A FASTER EXTERNAL SORTING ALGORITHM USING NO ADDITIONAL DISK SPACE

Md. Rafiqul Islam[+], Mohd. Noor Md. Sap[++], Md. Sumon Sarker[+], Sk. Razibul Islam[+]

[+]Computer Science and Engineering Discipline, Khulna University, Khulna-9208, Bangladesh.

[++]Faculty of Computer Science and Information System, University of Technology Malaysia, Skudai, Johor Bahru, Malaysia

dmri1978@yahoo.com, mohdnoor@fsksm.utm.my, sumonsrkr@yahoo.com, optimist_2195@yahoo.com

Abstract

The effective performance of the external sorting is analyzed in terms of both time and I/O complexities. This paper is concerned with a more efficient external sorting algorithm, where both the time and I/O (read and write) complexities have been reduced. The proposed method is a hybrid technique that uses quick sort and merge sort in two distinct phases. Both the time and I/O complexities of the proposed algorithm are analyzed here and compared with the complexities of the existing similar algorithms. The proposed algorithm uses special in-place merging technique, which creates no extra backup file for manipulating huge records. For this, the algorithm saves huge disk space, which is needed to hold the large file. This also reduces time complexity and makes the algorithm faster.

Keywords: complexity, external sorting, in-place merging, quick sort, runs.

1. INTRODUCTION

Sorting means to arrange records or data in an ascending or descending order. But when the size of the records become so much larger than the internal memory can hold at a time, then the situation arise to use external sorting. It is quite different from internal sorting, even though the problem in both cases is to sort a given file into increasing or decreasing order.

The most common external sorting algorithm still uses the merge sort as described by Knuth [1]. 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 [2] and M. R. Islam *et al*. [3] proposed their algorithms in which no external file is needed; only the original file is used. M. R. Islam *et al*. [4][5] proposed an algorithm that uses no extra disk space and faster than both the algorithms proposed by Dufrene and Lin [2] and M. R. Islam *et al*. [3]. Here we have proposed an efficient external sorting algorithm with no additional disk space based on the algorithms proposed by Dufrene and Lin [2], M. R. Islam *et al*. [4] and M. R. Islam *et al*.

[3][6]. The proposed algorithm will reduce both the time and I/O complexities of the algorithm proposed by M. R. Islam *et al*. [4][7].

2. ALGORITHM REVIEW

The proposed algorithm is based on the algorithms proposed by Dufrene and Lin [2], M. R. Islam *et al*. [4] and M. R. Islam *et al*. [3][7]. Among these algorithms, the overall performance of M. R. Islam *et al*. [4] is better than others. So, we will only review M. R. Islam *et al*. [4] algorithm in the next subsection.

2.1 A New External Sorting Algorithm with No Additional Disk Space with Special In-place Merging Technique

This algorithm proposed by M. R. Islam *et al*. [4], which works in two phases. In the first phase the algorithm works same as Dufrene and Lin's [2] algorithm with the only difference that at the last iteration of first phase, lower half of main memory is written to the position of Block_1 of the external file keeping Block_2 in the upper half of main memory. In the second phase, the records of Block_S are read into lower half of main memory. Then the records of lower and upper half of memory are merged using special In-place merging technique which requires no extra space. After sorting in the memory using In-place merging, records of the upper half of memory are written to Block_S and records of Block_S-1 are read into upper half of main memory. When the last block Block_3 has been processed, the lower half of main memory contains lowest sorted data among Block_2 to Block_S. Then the records of lower half of memory are written in the position of Block_2. This process continues until Block_S-1 and Block_S are processed. The I/O complexity of the algorithm is less than that of algorithm proposed by M. R. Islam *et al*. [3].

3. A FASTER EXTERNAL SORTING ALGORITHM USING NO ADDITIONAL DISK SPACE

The proposed algorithm works in several phases. In the first phase, the external file is divided into equal sized blocks, where the size of each block is approximately equal to the available main memory (RAM) of the computer. If the size of the available internal memory is M then the size of each block is M and if the size of external file is N then the number of block, S = N/M. Block_1 is read into memory. Then the records of the main memory are sorted using *quick sort* and again written to Block_1. The process continues until the last block, Block_S, has been processed. After this the proposed algorithm switches to its next phase.

Each sorted block is divided into two sub-blocks, B_1 and B_2. The sub block B_1 of Block_1 and sub block B_1 of Block_2 are read into the lower half and upper half of the memory array respectively. Then the records of the lower half and the upper half of the main

memory, which are individually sorted are merged using special In-place merging technique [4]. After sorting, the records of the upper half of the main memory are written to B_1 of Block_2 and the records of the sub block B_1 of Block_3 are read into the upper half of the main memory. Then the records of the main memory are again merged using special In-place merging technique and the records of the upper half of the main memory are written in the position of B_1 of Block_3 and read the records of sub block B_1 of Block_4 to the upper half of the main memory. Repeat this process until the records of sub block B_1 of Block_S are read into the upper half of the main memory and processed. Now the lower half of the main memory contains the lowest sorted records among the records from Block_1 to Block_S and is written in the position of B_1 of Block_1.

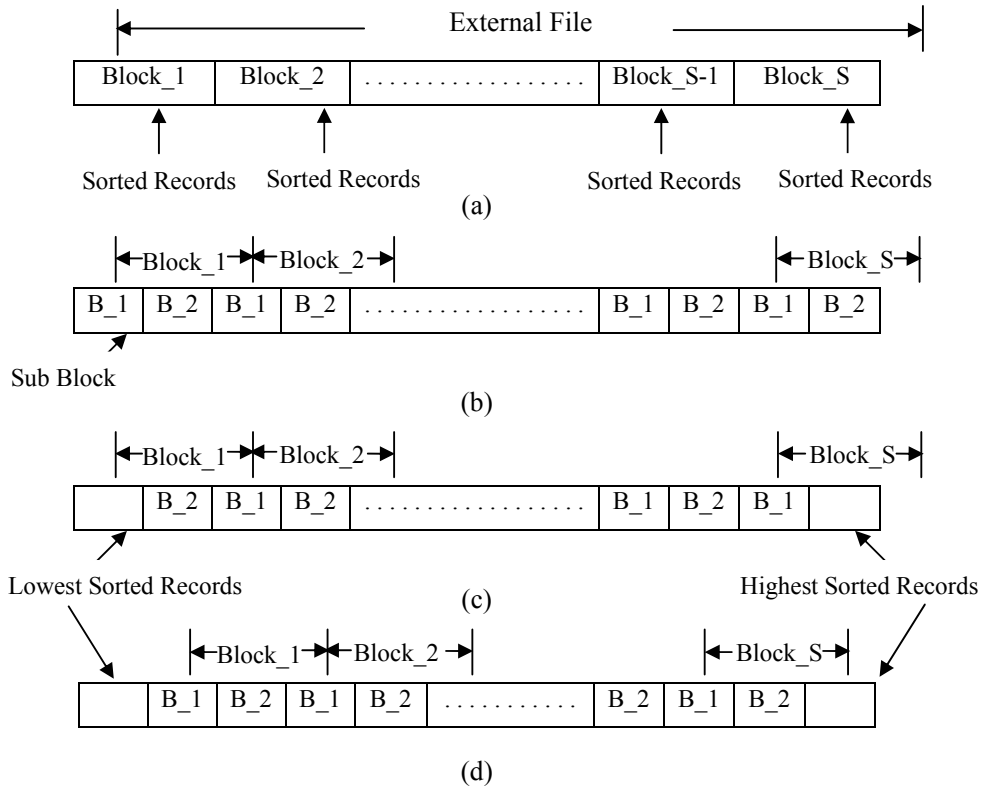

Figure 1. External file after (a) applying quick sort (b) splitting each block into sub blocks, (c) first iteration (d) renaming the blocks and sub blocks except first and last sub block.

Now sub block B_2 of Block_S and sub block B_2 of Block_S-1 are read into the upper and lower half of the memory array respectively. Then the records of the lower half and upper half of the main memory are merged using special In-place merging technique. After merging, the records of the lower half of the main memory are written to B_2 of Block_S-1 and the records of the sub block B_2 of Block_S-2 are read into the lower half of the main memory. Then the records of the main memory are again merged using special In-place merging technique and

the records of the lower half of the main memory are written in the position of B_2 of Block_S-2 and read the records of sub block B_2 of Block_S-3 to the lower half of main memory. Repeat this process until the records of the sub block B_2 of Block_1 are read into the lower half of the main memory and processed. Now the upper half of the main memory contains the highest sorted records among the records from Block_1 to Block_S and is written in the position of B_2 of Block_S.

At this point sub block B_1 of Block_1 and sub block B_2 of Block_S contains the lowest and highest sorted records respectively. Now, read B_2 of Block_1 and B_1 of Block_2 in the main memory and after merging, write lower and upper half at the position of B_2 of Block_1 and B_1 of Block_2 respectively. Now rename B_2 of Block_1 as B_1 and B_1 of Block_2 as B_2 and let both B_1 and B_2 are under Block_1. Similarly merge B_2 of Block_2 and B_1 of Block_3 and after renaming, let they are under Block_2. Repeat this technique until B_1 of Block_S has been processed. Let the last new Block be Block_S. Then apply the above procedure for the new blocks, Block_1 to Block_S, to get the next lowest and highest sorted records.

After each iteration the size of the external file is decreased by one sub block. Completing all iterations we get the entire file sorted. Algorithm 3.1 in Figure 2 brings out the whole procedure.

1. Declare blocks in the external file to be equal of the available main memory. Let the blocks be Block_1, Block_2, … , Block_(S-1), Block_S. set $P = 1$.
2. Read Block_P to the main memory.
3. Sort the data of the main memory using *quick sort*, write the sorted data to Block_P and set $P = P + 1$, if $P \leq S$ read Block_P to the main memory and repeat this step.
4. Divide each block into two sub blocks B_1 and B_2. Here each sub block equals the half of the available main memory.
5. Read sub block B_1 of Block_1 to the lower half of the available main memory. Set $Q = 2$.
6. Read sub block B_1 of Block_Q to the upper half of the available main memory.
7. Merge the data of the main memory using special In-place merging [3], write upper half to B_1 of Block_Q and set $Q = Q + 1$, if $Q \leq S$, read B_1 of Block_Q to upper half of the available main memory and repeat this step.
8. Write the lower half to B_1 of Block_1.

Figure 2. External Sorting Algorithm

9. Read B_2 of Block_S to the upper half of the available main memory. Set $Q = S - 1$.
10. Read B_2 of Block_Q to the lower half of the available main memory.
11. Merge the data of the main memory using special In-place merging [4], write lower half to B_2 of Block_Q and set $Q = Q - 1$. If $Q \geq 1$ read B_2 of Block_Q to lower half of the main memory and repeat this step.
12. Write the upper half to B_2 of Block_S.
13. From B_2 of Block_1 to B_1 of Block_S, declare new blocks in the external file to be equal of the available memory and again give the number from Block_1 to Block_S. Divide each block into two sub blocks B_1 and B_2. Here each sub block equals the half of the available main memory. Set $P = 1$.
14. Read B_1 and B_2 of Block_P to the main memory. Merge lower and upper half of Block_P. Write the sorted data to Block_P in the external file and set $P = P + 1$. If $P \leq S$, repeat this step.
15. If $S > 1$, go to step 5.

Figure 2. External Sorting Algorithm (cont'.)

4. COMPLEXITIES ANALYSIS OF THE PROPOSED ALGORITHM

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

4.1 Input Complexity

In the first phase N/M blocks will have to be processed by *quick sort*, so it will take N/M read (input) operations in the first phase. In the next phase to obtain the first lowest sorted sub block it will take N/M read operations and to obtain the first highest sorted sub block it will also take N/M read operations. Next (N/M – 1) blocks will have to be processed by special In-place merging to generate the new blocks. So it will take (N/M – 1) read operations. Then, to obtain the next lowest and highest sorted sub blocks it will take (N/M – 1) read operations in both cases and so on. So total disk input is

$$(\frac{N}{M} + \frac{N}{M} + \frac{N}{M}) + \{(\frac{N}{M} - 1) + (\frac{N}{M} - 1) + (\frac{N}{M} - 1)\} + \ldots + \{2 + 2 + 2\} + 1$$

$$\Rightarrow 3(\frac{N}{M}) + 3(\frac{N}{M} - 1) + \ldots + 3(2) + 1$$

$$\Rightarrow 3\{\frac{N}{M} + (\frac{N}{M} - 1) + \ldots + 2\} + 1$$

$$\Rightarrow 1 + 3\sum_{i=2}^{N/M} i \qquad \ldots\ldots\ldots \qquad (1)$$

4.2 Output Complexity

In the first phase the proposed algorithm will take N/M write (output) operations. In the next phase to obtain the first lowest sorted sub block it will take N/M write operations and to obtain the first highest sorted sub block it will take N/M write operation. Next (N/M – 1) blocks will have to be processed to generate the new blocks. So it will require (N/M – 1) write operations. Then, to obtain the next lowest and highest sorted sub block it will take (N/M – 1) write operations in both cases and so on. So the total output operation is

$$\{\frac{N}{M} + \frac{N}{M} + \frac{N}{M}\} + \{(\frac{N}{M} - 1) + (\frac{N}{M} - 1) + (\frac{N}{M} - 1)\} + \ldots + \{2 + 2 + 2\} + 1$$

$$\Rightarrow 3(\frac{N}{M}) + 3(\frac{N}{M} - 1) + \ldots + 3(2) + 1$$

$$\Rightarrow 3\{\frac{N}{M} + (\frac{N}{M} - 1) + \ldots + 2\} + 1$$

$$\Rightarrow 1 + 3\sum_{i=2}^{N/M} i \qquad \ldots\ldots\ldots \qquad (2)$$

4.3 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 $(N/M)(n \log_e n)$. In the next phase, the algorithm uses special In-place merging technique. The time complexity of the merging technique depends on the number of comparison (as given by Knuth [1]). To merge n data using special In-place merging technique it will need n comparisons. So the time complexity to obtain the first lowest and highest sorted sub block is $(N/M - 1)\ n + (N/M - 1)\ n = 2\ (N/M - 1)\ n$. As, after each iteration the file size is decreased by one sub block, so the total time complexity is

$$\{(\frac{N}{M}(n \log_e n) + 2(\frac{N}{M} - 1)n\} + [(\frac{N}{M} - 1)\, n + 2\{(\frac{N}{M} - 1) - 1\}n] + \ldots + [2n + 2\,(2-1)\, n] + n$$

$$\Rightarrow (\frac{N}{M})\,(n \log_e n) + 2n \sum_{i=1}^{N/M-1} i + n \sum_{i=1}^{N/M-1} i$$

$$\Rightarrow (\frac{N}{M})\,(n \log_e n) + 3n \sum_{i=1}^{N/M-1} i \qquad \ldots\ldots\ldots \qquad (3)$$

5. COMPARISON AND DISCUSSION

For simplicity of discussion we shall use the following notations for the algorithms (existing algorithms) to point out in this paper.

(a) Dufrene and Lin, "An efficient sorting algorithm with no additional disk space", [2], (Algorithm 1)

(b) M. R. Islam *et al.*, "A New External Sorting Algorithm with No Additional Disk Space with Special In-place Merging Technique", [4], (Algorithm 2)

(c) Proposed Algorithm, "A Faster External Sorting Algorithm Using No Additional Disk Space", (Algorithm 3)

The I/O and time complexities of the algorithms proposed by *Algorithm 1*, *Algorithm 2* and *Algorithm 3* are shown in Table 1.

Table 1: Complexities of the Algorithms proposed by Algorithm 1, Algorithm 2 and Algorithm 3

Complexity	Algorithm 1	Algorithm 2	Algorithm 3
Input	$\frac{N^2}{2B^2} + \frac{N}{2B} - 1$	$\frac{N^2}{2B^2} - \frac{N}{2B} + 1$	$1 + 3 \sum_{i=2}^{N/M} i$
Output	$\frac{N^2}{2B^2} + \frac{N}{2B} - 1$	$\frac{N^2}{2B^2} - \frac{N}{2B} - 1$	$1 + 3 \sum_{i=2}^{N/M} i$
Time	$n \log_e n \sum_{i=1}^{N/B-1} i$	$(\frac{N}{B} - 1) n \log_e n + n \sum_{i=1}^{N/B-2} i$	$(\frac{N}{M})\,(n \log_e n) + 3n \sum_{i=1}^{N/M-1} i$

Here the I/O and time complexity of *Algorithm 2* is better than that of *Algorithm 1*. So the overall performance of *Algorithm 2* is better than *Algorithm 1*.

So we shall compare and discuss the complexities of *Algorithm 2* and the *Algorithm 3*. Moreover, the value of M in the *Algorithm 3* equals to $2B$ in the *Algorithm 2*. So, while performing comparison M has been replaced by $2B$.

5.1 Comparison of Input Complexities

Let, the input complexity of *Algorithm 2*, $I_1 = \frac{N^2}{2B^2} - \frac{N}{2B} + 1$ and

the *Algorithm 3*, $I_2 = 1 + 3 \sum_{i=2}^{N/2B} i$

Now, $I_1 - I_2$

$= \frac{N^2}{2B^2} - \frac{N}{2B} + 1 - 1 - 3 \sum_{i=2}^{N/2B} i$

$= \frac{N^2}{2B^2} - \frac{N}{2B} - 3 \sum_{i=2}^{N/2B} i$

Here, for $N > 6B$, $(I_1 - I_2) > 0$ or $I_1 > I_2$. So, the input complexity of the *Algorithm 3* is less than that of *Algorithm 2* for $N > 6B$.

5.2 Comparison of Output Complexities

Let, the output complexity of *Algorithm 2*, $O_1 = \frac{N^2}{2B^2} - \frac{N}{2B} - 1$ and

the *Algorithm 3*, $O_2 = 1 + 3 \sum_{i=2}^{N/2B} i$

Now, $O_1 - O_2$

$= \frac{N^2}{2B^2} - \frac{N}{2B} - 1 - 1 - 3 \sum_{i=2}^{N/2B} i$

$= \frac{N^2}{2B^2} - \frac{N}{2B} - 2 - 3 \sum_{i=2}^{N/2B} i$

Here, for $N \geq 10B$, $(O_1 - O_2) > 0$ or $O_1 > O_2$. So, the output complexity of the *Algorithm 3* is less than that of *Algorithm 2* for $N \geq 10B$.

5.3 Comparison of Time Complexities

Let, the time complexity of *Algorithm 2*, $T_1 = (\frac{N}{B} - 1)n \log_e n + n \sum_{i=1}^{N/B-2} i$ and

the *Algorithm 3*, $T_2 = (\frac{N}{2B})\ (n \log_e n) + 3n \sum_{i=1}^{N/2B-1} i$

Now, $T_1 - T_2$

$$= (\frac{N}{B} - 1)n \log_e n + n \sum_{i=1}^{N/B-2} i - (\frac{N}{2B})\ (n \log_e n) - 3n \sum_{i=1}^{N/2B-1} i$$

$$= (\frac{N}{B} - 1 - \frac{N}{2B})n \log_e n + n \sum_{i=1}^{N/B-2} i - 3n \sum_{i=1}^{N/2B-1} i$$

$$= (\frac{N}{B} - 1 - \frac{N}{2B})n \log_e n + n \{ \sum_{i=1}^{N/B-2} i - 3 \sum_{i=1}^{N/2B-1} i \}$$

$$= (\frac{N - 2B}{2B})n \log_e n + n \{ \sum_{i=1}^{N/B-2} i - 3 \sum_{i=1}^{N/2B-1} i \}$$

Here, for $N > 2B$, $(T_1 - T_2) > 0$ or $T_1 > T_2$. So, the time complexity of the *Algorithm 3* is less than that of *Algorithm 2* for $N > 2B$.

6. THEORETICAL RESULTS:

The theoretical results of Algorithm 1, Algorithm 2 and Algorithm 3 have been shown in tabular format:

Table 2: Reduction of input Complexity of Algorithm 3 from Algorithm 1 and Algorithm 2

External file size (MB)	RAM size (MB)	Ratio (Algorithm 3 / Algorithm 1)	Reduction from Algorithm 1 (%)	Ratio (Algorithm 3 / Algorithm 2)	Reduction from Algorithm 2 (%)
400	200	0.77778	22.22	1.0000	0.00
600	200	0.80000	20	1.0000	0.00
800	200	0.80000	20	0.9655	3.45
1000	200	0.7963	20.37	0.9348	6.52
1200	200	0.79221	20.78	0.9104	8.96

Form the Table 2 we see that in case of Algorithm 1 if External File ≥ 200MB and in case of Algorithm 2 if External File > 600MB, reduction of reading input of Algorithm 3 in percentage is gradually increasing.

Table 3: Reduction of Output Complexity of Algorithm 3 from Algorithm 1 and Algorithm 2

External file size (MB)	RAM size (MB)	Ratio (Algorithm 3 / Algorithm 1)	Reduction from Algorithm 1 (%)	Ratio (Algorithm 3 / Algorithm 2)	Reduction from Algorithm 2 (%)
400	200	0.77778	22.22	1.4000	- 40.00
600	200	0.80000	20	1.1429	-14.29
800	200	0.80000	20	1.0370	-3.70
1000	200	0.7963	20.37	0.9773	2.27
1200	200	0.79221	20.78	0.9385	6.15

Form the Table 3 we see that in case of Algorithm 1 if External File ≥200MB and in case of Algorithm 2 if External File ≥1000MB, reduction of writing output of Algorithm 3 in percentage is gradually increasing.

Table 4: Reduction of Time Complexity of Algorithm 3 from Algorithm 1 and Algorithm 2

External file size (MB)	RAM size (MB)	Ratio (Algorithm 3 / Algorithm 1)	Reduction from Algorithm 1 (%)	Ratio (Algorithm 3 / Algorithm 2)	Reduction from Algorithm 2 (%)
400	200	0.3604	63.96	0.6498	35.02
600	200	0.2378	76.82	0.6000	40.00
800	200	0.1764	82.36	0.5845	41.55
1000	200	0.1455	85.45	0.5783	42.17
1200	200	0.1258	87.42	0.5774	42.25

From Table 4 we see that in case of both Algorithm 1 and Algorithm 2, if External file size ≥ 200MB, reduction of execution time of Algorithm 3 in percentage is gradually increasing.

7. EXPERIMENTAL RESULTS

The experimental results of Algorithm 1, Algorithm 2 and Algorithm 3 have been shown in tabular format below:

The configuration of the system where we have performed the experiment:

Operating System : Microsoft Windows XP Professional, Version 2002, Service Pack 2.

Processor : Intel (R) Pentium(R) 4 CPU 1.80GHz

Memory : 256MB RD RAM

Environment : Visual C++.Net 2003

Table 5: Experimental Results for Input

External File size (MB)	RAM Size (MB)	Input for Algorithm (1)	Input for Algorithm (2)	Input for Algorithm (3)	Reduction from Algorithm 1 (%)	Reduction from Algorithm 2 (%)
400	200	9	7	7	22.22	0.00
600	200	20	16	16	20.00	0.00
800	200	35	29	28	20.00	3.45
1000	200	54	46	43	20.37	6.52
1200	200	77	67	61	20.77	8.96

Table 6: Experimental Results for Output

External File size (MB)	RAM Size (MB)	Output for Algorithm (1)	Output for Algorithm (2)	Output for Algorithm (3)	Reduction from Algorithm 1 (%)	Reduction from Algorithm 2 (%)
400	200	9	5	7	22.22	- 40.00
600	200	20	14	16	20.00	- 14.29
800	200	35	27	28	20.00	- 3.70
1000	200	54	44	43	20.37	2.27
1200	200	77	65	61	20.77	6.15

Table 7: Experimental Results for Execution Time

External File size (MB)	RAM Size (MB)	Time for Algorithm (1) in minutes	Time for Algorithm (2) in minutes	Time for Algorithm (3) in minutes	Reduction from Algorithm 1 (%)	Reduction from Algorithm 2 (%)
400	200	15.11	15.31	11.14	26.27	27.24
600	200	33.22	28.52	21.21	36.15	25.63
800	200	55.43	43.23	32.25	41.82	25.40
1000	200	88.39	61.44	47.46	46.31	22.75
1200	200	125.53	88.12	64.37	48.72	26.95

8. CONCLUSIONS

We have improved the performance of the external sorting in the proposed algorithm in terms of both I/O and time complexities. The algorithm uses *quick sort* and In-place merging technique [4] and reduces comparisons to generate minimum and maximum sorted sub blocks of records and uses no extra file. Moreover, from the discussion in the previous section, we get the following results:

1. The time complexity of the Algorithm 3 is less than that of Algorithm 2.
2. The input complexity of the Algorithm 3 is less than that of Algorithm 2 for $N > 6B$.
3. The output complexity of Algorithm 3 is less than that of Algorithm 2 for $N \geq 10B$.

So, the overall performance of the Algorithm 3 is better than that of Algorithm 2 in terms of both I/O and time complexities.

REFERENCES

[1] D. E. Knuth, Sorting and Searching, "*The art of computer programming*" Vol. 3, Addison- Wesley, Reading, MA, 2nd edition, 1998.

[2] W. R. Dufrene, F. C. Lin, "*An efficient sorting algorithm with no additional space*", Compute. J. 35(3) (1992).

[3] M. R. Islam, S. M. R. Uddin, C. Roy, *"Computational complexities of the external sorting algorithm with no additional disk space",* International Journal of Computer, Internet and Management (IJCIM), Thailand, Vol. 13, Issue 3, September - December, 2005.

[4] M. R. Islam, W. Nusrat, M. Hossain, S.M.M. Rana, "*A New External Sorting Algorithm with No Additional Disk Space with Special In-place Merging Technique*", Presented at International Conference on Computer and Information Technology (ICCIT), 26 - 28 December 2004, (Dhaka, Bangladesh).

[5] M. R. Islam, N. Adnan, N. Islam, S. Hossen, "*A new external sorting algorithm with no additional disk space*", *Information Processing Letters* 86(2003), 229-233.

[6] N. Adnan, R. Islam, N. Islam, S. Hossen, "A faster hybrid external sorting algorithm with no additional disk space" published in the proceedings of International Conference on Computer and Information Technology (ICCIT), 27-28 December 2002 (Dhaka, Bangladesh)

[7] S. M. Raquib Uddin, Md. Rafiqul Islam, Chinmoy Roy, "Complexities of an efficient external sorting algorithm with special cases" published in the proceedings of Conference on Computer and Information Technology (ICCIT), 19-21 December, 2003 (Dhaka, Bangladesh).