International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 1 (2018) pp. 195-201

An Innovative Counting Sort Algorithm for Negative Numbers

Siddharth Srivastava [1]
Research Scholar, Department of Computer Science and Engineering,
Indian Institute of Technology, Kanpur (UP), India

Umesh Chandra Jaiswal
Professor, Department of Computer Science and Engineering
Madan Mohan Malaviya University of Technology, Gorakhpur, India.

Shachi Mall
Assistant Professor, Department of Computer Science and Engineering
Madan Mohan Malaviya University of Technology, Gorakhpur, India.
Orcid Id: orcid.org/0000-0002-4443-4885

Abstract

Counting sort is basically a linear time sorting algorithm. It sorts given array having integer elements ranging from 0 to K (where 'K' is upper limit of integer which can be present in input array). Counting sort can only be applied to positive integers and yields running time of Θ (n+K), where 'n' is length of input array and 'K' is highest integer which may be present in input array. Here in this paper we will introduce a modified version of counting sort which is capable of sorting both positive as well as negative integers, that too in linear time. Due to this the range of integers present in input array gets increased from (0 to K) to (-ve integer to +ve integer). The basics of this modification shows that counting sort can easily and efficiently be applied on mix of positive and negative integers, that too without hampering the running time of counting sort which remains linear as in previous case. We named this modified version of counting sort as negative version of counting sort. We have implemented this sort in our home-made software in which user can simply type integers in unsorted order and using negative version of counting sort software sorts the unsorted inputted array.

Keywords: Counting Sort, Negative Version of Counting Sort, Linear time, theta notation, running sum.

INTRODUCTION

Negative version of counting sort is the modified form of simple counting sort algorithm. As we all know that counting sort is incapable of sorting negative numbers because it uses mapping array or intermediate array (which is used to map unsorted input array to sorted output array). This paper presents the negative version of counting sort using negative numbers as well as positive numbers; the designed algorithm will sort the numbers as per the order of the numbers. This paper first, discusses about the basics of traditional counting sort and later on explains negative version of counting sort. Finally, it has been discussed with an example that this new version of counting sort is capable of sorting both negative and positive numbers in linear time.

LITERATURE SURVEY

Introduction to Algorithms by Thomas C Cormen [1] is a standard and word wide accepted book for design and analysis of algorithm and it is highlighted in the book that counting sort cannot work on negative numbers. Not a single paper is available in the literature that deals with negative numbers using counting sort algorithm. This paper presents a redesigned counting sort algorithm that it can sort the negative numbers given in the list.

1) Basics of Counting Sort

Counting sort is used to sort given input array (of size say 'n') consisting of positive integers (ranging from 0 to max) only. To sort this input array counting sort creates an array which can be termed as mapping array (of size max, where max is greatest positive integer present in input array). In this mapping array frequency of occurrence of each element present in input array is stored. Then some manipulations are performed on this mapping array (we will see later in this paper, how this is done). This mapping array is used to produce an output array which contains all elements present in input array, but in sorted (ascending or descending) order. Now let's explain the algorithm of counting sort.

2) Algorithm of Counting Sort

This section of the paper has given the original Counting Sort algorithm and later on the algorithm is explained with suitable example.

Algorithm of Counting Sort

```
Counting Sort(integer input[], integer output[], integer max {
    // input[] is input array containing +ve integers in unsorted order.
    // output[] is array which is produced as result of execution of counting sort and contains
       all elements of input array in sorted order.
    // max is maximum or greatest integer present in input array.
1.  integer n=input.length(); //Store length of input array in variable 'n'.
2.  integer mapping[max]; //array of size max.
3.  for(integer i=0;i<=max;i++)
4.  mapping[i]=0;  //initialize each element of array to 0.
5.  For(integer i=1;i<=n;i++)
6.  mapping[input[i]]=mapping[input[i]]+1;

    /* Loop to store frequency of occurrence of element in "input[]" array to array "mapping[]"using element of "input[]" array as index to array "mapping[]".*/
7.  For(integer i=1;i<=max;i++)
8.  mapping[i]=mapping[i-1]+mapping[i];
    //This Loop is used to produce "running sum" which tell that how many elements are less than or equal to 'i'.//
9.  For(integer i=n; i>=1;i--)
10. {output[mapping[input[i]]]=mapping[input[i]];
11. mapping[input[i]]=mapping[input[i]]-1;
12. }
    /* This loop produces the output array which contains all elements of input array in sorted order. Explanation of Above Listed Algorithm */
```

The above algorithm is the counting sort algorithm. The Lines bracketed in // or */ are comment lines. The algorithm is discussed with help of example. This algorithm Variable Description as follows:

i. **input[]**:- represents array which gets the input by the user and stores +ve integers ranging from 0 to max in unsorted way.

ii. **Output[]**:- This array represents an array which is produced as output of execution of counting sort algorithm on input[] array. It stores all elements of input[] array in sorted order.

iii. **max**: This variable is used to store highest integer element present in input[] array.

iv. **n**: This a variable used to store size of input[] array.

v. **mapping[]**: This is an intermediate array, using which input[] array is mapped to output[] array in sorted manned.

vi. **i**: Loop variable used for loop iteration.

vii. The line by line explanation of the above algorithm has been discussed in Table I. The input array has been defined as given below:

input[]={4,3,5,2,1,5,2,5,6} (Example value)

Table 1: The input array

Line Number	Explanation	Example
Line 1	store length of input array in variable 'n'	n=9 (from example value)
Line 2	mapping[max]	Create mapping[] array of size ranging from 0 to max. where max=6. (from example value)
Line 3-Line 4	Initialize elements of mapping[] array to zero.	mapping[]={0,0,0,0,0,0,0}
Line 5-Line 6	Loop to store frequency of occurrence of element in "input[]" array to array "mapping[]" using element of "input[]" array as index to array "mapping[]".	After execution of Loop value of mapping[]={0,1,2,1,1,2,1}. Index of array mapping[] 0 1 2 3 4 5 6; Value stored in array mapping[] 0 1 2 1 1 3 1
Line 7-Line 8	This Loop is used to produce "running sum" which tell that how many elements are less than or equal to 'i'.	After execution of Loop value of mapping[]={0,1,3,4,5,8,9}. Index of array mapping[] 0 1 2 3 4 5 6; Value stored in array mapping[] 0 1 3 4 5 8 9
Line 9-Line 11	Produces output[] array	After execution of loop value of output[]={1,2,2,3,4,5,5,5,6}

COUNTING SORT ALGORITHM

Having the knowledge of original counting sort, it becomes easy to understand the modified counting sort. The basic idea behind the counting sort is simple. The new algorithm uses two arrays namely "mapping_plus[]" and "mapping_minus[]" instead of "mapping[] which simply maps positive number to "output[]" array. The "mapping_plus[]" array stores the frequency of occurrence of positive integers present in "input[]" array whereas "mapping_minus[]" array stores the frequency of occurrence of negative integers present in "input[]" array. This concept is shown pictorially in Figure 1.

To understand the concept let us take an example. Suppose input array is input[]={3,-1,-2,0,2}and this is to be sorted in ascending order. The working of the new designed algorithm is discussed step by step below. Following is the complete algorithm steps and process.

Steps Process

1) Create mapping_plus[] and mapping_minus[] arrays of size 3 and 2 respectively. Because in input[] array highest element is 3 and lowest is -2. Since size of array cannot be negative so we initialize size of mapping_minus[] as 2 i.e. we take mod of lowest number present in array input[] if lowest number is negative. If lowest number is non-negative then algorithm works as simple counting sort algorithm.
2) Store frequencies of occurrence of +ve and –ve numbers in respective mapping arrays.i.e. mapping_plus[]={1,0,1,1} and mapping_minus[]={0,1,1}.
3) Now first perform "running sum" on –ve mapping array from i=mapping_minus.length to 0. This make mapping_minus[]={2,2,1}. Then perform running sum on +ve mapping array from i=0 to mapping_plus.length. This will make mapping_plus[]={1,1,2,3}.Now add mapping_minus[0]. To all elements of mapping_plus[]. This will make mapping plus as mapping_plus[]={3,3,4,5}. This is done to make the running sum consistent since –ve integers are followed by positive integers in sorting (ascending order). Since we use two arrays one for mapping –ve and other for mapping +ve numbers so positive array must have idea that how many –ve elements are there. This is done using step shown above.
4) Now map output array using +ve and –ve mapping arrays to get output[] array in sorted order. How this is done? This step will be clear when reader go through algorithm shown next.enterprises, the in-house business processes, the customer satisfaction, the measures to improve the financial results. It allows to answer four most important questions for the enterprise: how is it assessed by the clients (client's aspect)? Which processes could provide the enterprise with the unique position (in-house aspect)? How to achieve further improvements (innovation and learning aspect)? How is the enterprise assessed by the shareholders (financial aspects)?

1) Modified Counting Sort Algorithm

The following is the Modified Algorithm for negative numbers:

Algorithm of Modified Counting Sort

```
Modified_Version_of_Counting_Sort(integer input[], integer output[], integer max, integer min)
{/* input[]: An array used to store input given by the user which is only integer values.
output[]: An array used to store result after execution of modified version of counting sort.
max: A variable contains highest value present in "input[]" array.
min: A variable contains lowest value present in "input[]" array.
*/
1. if(min<0)
2. min=min*(-1) //making min +ve
3. integer mapping_plus[max],mapping_minus[min]
/* arrays of size max and min used to store frequency of +ve and –ve integers present in "input[]" array respectively */
4. for(integer i=0;i<=max;i++)
5. mapping_plus[i]=0; // initialize +ve mapping array to 0.
6. for(integer i=0;i<=min;i++)
7. mapping_minus[i]=0; //initialize –ve mapping array to 0.
8. for(integer i=1;i<=input.length;i++)
9. {if(input[i]>=0)
10. mapping_plus[input[i]]=mapping_plus[input[i]]+1;
11. else if(input[i]<0)
12. mapping_minus[input[i]*-1]=mapping_minus[input[i]*-1]+1;
13. }
/* Loop to store frequency of occurrence of +ve and –ve integers present in "input[]" array into "mapping_plus[]" and "mapping_minus[]" arrays respectively.*/
14. for(integer i=min-1;i>=0;i--)
15. mapping_minus[i]=mapping_minus[i]+mapping_minus[i+1];
/*Loop to perform running sum on –ve mapping array from min to 0.*/
16. for(integer i=1;i<=max;i++)
```

```
17. mapping_plus[i]=mapping_plus[i]+mapping_plus[i-1];
/*Loop to perform running sum on +ve mapping array from 0 to max.*/
18. for(integer i=0;i<=max;i++)
19. mapping_plus[i]=mapping_plus[i]+mapping_minus[0];
/*Loop to add running sum of 0th position of –ve mapping array to +ve mapping array elements in order to make +ve mapping running sum consistent. */
20. for(integer i=1;i<input.length;i++)
21. { if(input[i]>=0)
22. { output[mapping_plus[input[i]]]=input[i];
23. mapping_plus[input[i]]=mapping_plus[input[i]]-1;
24. }
25. else if(input[i]<0)
26. {output[mapping_minus[input[i]*-1]]=input[i];
27. mapping_minus[input[i]*-1]=mapping_minus[input[i]*-1]-1;
28. }
29. }
/*Loop to produce "output[]" array which stores all elements of "input[]" array but in sorted order.*/
}
```

2) Explanation of new Designed Algorithm

The following is the description of variables used in the newly designed algorithm. Variable Description:

i. **input[]**:- This represents an array which takes input by the user and stores +ve and –ve integers ranging from min to max in unsorted way.

ii. **Output[]**:-This represents an array which is produced as output of execution of negative numbers. This gives a new version of Counting Sort algorithm on input[] array. It stores all elements of your array in sorted order.

iii. **max**: variable used to store highest integer element present in input[] array.

iv. **min:** variable used to store lowest integer element present in input[] array.

v. **input.length**: used to represent size of input[] array (say n=input.length).

vi. **mapping_plus[]**: intermediate array using which +ve elements of input[] array are mapped to output[] array in sorted manned.

vii. **mapping_minus[]**: intermediate array using which -ve elements of input[] array are mapped to output[] array in sorted manned.

viii. **i**: Loop variable used for loop iteration.

ix. The line by line explanation of the above algorithm has been discussed in Table 2. The input array has been defined as given below:

input[]={3,-1,-2,0,2}; (Example Value)

Table 2: Explanation of the above Algorithm

Line Number	Explanation	Example
Line 1-Line 2	Check if min<0 then take mod of min i.e. min=min*-1.	max=3 and min=-2 (from example value) since min<0 so min=2 which is nothing but mod(-2) or mod(min)
Line 3	mapping_plus [max] & mapping_minus[min]	Create +ve and –ve mapping array of size ranging from 0 to max and 0 to min respectively. Using these arrays integers are mapped to output array in future steps. where max=3 and min=2. (from example value & Line 1-Line 2)
Line 4-Line 7	Initialize elements of mapping_plus [] and mapping_minus[] arrays to zero.	mapping_plus[]={0,0,0,0} & mapping_minus[]={0,0,0}
Line 8-Line 13	Loop to store frequency of occurrence of element in "input[]" array to arrays "mapping_plus[]" & "mapping_minus[]" using element of "input[]" array as index to array "mapping_plus[]" and "mapping_minus[]" arrays. "mapping_plus[]" array stores frequency of occurrence of +ve integers and "mapping_minus[]" array	After execution of Loop value of mapping_plus[]={0,0,1,1}.

Index of array mapping_plus[]	0	1	2	3
Value stored in array mapping_plus[]	1	0	1	1

Index of array mapping_minus[]	0	1	2
Value stored in array mapping_minus[]	0	1	1

<table>
<tr><td></td><td>stores frequency of occurrence of –ve integers present in “input[]” array.</td><td></td></tr>
<tr><td>Line 14-Line 15</td><td>This Loop is used to produce “running sum” on –ve mapping array which tell that how many elements are less than or equal to ‘i’.</td><td>After execution of Loop value of mapping_minus[]={2,2,1}.<table><tr><td>Index of array mapping_minus[]</td><td>0</td><td>1</td><td>2</td></tr><tr><td>Value stored in array mapping_minus[]</td><td>2</td><td>2</td><td>1</td></tr></table></td></tr>
<tr><td>Line 16-Line 17</td><td>This Loop is used to produce “running sum” on +ve mapping array which tell that how many elements are less than or equal to ‘i’.</td><td>After execution of Loop value of mapping_plus[]={1,1,2,3}.<table><tr><td>Index of array mapping_plus []</td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>Value stored in array mapping_plus []</td><td>1</td><td>1</td><td>2</td><td>3</td></tr></table></td></tr>
<tr><td>Line 18-Line 19</td><td>Loop to add value stored in mapping_min [0] to all elements of mapping_plus [] array. This tells mapping_plus [] array that these many –ve numbers are present in input array and make running sum of +ve array consistent.</td><td>After Execution of Loop value of mapping_plus[]={3,3,4,5}</td></tr>
<tr><td>Line 20-Line 29</td><td>Loop to map values stored in “input[]” array to “output[]” array using “mapping_plus[]” and “mapping_minus[]” arrays. When value of integer in input[] array is +ve then it is mapped using +ve mapping array. Similarly when integer value in input[] array is –ve it is mapped to output[] array using –ve mapping array.</td><td>After execution of Loop output array produced is output[]={-2,-1,0,2,3}</td></tr>
</table>

3) Analysis of Newly Modified and existing of Counting Sort

If we analyse simple counting sort algorithm given in section 2.1 then we will find that it’s running time is simply Θ(n), if length of input array is ‘n’. This analysis can easily be concluded by viewing core lines of sorting algorithm i.e. from line 9 to line 11. Since all other steps of algorithm can easily be computed during input phase i.e. when the system is reading input from input[] array and maximum length of input is known. Then mapping phase can also be computed. Only running sum loop and output loop is executed after input data is read by the system. Hence time complexity of this algorithm is Θ(n), a linear time complexity.

To compute the time complexity of the new designed algorithm, the algorithm discussed in section 3 is to be analysed line by line. If the algorithm is read is carefully, it will be found that if max and min are given in advance, then the computation from line 1 to 13 can be performed simultaneously with the reading of input data from array input[]. Loops in line 14,16 and 18 iterates for min, max and max time respectively. Let us assume that one iteration takes 1 unit time, then time required for the above mentioned three loops in (min+ 2*max). Loop in line 20 iterates for input.length time. If the size of array is ‘n’, then input.length=n, hence loop iterates for n unit of time. Therefore total time complexity is (min+2*max+n) which is linear in nature. If k=min+2*max, the execution time comes out to be (n+k). If k tends to n then the time complexity is θ(n+n) which is θ(2n) that equals to θ(n). These analysis shows that time complexity of new designed algorithm is linear and similar to existing counting sort algorithm.. Hence new designed version of counting sort executes in linear time to yield output consisting of positive as well as negative integers in sorted order.

IMPLEMENTATION INNOVATIVE COUNTING SORT ALGORITHM

This paper presents an innovative counting sort algorithm for negative numbers. To show the working of this new algorithm, software has been developed using Java Swing. This software consists of one input field that takes input values including negative numbers to sort the input values. To execute this software, one has to press simulate button. This software displays the step by step execution of this algorithm. Several output elements have been displayed using different colour scheme as given in Table 3.

CONCLUSION

Counting sort algorithm may further be modified for positive and negative real (decimal) numbers as future extension of the work carried out for this paper.

Table 3: Colour Scheme

Values	**Colour Schemes**
Values of +ve mapping array	Yellow
Value of –ve mapping array	Orange
Value of output array	Green
Initialize,+ve & -ve mapping, Runsum on +ve mapping, Runsum on –ve mapping, update runsum on +ve mapping, final output	White (with input array elements as gray)
Iteration 1, Iteration 2, … so on	#00FFFF colour (with input array elements as #00FFFF)
Table Showing colour combination schemes used in our software.	

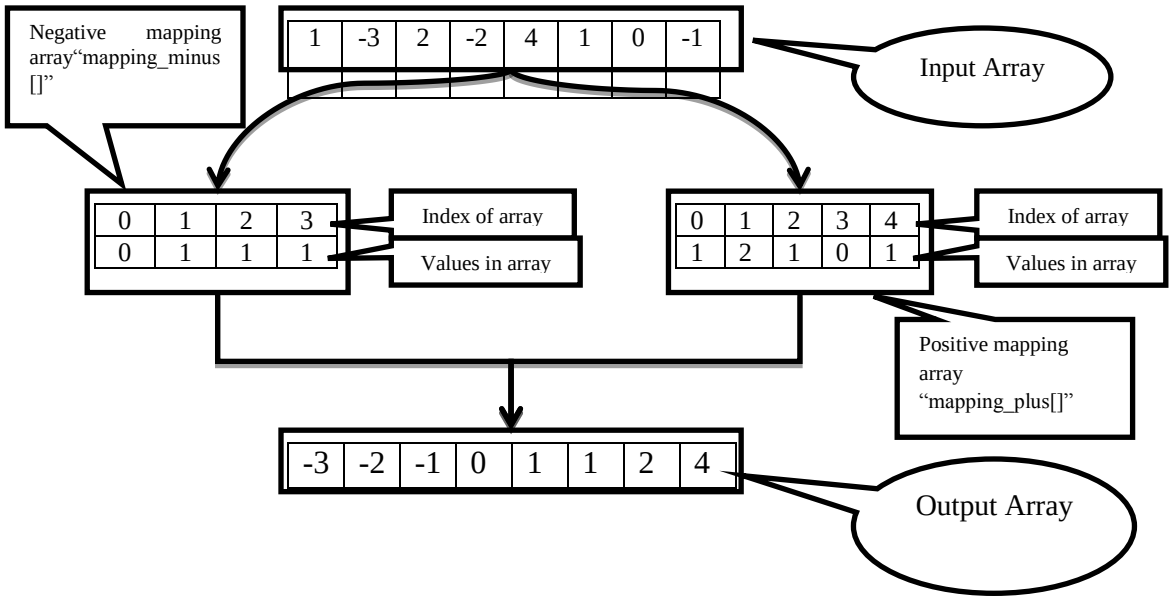

Figure 1: Shows concept used in Innovative of counting sort algorithm.

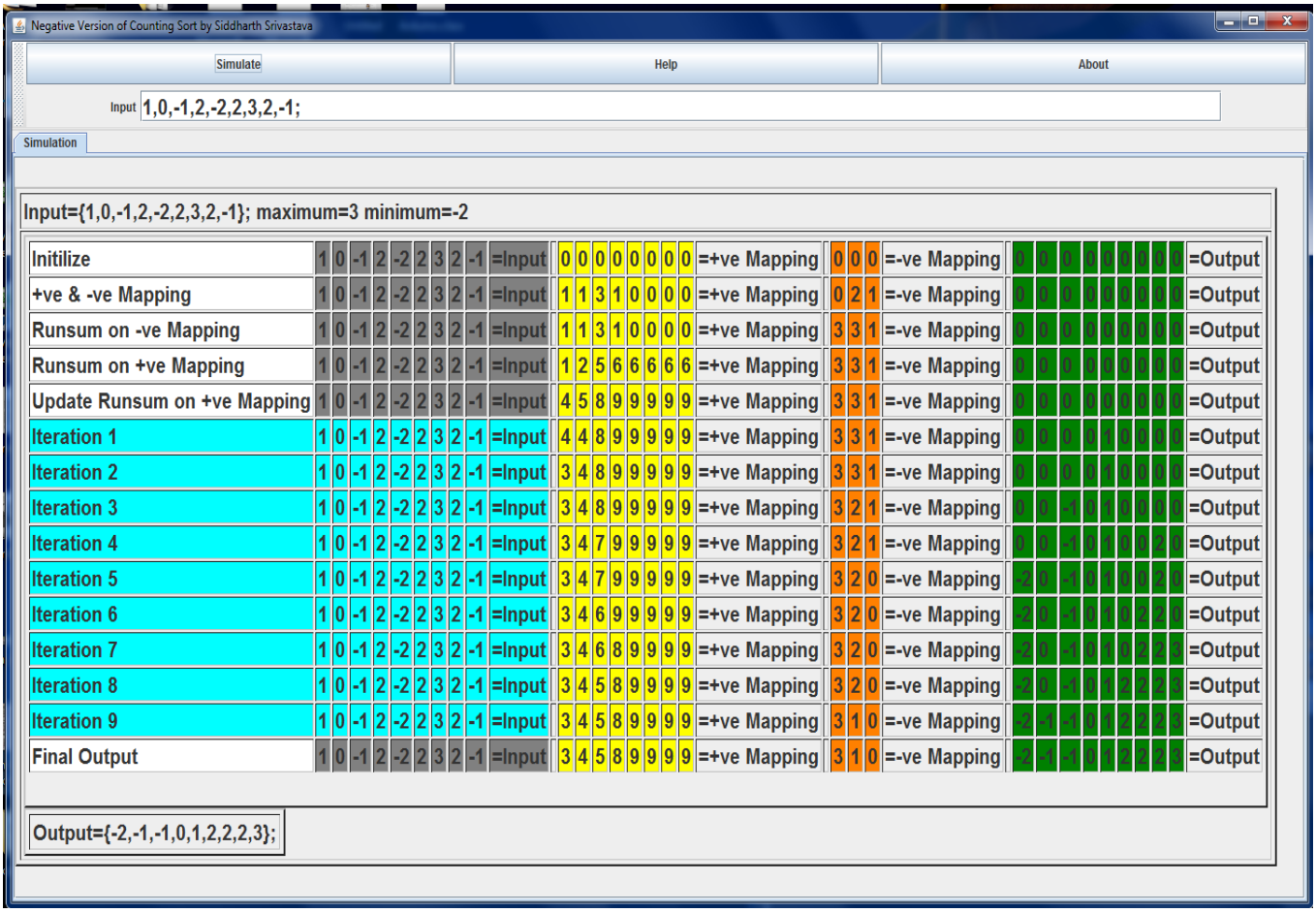

Figure 2: Snapshot of counting sort can sort an array of numbers consisting positive as well as negative numbers.

REFERENCES

[1] Thomas H. Cormen, C.E. Leiserson, R.L. Rivest,C.Strin,"Introduction to Algorithms",PHI Publications, 2nd Edition, 2007.

[2] Siddharth Srivastava, "Virtual Technology", Lambert Academic Publication, 2012.

[3] H. Schield, "Java The Complete Reference", Tata McHill Publication, 7th Edition, 2007.

[4] Edmonds, Jeff (2008), "5.2 Counting Sort (a Stable Sort)", How to Think about Algorithms,