blocks to be processed are Block_2 and Block_3, upon the completion of which the entire file is sorted.

The Proposed Algorithm

The proposed algorithm is also the generalization of the internal Bubble Sort. The algorithm works in two phases. In the first phase the algorithm works same as Dufrene and Lin's algorithm. The only difference is that, at the last iteration of the first phase, after sorting the records of Block_1 and Block_2 in the main memory using Quick sort, in Dufrene and Lin's algorithm the records of both lower and upper half of the main memory are written in the external file, but in the proposed algorithm the records of the lower half are written in the position of Block_1 of the external file and complete the first phase. Which will reduce the input complexity of the proposed algorithm. After first phase Block_1 contains the lowest sorted records among all the records and Block_3 to Block_S contain the individually sorted records. Which is illustrated in Fig. 3.1. After this the algorithm switches to its second phase.

In the second phase the records of Block_3 are read into the lower half of the main memory. Then the records of the lower half and upper half of the main memory, which are individually sorted, are merged using special in-place merging technique. The technique is special one, because it requires no extra space. In this technique

the records of the lower and upper half of the main memory are compared and smaller data is placed in its proper position until the lower half of the main memory becomes sorted.

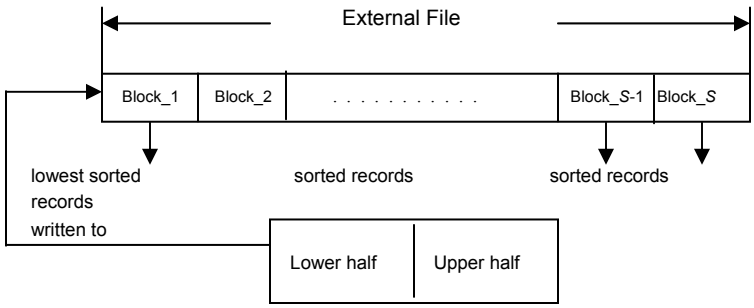

Fig. 3.1. External file and main memory after first phase

In this technique three-location pointers are needed. They are replace pointer (the position where the sorted data will be replaced), lower pointer (the position of the lowest unsorted data in the lower half of the main memory), and upper pointer (the position of the lowest unsorted data in the upper half of the main memory).

At first replace pointer and upper pointer means the first positions of the lower and upper half of the main memory respectively. After each comparison the position of the replace pointer is increased. Now, compare the data in the positions of replace pointer and upper pointer until upper pointer holds smaller data than that of replace pointer. Then swap the data in the positions of upper and replace pointers. Then assign the position of the upper pointer as lower pointer as well as increase the position of the upper pointer. After that, compare the contents of upper and lower pointers. If upper pointer holds smaller data than that of lower pointer then swap the contents of replace and upper pointers and increase the position of the upper pointer. Otherwise, assign the contents of lower pointer to the position of replace pointer. Then shift each of data in the range of the next position of the lower pointer and the previous position of the upper pointer by one position left. After that place the contents of replace pointer in the previous position of the upper pointer. This process continues until the position of the replace pointer becomes the first position of the upper half of the main memory. Then the lower half of the main memory is written to the position of Block_2 of the external file. After that the contents of the lower and upper pointers are compared and the smaller data are placed from the beginning of the lower half. Then increase the position of lower or upper pointer, which contains the smaller data. This process continues until all the data of the upper half become sorted. The example of this technique is shown in Table A-1 of Appendix A.

Then the records of Block_4 are read into the upper half of the main memory and after sorting using in-place merging technique the records of the lower half of the main memory are written to the position of Block_3. This process continues until the records of Block_S-1 are read into the main memory. When the records of Block_S are read into the lower half of the main memory, the records of the last positions of the lower and upper half are compared. The larger record is positioned from the ending of the upper half of the main memory using in-place merging technique until the upper half of the main memory becomes sorted. Then the records of the upper half of the main memory are written to the position of Block_S in the external file. After that the records of the lower half of the main memory are merged and placed the larger records from the ending of the upper half of the main memory. After this the first iteration of the second phase is completed. The example of this technique is shown in Table A-2 of Appendix A.

After each iteration the size of the external file is decreased by one block. In the last iteration only Block_2 and Block_3 are to be processed upon completion of which the entire file is sorted. The total procedure is shown in algorithm 3.1.

Algorithm 3.1

Algorithm for external sorting with no additional disk space using special in-place merging technique.

1. Declare blocks in the external file to be half of the main memory. Let the blocks be Block_1, Block_2, …, Block_S-1, Block_S.
2. Set $P = S$ and $Q = 1$.
3. Read Block_Q and Block_P to the lower half and upper half of the main memory.
4. Sort the data of the main memory using Quick sort, write upper half to Block_P and set $P = P-1$, read Block_P to the upper of the main memory and repeat this step until $P > 2$.
5. When $P = 2$ after sorting the main memory write the lower half of the main memory to block_Q and set $P = S$.
6. Set $Q = 3$.
7. Read Block_Q to the lower half of the main memory, merge the data of the main memory using In_Place_Merge( ).
8. I. Write the lower half of main memory to Block_Q.

   II. Then the records of upper half of the main memory are into two parts which are individually sorted, (one part is from the position '*k*' to the previous position of '*u*' and the other part is from position '*u*' to the end of main memory, the value of '*k*' and '*u*' can be get from In_Place_Merge( )) merge the records and write simultaneously from beginning of the lower half of main memory.

   III. Set $Q = Q + 1$ and go to Step 7 until $Q < P$.
9. When $Q = P$

   I. after performing Step 7 write the upper half of the main memory into Block_Q.

   II. Merge the records of the lower half (one part is from the beginning of the lower half to the position of '*k*' and the other part is from the next position of '*k*' to the position of '*u*') and write them to the upper half of the main memory.

   III. Set $P = P-1$ and go to Step 6 until $P <= 3$.

In_Place_Merge( )

```
{
// This procedure is used to merge the records in main memory. M represents the //main memory, n is the
number of records in main memory. k, u, r represent the //positions of lower pointer, upper pointer and
replace pointer of the main //memory respectively.
if ( Q < P) then {
r := 1; u := n/2 +1;
repeat r := r + 1 until (M[r] <= M[u])
swap (M[r], M[u]); r := r + 1, k := u; u := u + 1;
repeat {
   if (M[u] <= M[k]) then
        { swap(M[u], M[r]); u := u + 1; }
```

```
   else {  temp_value := M[r], M[r] :=M[k]; temp := k;
            repeat { M[temp] := M[temp+1]; temp := temp +1; } until ( temp < u)
            M[temp -1] := temp_value; }
   r := r + 1;
} until (r < = n / 2)
}  // end if when Q < P
else { // if Q = P
    r = n; k = n / 2;
    repeat r := r -1 until (M[r] >= M[k] )
  swap (M[r], M[k]); r := r - 1; u := k; k:= k - 1;
repeat  {
   if (M[k] >= M[u])  then
        { swap(M[u],  M[r]);  k := k - 1; }
   else {  temp_value := M[r], M[r] :=M[u]; temp := u;
            repeat { M[temp] := M[temp-1]; temp := temp -1; } until ( temp > k)
            M[k + 1] := temp_value; }
   r := r - 1;
} until (r > n/2)
}// end if when Q = P
}
```

Analysis of Complexities

In this section we will deduce the disk I/Os and the time complexities of the proposed algorithm.

***Input complexity*:** In the first phase $\frac{N}{B}$ blocks will have to be processed, so it will take $\frac{N}{B}$ read (input) operation in the first phase. After this phase Block_1 of the external file contains the lowest sorted records of the entire file. In the first iteration of the second phase the algorithm will take input from Block_3 to Block_S as after the first phase the records of the upper half of the main memory remain in the main memory. So, in the first iteration of the second phase it requires $\frac{N}{B} - 2$ read (input) operation. And after each iteration the number of the input operation is decreased by one. Thus in the second phase the number of input operation is $(\frac{N}{B} - 2) + (\frac{N}{B} - 3) + (\frac{N}{B} - 4) + \ldots + 1$.

So total disk input is

$$\frac{N}{B} + (\frac{N}{B} - 2) + (\frac{N}{B} - 3) + (\frac{N}{B} - 4) + \ldots + 1$$

$$= \frac{1}{2}\frac{N}{B}(\frac{N}{B}+1) - (\frac{N}{B}-1)$$

$$= \frac{N^2}{2B^2} - \frac{N}{2B} + 1$$

***Output complexity*:** In the first phase the algorithm will take $\frac{N}{B}-1$ write (output) operation. Because at the last step of the first phase after sorted the records of lower half of the main memory and Block_2, only the records of the lower half of the main memory written in the position of Block_1 and the records of the upper half remain in the main memory. In the first iteration of the second phase the algorithm will take $\frac{N}{B}-2$ write (output) operation. Because after merging the records of the lower half of the main memory and Block_S only the records of the upper half of the main memory are written in the position of Block_S. After each iteration of the second phase the number of output operation is decreased by one. So, the total output operation in the second phase is $(\frac{N}{B}-2)+(\frac{N}{B}-3)+(\frac{N}{B}-4)+\ldots+2$

So, the total output operation is

$$(\frac{N}{B}-1)+(\frac{N}{B}-2)+(\frac{N}{B}-3)+(\frac{N}{B}-4)+\ldots+2$$

$$= \frac{1}{2}(\frac{N}{B}-1)\frac{N}{B} - 1$$

$$= \frac{N^2}{2B^2} - \frac{N}{2B} - 1$$

***Time complexity*:** The time complexity of the internal Quick sort is $O(n \log_e n)$ in average case, (as given by Knuth [1]). Here n is the number of records to be sorted. So, the time complexity of the first phase of the proposed algorithm is $(\frac{N}{B}-1)(n \log_e n)$. In the second phase, the algorithm use special in-place merging technique. The time complexity of the merging technique depends on the number of comparison (as given by Knuth (1973). To merge n data using the proposed algorithm it will need n comparisons. So, the time complexity of the first iteration of the second phase is $(\frac{N}{B}-2)n$. And after each iteration the number of file size is decreased by one block. So, the time complexity of the second phase is

$$[(\frac{N}{B}-2)+(\frac{N}{B}-3)+(\frac{N}{B}-4)+\ldots+1]n$$

$$= n\sum_{i=1}^{N/B-2} i$$

Now, the total time complexity is $n\log_e n(\frac{N}{B}-1) + n\sum_{i=1}^{N/B-2} i$.

Discussion

In this section we will compare and discuss the complexities of the algorithms.

Complexities of the existing algorithms: The I/O and time complexities of Dufrene and Lin's algorithm and Islam *et al*. (2003) algorithm are deduced and shown in paper (Defrenc and Line, 1992) and paper (Islam *et al*., 2003) respectively. The complexities of those algorithms are shown in Table 1.

Table 1. Complexities of the algorithms proposed by Dufrene and Lin (1992) and Islam *et al.* (2003).

Complexity	Input	Output	Time
Dufrene and Lin (1992)	$\frac{N^2}{2B^2}+\frac{N}{2B}-1$	$\frac{N^2}{2B^2}+\frac{N}{2B}-1$	$n\log_e n \sum_{i=1}^{N/B-1} i$
Islam *et al.* (2003)	$\frac{N^2}{2B^2}+\frac{N}{2B}-1$	$\frac{N^2}{2B}+\frac{2N}{B}-\frac{3N}{2}+B-2$	$(\frac{N}{B}-1)n\log_e n + n\sum_{i=1}^{N/B-2} i$

Comparisons of input complexities: Let, the input complexities of Islam *et al*. (2003), Dufrene and Lin and the proposed algorithms are $I_R = \frac{N^2}{2B^2}+\frac{N}{2B}-1$, $I_D = \frac{N^2}{2B^2}+\frac{N}{2B}-1$ and $I_P = \frac{N^2}{2B^2}-\frac{N}{2B}+1$ respectively.

Now, $I_R - I_P = \frac{N^2}{2B^2}+\frac{N}{2B}-1-\frac{N^2}{2B^2}+\frac{N}{2B}-1$

$= \frac{N}{B}-2$

$= \frac{N-2B}{B}$

Here $N > 2B$. So, $I_R - I_P > 0$ or $I_R > I_P$. Since $I_R = I_D$, so $I_D > I_P$. Hence, the input complexity of the proposed algorithm is less than that of both Dufrene and Lin's algorithm and Islam *et al.* (2003) algorithm.

Comparisons of output complexities: Let, the output complexities of Islam *et al*. (2003), Dufrene and Lin (1992) and the proposed algorithms are $O_R = \frac{N^2}{2B}+\frac{2N}{B}-\frac{3N}{2}+B-2$, $O_D = \frac{N^2}{2B^2}+\frac{N}{2B}-1$ and $O_P = \frac{N^2}{2B^2}-\frac{N}{2B}-1$ respectively.

Now, $O_R - O_D = \frac{N^2}{2B}+\frac{2N}{B}-\frac{3N}{2}+B-2-\frac{N^2}{2B^2}-\frac{N}{2B}+1$

$= \frac{N^2(B-1)}{2B^2}+\frac{N(4-3B-1)}{2B}+B-1$

$= (B-1)(\frac{N^2}{2B^2}-\frac{3N}{2B}+1)$

$= (B-1)(\frac{N^2-3NB+2B^2}{2B^2})$

$= \frac{B-1}{2B^2}(N-B)(N-2B)$

For , $N > 2B$, $O_R - O_D$ is always positive *i.e.* $O_R - O_D > 0$. So, $O_R > O_D$.

Again, $O_D - O_P = \frac{N^2}{2B^2} + \frac{N}{2B} - 1 - \frac{N^2}{2B^2} + \frac{N}{2B} + 1$

$= \frac{N}{B}$

Which is always positive i.e. $O_D > O_P$. So, $O_P < O_D < O_R$ which means the output complexity of the proposed algorithm is less then that of both Dufrene and Lin's algorithm and Islam *et al*. (2003) algorithm. So, the proposed algorithm is better than Islam *et al*. (2003) algorithm as well as Dufrene and Lin's algorithm in case of output complexity.

For various external file sizes the reduction of output (write) operations of the proposed algorithm using special in-place merging technique from the algorithm proposed by Islam *et al.* (2003) is calculated and given in Table 2. Here the ratio of output operations, $\frac{O_P}{O_R} = \frac{\frac{N^2}{2B^2} - \frac{N}{2B} - 1}{\frac{N^2}{2B} + \frac{2N}{B} - \frac{3N}{2} + B - 2}$.

Here for $N = 320$, $B = 64$ we calculate from the previous equation that $O_P = 9$ and $O_R = 392$.

So, $O_p : O_R = 9 : 392$

$= 1 : 44$

So, reduction in percentage = $[(44-1) / 44] * 100$

$= 97.72$

Table 2. Results with output complexities of Islam *et al*. (2003) algorithm and the proposed algorithm.

External file size (N) (MB)	RAM size (M) (MB)	Ratio of output complexity	Reduction of output complexity (%)
320	128	1 : 44	97.72
640	128	1 : 53	98.11
1280	128	1 : 58	98.28
2560	128	1 : 61	98.36

Comparisons of time complexities: Let, the time complexities of Islam *et al*. (2003), Dufrene and Lin and the proposed algorithms are $T_R = (\frac{N}{B} - 1)n \log_e n + n \sum_{i=1}^{N/B-2} i$, $T_D = n \log_e n \sum_{i=1}^{N/B-1} i$ and $T_P = (\frac{N}{B} - 1)n \log_e n + n \sum_{i=1}^{N/B-2} i$ respectively. As $T_R = T_P$ and $T_R < T_D$ (collected from (Islam *et al*., 2003)), so $T_P < T_D$ i.e. the time complexity of proposed algorithm is less than that of Dufrene and Lin's algorithm.

Hence, the proposed algorithm is better than the algorithm proposed by Islam *et al.* (2003) in terms of I/O complexities. The proposed algorithm is also better than the algorithm proposed by Dufrene and Lin in terms of I/O complexities as well as time complexity.

Comparisons of the complexities: Let, C be the complexity of an algorithm and $C = I + O + T$ where I, O and T be the input, output and time complexities of the algorithm respectively.

Let, C_D and C_R be the complexities of the algorithms proposed by Dufrene and Lin and Islam *et al*. (2003) and C_P be the complexity of the proposed algorithm. So, we can write –

$C_D = I_D + O_D + T_D$, $C_R = I_R + O_R + T_R$ and $C_P = I_P + O_P + T_P$.

Now, from the previous sections $I_P < I_D$ and $I_P < I_R$; $O_P < O_D$ and $O_P < O_R$; $T_P < T_D$ and $T_P = T_R$. We know the performance of external sorting algorithm depends on input, output and time complexities. As, $I_P < I_D$, $O_P < O_D$ and $T_P < T_D$, so $C_P < C_D$ *i.e*. the total complexity of the proposed algorithm is less than that of Dufrene and Lin's algorithm. Again, $I_P < I_R$, $O_P < O_R$ and $T_P = T_R$ So, $C_P < C_R$ *i. e.* the total complexity of the proposed algorithm is less than that of Islam *et al*. (2003) algorithm.

Table 3 shows the experimental result comparison of the proposed algorithm with the algorithm proposed by Islam *et al*. (2003). Both algorithms result have been implemented in Microsoft Visual C++ 6.0. in Pentium III microprocessor based PC. The executions times are related to C_P and C_R. That means we have taken total time for reading, sorting and writing data.

Table 3. Comparison of time for read, sort and write of the proposed algorithm with Islam *et al*. (2003) algorithm (experimental result).

External file size (MB)	No. of records in external file	Internal memory size (MB)	Used memory size (MB)	Time of Islam *et al.* (2003) algorithm (Seconds)	Time of the proposed algorithm (Seconds)
135	35389440	128	90	432	304
180	47185920	128	90	738	548
270	70778880	128	90	2196	1934

Conclusion

We have proposed an external sorting algorithm with no additional disk space using special in-place merging technique. Firstly Quick sort and then in-place merging technique is used in the proposed algorithm. The algorithm uses minimum comparisons to merge the records and creates no extra file. Moreover this algorithm reduces the I/O complexity of Islam *et al*. (2003) algorithm significantly. The time complexity of the proposed algorithm is less than that of Dufrene and Lin's algorithm and same as Islam *et al*. (2003) algorithm.

References

Adnan, M.N.; Islam, M.R.; Islam, M.N. and Hossen, M.S. 2202. A faster hybrid external sorting algorithm with no additional disk space. Proceedings of International Conference on Computer and Information Technology (ICCIT), 27-28 December, Dhaka, Bangladesh, pp. 6-9.

Dufrene, W.R. and Lin, F.C. 1992. An efficient sorting algorithm with no additional space. *Computer Journal*, 35(3): 308-310.

Islam, M. R.; Raquib Uddin, S.M., and Chinmoy, R. 2005. Computational complexities of the external sorting algorithm with no additional disk space. *International Journal of Computer, Internet and Management,* 13(3): 60-68.

Islam, M.R.; Adnan, M.N.; Islam, M.N and Hossen, M.S. 2003. A new external sorting algorithm with no additional disk space. *Information Processing Letters,* 86: 229- 233.

Knuth, D.E. 1973 *Sorting and Searching*, *the Art of Computer Programming*. Vol. 3, Addison–Wesley, Reading, MA., pp. 247-299.

Leu, F.C.; Tsai, Y.T. and Tang, C. 2000. An efficient external algorithm. *Information Processing Letters*, 75: 159-163.

Nodine, M.S. and Vitter, J.S. 1995. Greet sort: An optimal sorting algorithm for multiple disks. *Journal of the Association of Computing Machinery* (*ACM*), 42(4): 919-933.

Rana, M.S.M.; Hossain, M.M. and Nusrat, W. 2005. A new external sorting algorithm using special in-place merging technique with no additional disk, space. Undergraduate Thesis (unpublished), Computer Science and Engineering Discipline, Khulna University.

Uddin, S.M.R.; Islam, M.R. and Chinmoy, R. 2003. Complexities of an efficient external sorting algorithm with special cases, *proceedings of* C*onference on Computer and Information Technology (ICCIT)*, 19-21 December, Dhaka, Bangladesh, pp. 126-129.

Wegner, L. and Teuhola, J.I. 1989. The external Heapsort. *IEEE Transaction on Software Engineering*, 5(7): 917-925.

Appendix A

Table A-1. Example of special in-place merging technique except the last step for each iteration of the second phase.

Main Memory		Contents of lower pointer M[k]	Contents of upper pointer M[u]	Contents of replace pointer *M[r]*
Lower half	Upper half			
6, 13, 16, 27, 31, 35, 48, 65	14, 19, 23, 39, 41, 64, 79, 95	6	14	6
6, 13, 16, 27, 31, 35, 48, 65	14, 19, 23, 39, 41, 64, 79, 95	13	14	13
6, 13, 16, 27, 31, 35, 48, 65	14, 19, 23, 39, 41, 64, 79, 95	16	14	16
6, 13, 14, 27, 31, 35, 48, 65	16, 19, 23, 39, 41, 64, 79, 95	16	19	27
6, 13, 14, 16, 31, 35, 48, 65	27, 19, 23, 39, 41, 64, 79, 95	27	19	31
6, 13, 14, 16, 19, 35, 48, 65	27, 31, 23, 39, 41, 64, 79, 95	27	23	35
6, 13, 14, 16, 19, 23, 48, 65	27, 31, 35, 39, 41, 64, 79, 95	27	39	48
6, 13, 14, 16, 19, 23, 27, 65	31, 35, 48, 39, 41, 64, 79, 95	31	39	65
6, 13, 14, 16, 19, 23, 27, 31	35, 48, 65, 39, 41, 64, 79, 95	35	39	35

Table A-2. Example of special in-place merging technique in the last step of each iteration of the second phase

Main Memory		Contents of lower pointer M[k]	Contents of upper pointer M[u]	Contents of replace pointer *M[r]*
Lower half	Upper half			
6, 13, 16, 27, 31, 35, 48, 65	14, 19, 23, 39, 41, 64, 79, 95	65	95	95
6, 13, 16, 27, 31, 35, 48, 65	14, 19, 23, 39, 41, 64, 79, 95	65	79	79
6, 13, 16, 27, 31, 35, 48, 65	14, 19, 23, 39, 41, 64, 79, 95	65	64	64
6, 13, 16, 27, 31, 35, 48, 64	14, 19, 23, 39, 41, 65, 79, 95	48	64	41
6, 13, 16, 27, 31, 35, 48, 41	14, 19, 23, 39, 64, 65, 79, 95	48	41	39
6, 13, 16, 27, 31, 35, 39, 41	14, 19, 23, 48, 64, 65, 79, 95	35	41	23
6, 13, 16, 27, 31, 35, 23, 39	14, 19, 41, 48, 64, 65, 79, 95	35	39	19
6, 13, 16, 27, 31, 35, 19, 23	14, 39, 41, 48, 64, 65, 79, 95	35	23	14
6, 13, 16, 27, 31, 14, 19, 23	35, 39, 41, 48, 64, 65, 79, 95	31	23	23