International Journal of Computer Applications (0975 - 8887)
Volume 121 - No. 8, July 2015

Two Approaches of Natural Numbers Sorting: TAISN and Improved Array-Indexed Algorithms

Abdullah Sheneamer
Department of Computer Science
University of Colorado at Colo. Springs, USA
Jazan University, Saudi Arabia

Ahmed Alharthi
Department of Computer Science
University of Colorado at Colo. Springs, USA
Saudi Aramco, Saudi Arabia

Hanan Hazazi
Department of Computer IS
Regis University
Denver, CO USA

ABSTRACT

Data Structure is one of the fundamental areas of the computer science. Sorting is crucial in data structure, which creates the list of sequence items. In this paper, we present two techniques of sorting algorithm for natural numbers, which uses the array indexing methodology and insert that number into the proper index of the array without performing any element comparisons and swapping. The first algorithm improves Array-Indexed Sorting Algorithm for natural numbers [1] by adding negative numbers. The second algorithm is a new sorting algorithm that refers to Two Arrays-Indexed Sorting Algorithm for Natural Numbers (TAISN). The two techniques of sorting algorithm for natural numbers are efficient to give a much better performance than the existing sorting algorithms of the $O(n^2)$ class, for large array size with same length of digits of input data.

General Terms:

Data Structures, Algorithms

Keywords

Sorting , Natural Number Sorting.

1. INTRODUCTION

Many researchers in past years have tried to reduce the space and time of the sorting algorithms [11]. Many resources are exploited to propose a more working sorting algorithm. Up to date, there have been discussions aiming to improve the performance of the Sorting Algorithm. There are many sorting algorithms that are researched, developed, and introduced on this area. Sorting algorithms have three types of comparison (i) O (n^2), (ii) $O(n \log n)$, and (iii) O(n). The $O(n \log n)$ algorithms run faster than the $O(n^2)$ algorithms, but in general, $O(n^2)$ algorithms need lesser space on the RAM [1] because they are non-recursive. The $O(n)$ algorithms run faster than $O(n^2)$ algorithms. Therefore, in this paper, we propose TAISN sorting algorithm for natural numbers and improve Array-Indexed Algorithm [1] for decreasing the running time. The rest of the paper is organized as follows:

Section 2, which discusses the problem statement.
section 3, which discuses other related works.
Section 4, which discusses our proposed algorithms.
Section 5, which describes the complexity time analysis.
Section 6, which assesses the comparison of TAISN Algorithm and Array-Indexed Algorithm with existing sorting algorithms.
Section 7, which discusses the threats to validity,
Section 8, which is the conclusion and proposes the assessment of the topic in the future work.
Section 9 covers the experiment environment.

2. PROBLEM STATEMENTS

The problem of sorting is a general problem that frequently arises in the computer programming. There are many sorting algorithms that have been proposed to make sorting process faster. Generally, what we try to understand is what makes good sorting algorithms? Speed is probably the top consideration, but other factors of interest includes versatility in handling various natural numbers, consistency of performance, memory requirements, length and complexity of the code, and stability factors [5]. To assess all these factors, as well as answering our research question, we try to investigate and compare the majority of the existing sorting algorithms with the new developed sorting algorithm.

3. RELATED WORKS

There are many of sorting methods that have been published in the past years. Below is a brief description for some of these techniques:

3.1 Bubble sort

Bubble Sort is the simplest way of sorting, and is considered slowest. It works by comparing two elements at a time, and swapping them if they are in the wrong order. This method is considered relatively inefficient and is not used anywhere, except for theoretical purposes. For example, if there is a list of 100 elements, Bubble sort will make 10000 comparisons to sort the list. As previously mentioned, this sort of algorithm is inefficient because of present-day standers. In the best case, the Bubble sort has $O(n)$ behavior. In the best average and worst case, it has a complexity of $O(n^2)$ [2].

3.2 Selection Sort

Selection Sort is an inefficient algorithm on the larger list, but it works well on smaller ones. Moreover, Selection Sort is noted for its simplicity, and it has performance advantages over more complicated algorithms in certain situations, especially where auxiliary memory is limited. Basically, the algorithm scan the list for the smallest elements, puts them at the first location, and after that it selects the second smallest and puts to the second location, and so on until it reaches the largest elements in the list [3]. Selection Sort is an in-place comparison sort. It has $O(n^2)$ complexity, and that is why it is inefficient while processing large lists, and performs worse than the similar insertion sort [4].

3.3 Insertion sort

Insertion Sort is another simple sorting algorithm, and is considered to be efficient sorting technique on small list, but very slow algorithm in the large list. Also inserting algorithm is simple and easy to implement [5]. It works by selecting one element in every pass, and inserts it to the original location in the new list. The worst-case complexity of insertion sort is also $O(n^2)$. [6].

3.4 Quick Sort

Quicksort or partition-exchange sort is a well-known sorting algorithm developed by Tony Hoare. This algorithm selects an element as a pivot and re-arrange the list in such a way that all elements in the list that are greater than pivot elements comes after that. This algorithm is significantly faster in practice than other $O(n \log n)$ algorithms because its inner loop can efficiently be implemented on most architecture. The algorithm has $O(n \log n)$ behavior in both average and best cases while it gives $O(n^2)$ in the worst case performance [7].

3.5 Heap Sort

Heap Sort is a comparison based sorting algorithm. It is part of the selection sort family that creates sorted array (or list). It begins by building a heap out of the data set, and then removing the largest object and placing it at the end of the sorted array. After that, it reconstructs the heap, removes the largest remaining objects, and places it in the following open position form at the end of the sorted array. It repeats the process until there are no items left on the heap, and the sorted array [8]. Heap sort is considered being slow and is not a stable sort in practices on the most machines as compared to well implement quicksort. Heap sort algorithm has $O(n \log n)$ runtime in both best and worse cases.

3.6 Merge Sort

Merge Sort is an $O(n \log n)$ efficient comparison-based sorting algorithm. It is highly efficient and considered a stable sort. It works on divide and conquer concept. Merge Sort divides the un-sorted list into two parts; sorting the two-sub lists recursively by applying the merge sort again. In the end it merges the sub-list [9].

4. THE PROPOSED ALGORITHMS

We present two approaches of the array-indexed algorithms. The first approach is the future work of Array-Indexed Sort Algorithm for Natural Numbers [1], which contains sorting for both positive and negative numbers and the study of the performance of the algorithm with the other best sorting algorithms.

4.1 Improved Array-Indexed Sorting Algorithm

The first algorithm can be understood by the following example: n=10, A[10]={ -5,9, -4, -10,2, -3,0, 2 ,4,6}. Input data array *A* and the first *for* loop of Algorithm 1 which are used to find the MinValue and MaxValue from the input array which contains both negative and positive numbers.

Algorithm 1 Improved Array Indexed Sort(A[ ], n) [1]

```
Require: An unsorted array A[ ] of size n
Ensure: An sorted array A[ ] of size n
  $x \leftarrow 0, MinValue \leftarrow 0, MaxValue \leftarrow 0$
  for $i \leftarrow 0$ to $n-1$ do
    if $MinValue >= A[i]$ then
      $MinValue = A[i]$
    end if
    if $MaxValue <= A[i]$ then
      $MaxValue = A[i]$
    end if
  end for
  $PositiveMinValue \leftarrow \|MinValue\| + 1$
  $MaxPos \leftarrow MaxValue + PositiveMinValue$
  $B \leftarrow int[MaxPos]$
  for $i \leftarrow 0$ to $n-1$ do
    $A[n] \leftarrow A[n] + PositiveMinValue$
  end for
  for $i \leftarrow 0$ to $n-1$ do
    $p \leftarrow A[n]$
    $B[p] \leftarrow p$
  end for
  $m \leftarrow 0$
  for $i \leftarrow 0$ to $n-1$ do
    if $B[i]! = 0$ then
      $A[m] \leftarrow B[i] - PositiveMinValue$
      $m++$
    end if
  end for
```

Example 1

A[0]= -1
A[I]= 1
A[2]= -4
A[3]= $-6 \leftarrow MinValue$
A[4]= 2
A[5]= -3
A[6]= 5
A[7]= -2
A[8]= 4
A[9]= 6 $\leftarrow MaxValue$

In Example 1, MinValue = -6, MaxValue = 6. If MinValue is a negative value, it will be converted into positive value. *B* array length must be assigned to *MaxPos* that is a sum result of *MaxValue* and *PositiveMinValue*. The second *for* loop is used for add *PositiveMinValue* to each item of *A* array. The third *for* loop is used for filling *B* array by transfer every element into *A* array at its index position. Then, each element of *B* is subtracted from *PositiveMinValue* and copied into *A* array. Finally, we have a sorted array.

4.2 Two Arrays-Indexed Sorting Algorithm

The second algorithm, we present TAISN, which is a new Sorting Algorithm that sorts both negative and positive numbers even if a

Table 1. : Example1 (Improved Array-Indexed Sorting Algorithm)

Iteration	B Array	Sorted A Array
j= 0	0	0
j= 1	B[1] = 1	A[0] = -6
j= 2	0	0
j= 3	B[3] = 3	A[1] = -4
j= 4	B[4] = 4	A[2] = -3
j= 5	B[5] = 5	A[3] = -2
j= 6	B[6]=6	A[4] = -1
j= 7	0	0
j= 8	B[8] = 8	A[5] = 1
j= 9	B[9] = 9	A[6] = 2
j= 10	0	0
j= 11	B[11] = 11	A[7] = 4
j= 12	B[12] = 12	A[8] = 5
j= 13	B[13]= 13	A[9] = 6

list of values contains duplicate values and gives much better running time than the Array-indexed sort [8] and the existing sorting algorithms of the same complexity class such as Bubble Sort, Selection Sort, Insertion Sort, and Merge Sort.

Algorithm 2 TAISN Algorithm(A[ ], n)

```
Require: An unsorted array A[ ] of size n
Ensure: An sorted array A[ ] of size n
  $MaxValue \leftarrow 0$
  for $i \leftarrow 0$ to $n-1$ do
    if $MaxValue < A[i]$ then
      $MaxValue \leftarrow 1$
    end for
    for $k \leftarrow 0$ to $n-1$ do
      if $A[k] >= 0$ then
        $t = k$
        if $B[t] == Empty$ then
          $B[t] \leftarrow t$
        else
          $B[t] \leftarrow B[t] +$ "NewLine" $+t$
        end if$A[k] < 0$
        $t \leftarrow A[k].TrimStart('-')$
        if $C[t] == Empty$ then
          $C[t] \leftarrow " - " + t$
        else
          $C[t] \leftarrow C[t]$ + "NewLine" + "-"+ $t$
        end if
      end if
    end if
  end for
  for $L \leftarrow MaxValue$ to 0 do
    if $C[L]! = Empty$ then
      $A[L] \leftarrow C[L]$
    end if
  end for
  for $Z \leftarrow MaxValue$ to 0 do
    if $B[Z]! = Empty$ then
      $A[Z] \leftarrow B[Z]$
    end if
  end for
```

The algorithm can be explained in the same way of Example 1 that has an in Improved Array Indexed Sort Algorithm: n=10, A[10]={ -5,9, -4, -10,2, -3,0, 2 ,4,6}. Input data array A and the first five lines of Algorithm 2 are used to find the MaxValue from the Input array that contains also both negative and positive numbers.

Example 2

A[0]= -5
A[I]= 9 $\leftarrow MaxValue$
A[2]= -4
A[3]= -10
A[4]= 2
A[5]= -3
A[6]= 0
A[7]= 2
A[8]= 4
A[9]= 6

In Example 2, MaxValue=9. The first six lines of the second for loop are used to copy the positive numbers in B array by indexing the same number. The rest of the second for loop lines are used to copy the negative numbers in C array by indexing the same number. The third for loop is used to copy C array to A array from the last element to the first, and the forth for loop is used to copy B array to A array.

5. TIME COMPLEXITY ANALYSIS

The Array-Indexed Sort has two loops. The first loop is to find the minimum and maximum value, which has running of times O(n). The second loop has two loops: 1. outer loop from Minimum to Maximum elements, which has running numbers of Maximum value.

6. COMPARISON OF PRPOPSED ALGORITHMS WITH EXISTING SORTING ALGORITHMS

We compared the TAISN Algorithm and Improved Array-Indexed Algorithm with other sorting techniques to check the performance of the presented two algorithms. Various sizes and items were selected using random number generator with values ranging from -3000 to 3000. We found our Improved Array- Indexed Algorithm better in terms of performance and efficiency than our TAISN algorithm and other existing sort algorithms through a comparison

Table 2. : Example2 (TAISN Algorithm)

Index	Unsorted A Array	Sorted B Array	Sorted C Array	Sorted A Array
k= 0	A[0]= -5	NULL	C[5] = -5	A[0]=-10
k= 1	A[1]= 9	B[9]=9	NULL	A[1]=-5
k= 2	A[2]= -4	NULL	C[4] = -4	A[2]=-4
k= 3	A[3]= -10	NULL	C[10] = -10	A[3]=-3
k= 4	A[4]= 2	B[2] = 2	NULL	A[4]=0
k= 5	A[5]= -3	NULL	C[3] = -3	A[5]=2
k= 6	A[6]= 0	B[0]=0	NULL	A[6]=2
k= 7	A[7]= 2	B[2] = 2 ||2	NULL	A[7]= 4
k= 8	A[8]= 4	B[4] = 4	NULL	A[8]=6
k= 9	A[9]= 6	B[6] = 6	NULL	A[9]=9

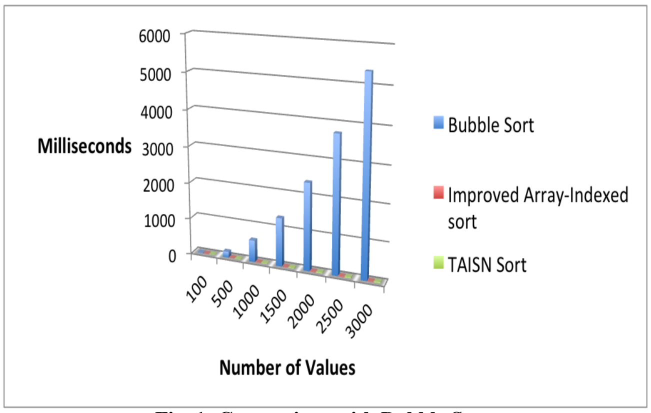

Fig. 1: Comparison with Bubble Sort

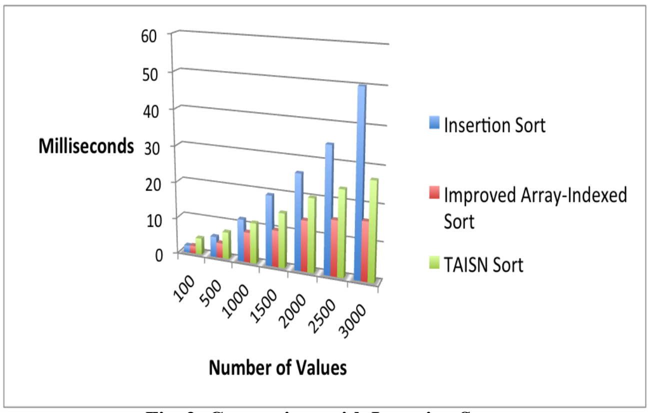

Fig. 2: Comparison with Insertion Sort

technique explained in the sub-sections below. We used the Stopwatch class in C.

6.1 Comparison with Bubble Sort

In the Fig 1 below, we placed elements number at x-axis and placed the execution time of the program in milliseconds at y-axis. We can see clearly that from 100 to 1000 elements the difference is not much, but when the size starts increasing after 500 elements to 3000, the bubble sort execution is increasing rapidly, while the TAISN sort shows much better performance than the Bubble Sort. Additionally, the Improved Array-Indexed Sort shows better efficiency and performance than Bubble Sort and TAISN Sort.

6.2 Comparison with Insertion Sort

In the Fig 2 below, we can clearly recognize that different performance gets bigger between Insertion Sort and the new two Sort Algorithms from 100 elements. As we see Insertion sort and TAISN sort. It is clear that the Improved Array-Indexed Sort is so much better performance than TAISN Sort and Insertion Sort after 100 elements.

6.3 Comparison with Selection Sort

In Fig 3, we can also see the difference is starting increase at 100 elements. We see Improved Array-Indexed Sort Algorithm is better performing than TAISN Sort Algorithm. Also the TAISN Sort Algorithm is so much better in performance than the Selection Sort Algorithm.

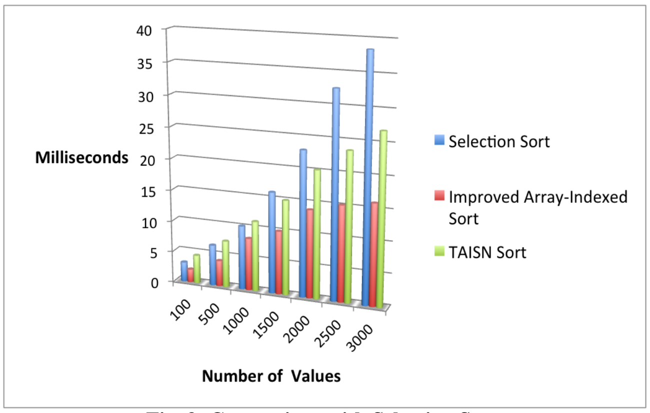

Fig. 3: Comparison with Selection Sort

6.4 Comparison with Merge Sort

In Fig 4, we can also observe that different executing time is increasing after a 100 elements. Also, Improved Array-Indexed Sort is better in terms of performance than the TAISN Sort in all the cases. The TAISN Sort is better in terms of performance than the Merge Sort. Also, the Improved Array-Indexed Sort performs better than the TAISN Sort and Selection Sort.

7. THREATS TO VALIDITY

The concern with the Improved Array-Indexed Algorithm is the accuracy, especially when we have duplicate values. If we have same values, the algorithm will override the old values. The TAISN Al-

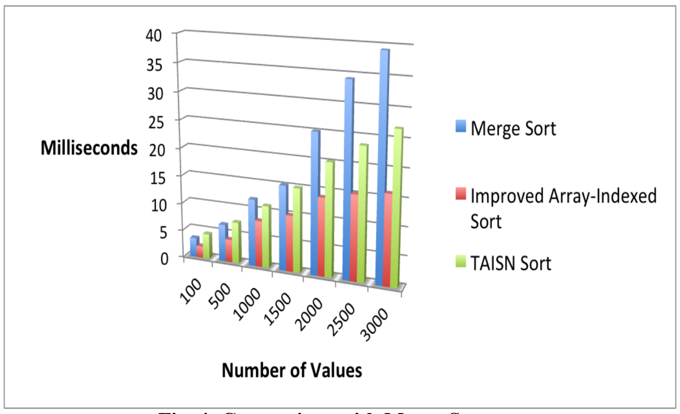

Fig. 4: Comparison with Merge Sort

gorithm works for duplicate values and same signs in the input because they deal with the duplicate values as strings. Our algorithms work with small numbers. The amount of space allocated for this algorithm would be dependent on the data and could be huge.

8. CONCLUSION FUTURE WORK

In this paper, we presented two approaches of the array-indexed algorithm, which are Improved Array-Indexed Sorting and TAISN Algorithms. The TAISN Algorithm gives a better executing time than the existing sorting algorithms. Overall, the evaluation shows that the TAISN Sorting Algorithm is very efficient for large numbers with the same length of elements. Also, the Improved Array-Indexed Sort Algorithm is better than the TAISN and the other existing sort algorithms in all cases. The future work includes addressing Improved Array-Indexed Sorting Algorithm accuracy when we have a problem on duplicate values and reducing the time complexity of Array-Indexed Sort Algorithm. Also, it can be an efficient and effective sorting algorithm than other existing algorithms.

9. EXPERIMENT ENVIRONMENT

We executed our experiments on an Intel Core i5 2.8 GHz machine with 4GB of memory 1600 MHz DDR3 running Mac OSX. The algorithm has been constructed using Xamarin Studio.

10. REFERENCES

[1] Babu, D. R., Shankar, R. S., Kumar, V. P., Rao, C. S., Babu, D. M., Sekhar, V. C. (2011, May). Array-indexed sorting algorithm for natural numbers. In Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on (pp. 606-609). IEEE.

[2] Donald K. The Art of ComputerProgramming, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley, 1997. ISBN 0-201- 89685-0. pp. 106-110 of section 5.2.2: Sorting by Exchanging.

[3] Butt, W. H., Javed, M. Y. (2008, December). A new Relative sort algorithm based on arithmetic mean value. In Multitopic Conference, 2008. INMIC 2008. IEEE International (pp. 374-378). IEEE.

[4] Seymour Lipschutz. Theory and Problems of Data Structures, Schaum's Outline Series: International Edition, McGraw- Hill, 1986. ISBN 0-07-099130-8., pp. 324-325, of Section 9.4:Selection Sort.

[5] Leiserson, C. E., Rivest, R. L., Stein, C. (2001). Introduction to algorithms. T. H. Cormen (Ed.). The MIT press.

[6] Seymour L . Theory and Problems of Data Structures, Schaum's Outline Series: International Edition, McGraw- Hill, 1986. ISBN 0-07099130 8., pp. 322-323, of Section 9.3: Insertion Sort.

[7] Hoare, C. A. R. ”Partition: Algorithm 63,” ”Quicksort: Algorithm 64,” and ”Find: Algorithm 65.” Comm. ACM 4(7), 321-322, 1961.

[8] Heap Sort.” Wikipedia. Wikimedia Foundation, 30 Nov. 2013. Web. 03 Dec. 2013.

[9] Merge Sort.” Wikipedia. Wikimedia Foundation, 12 Feb. 2013. Web. 03 Dec. 2013

[10] Knuth, D. E. (2006). Art of Computer Programming, Volume 4, Fascicle 4, The: Generating All Trees–History of Combinatorial Generation. Addison-Wesley Professional.

[11] Agarwal, A., Pardesi, V., Agarwal, N., Tech, M., Tech, C. N. M., DTU, D. B., BITS, S. (2013). A New Approach To Sorting: Min-Max Sorting Algorithm.International Journal of Engineering, 2(5).

11. APPENDIX A: ARRAY-INDEXED SORTING SORUCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
namespace Array_Indexed
{
        class Program
        {
                public static int PostiveMinValue;
                public static int MaxPos;
                public static int[] b;
static void Main(string[] args)
                {
                        Console.Write ("Enter_Input_Numbers:");
                        int xx = Convert.ToInt32(Console.ReadLine());
                        int[] a= new int[xx];

                        Random rng = new Random ();
                        for (int i = 0; i < xx; i++) {
                                string arr = rng.Next(-3000, 3000).ToString();
                                a[i] = Convert.ToInt32(arr);
                        }

                        Stopwatch StopWatch = new Stopwatch ();
                        StopWatch.Start ();

                        int MinValue = 0;
                        int MaxValue = 0;
                        for (int i = 0; i < a.Length; i++) {

                                if (MinValue >= a [i]) {
                                        MinValue = a [i];
                                }
                                if (MaxValue <= a [i]) {
                                        MaxValue = a [i];
                                }

                        }


                        PostiveMinValue = Math.Abs (MinValue) + 1;
                        MaxPos = MaxValue + PostiveMinValue;
                        b = new int[MaxPos + 1];

                        for (int n = 0; n < a.Length; n++) {
                                a [n] = a [n] + PostiveMinValue;
                        }

                        for (int m = 0; m< a.Length; m++) {
                                int p = a [m];

                                b [p] = p;
                        }
                        int index = 0;

                        for (int f = 0; f < MaxPos+1 ; f++) {
                                if (b [f] != 0) {

                                        a [index] = b [f] - PostiveMinValue;

                                        Console.WriteLine (a [index]);
                                        index++;
                                }
                        }

                        StopWatch.Stop();
                        Console.Write("Time:{0}", StopWatch.Elapsed.TotalMilliseconds);
                        Console.ReadLine();
                }
        }
}
```

12. APPENDIX B: TAISN SORTING SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
namespace New_Array_Indexed
{
  class Program

    {
        public static string[] a;
        public static string[] b;
        public static string[] c;
        public static int Max = 0;
        public static int x;
        public static int i = 0;
        public static Stopwatch stopWatch = new Stopwatch();
        static void Main(string[] args)
        {
            Console.Write("Enter_Input_Numbers:");
            x = Convert.ToInt32(Console.ReadLine());
          stopWatch.Start();
            a = new string[x];
            Random rng = new Random();
            for (int i = 0; i < x; i++)
            {
                a[i]= rng.Next(-3000, 3000) + "\r\n";
            }
           Max = 0;
          int ii = 0;
         while (ii < x)
            {
                int j = Convert.ToInt32(a[ii]);
                a[ii] = j.ToString();
                string substractMin = j.ToString().TrimStart('-');
                j = Convert.ToInt32(substractMin);
                if (Max < j)
                {
                    Max = j;
                } ii++;
            }
            b = new string[Max + 1];
            c = new string[Max + 1];
            for (int k = 0; k < x; k++)
            {
                int t = Convert.ToInt32(a[k]);
                if (t < 0)
                {
                    int f = t;
                    string s = t.ToString().TrimStart('-');
                    t = Convert.ToInt32(s);
                    if (c[t] == null)
                    {
                       c[t] = f.ToString();
                    }
                    else
                    {
                         c[t] = c[t] + "\r\n" + f;
                    }
                }
                else
                    if (b[t] == null)
                    {
                        b[t] = t.ToString();
                    }
                    else
                    {
                        b[t] = b[t] + "\r\n" + t;
                    }
            }
            Console.WriteLine( "");

foreach (string l in c)
            {
                if (l != null)
                {
                    Console.WriteLine(l);
                }
            }
             Console.WriteLine ("");
             foreach (string z in b)
             {
                 if (z != null)
                 {
                     Console.WriteLine(z);
                 }
             }
            stopWatch.Stop();
                Console.WriteLine ( "Time:" + stopWatch.Elapsed.TotalMilliseconds);
               Console.ReadLine();
           }
       }
}
```