International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075, Volume-2 Issue-6, May 2013

Flexi Sort Algorithm

Uthaya Vasanthan.S, Karthikeyan.B, Senthil Kumar.M

Abstract—the main objective of this project is to develop a sorting algorithm which forms an integral part of any domain, irrespective of the place in which the algorithm is used. And the scope of this system is not limited to a specific user group. Any user who wishes to perform sorting operation can make use of the system, given that the user has prior knowledge on basic computer and complier operation.

I. INTRODUCTION

The main purpose of this algorithm is to perform sorting of numeric values. We have many different algorithms to sort the numeric values but most of which follows a common pattern of direct comparison of the elements. But one has to understand the logic involved behind the comparison operator. The comparison operator makes the implementation of the sorting algorithm easier, but the real logic lies hidden deep inside. This algorithm aims at bringing out the hidden mathematical principles. Whatever may be the number of elements, the basic mathematical logic involved in deciding greatest of the two numbers is just by making the difference of two numbers (number at lower index – number at higher index). This basic mathematics led us in discovering this algorithm. This algorithm is an extension of this idea (the real logic behind the comparison operator). We also decided to make the algorithm user interactive. This led us to introduce a constant called "ran". The value of "ran" is obtained from the user. By keeping it as a base the numbers are sorted. The algorithm does not compare the two numbers directly and hence can be termed as "sorting without direct comparison". The user will also be intimated about the clock duration that the system takes to perform its action.

A. Problem description :

Sorting is a technique used in almost every application irrespective of the domain in which they are used. Most of the existing algorithms compare the two numbers to be sorted directly (with comparison operator). To sort the elements at position I and I+1 requires a comparison of the two elements, but how this sorting of element at position I and I+1 can be done without any direct comparison.

Consider the following case we have two numbers 50 and 20 at the positions I and I+1 respectively while this is being implemented in normal algorithms as 50>20 then swap the elements. But what is the inner logic behind the above case. The real mathematical logic behind the above case is the number at the position I+1 is subtracted from the number at position I.

Manuscript received May, 2013.

Uthaya Vasanthan.S, Student M.S III Year, SITE School, VIT University, Vellore, Tamil nadu, India.

Karthikeyan.B, Student M.S III Year, SITE School, VIT University, Vellore, Tamil nadu, India.

Senthil Kumar.M, Assistant Professor, SITE School, VIT University, Vellore, Tamil nadu, India.

If the obtained result is positive value (greater than 0) then swap the elements from their respective positions. If the obtained result is 0 (equal to zero) then retain the elements at their respective positions. If the obtained result is a negative value (less than zero) then retain the elements.

But how can this "0" be replaced by a random constant ("ran") that is obtained from the user. This intention led us to propose a new sorting algorithm.

II. DESIGN

- The sorting algorithm gets the elements from the user or the compiler generates it randomly by using the RANDOM () function.
- The constant (ran) generation is done by using the RANDOM ( ) function or can be obtained from the user.
- Then the difference between the numbers at the positions K and K+1 is performed.
- The result of the above operation is then subtracted from the constant (ran).
- If the resultant value of above operation is greater than (ran), then do not swap the elements.
- If the resultant value is equal to (ran), then do not swap the elements.
- If the resultant value is less than (ran), then swap the elements.

Algorithm Flow :

1-Get numbers to be sorted from the user.

2-Get the constant (ran) from the user.

3-Make the difference of the two numbers at position lower and higher respectively.

4-The result of Step (3) is then subtracted from the constant (ran).

5- The result of Step (4) is then compared with the original value of (ran)

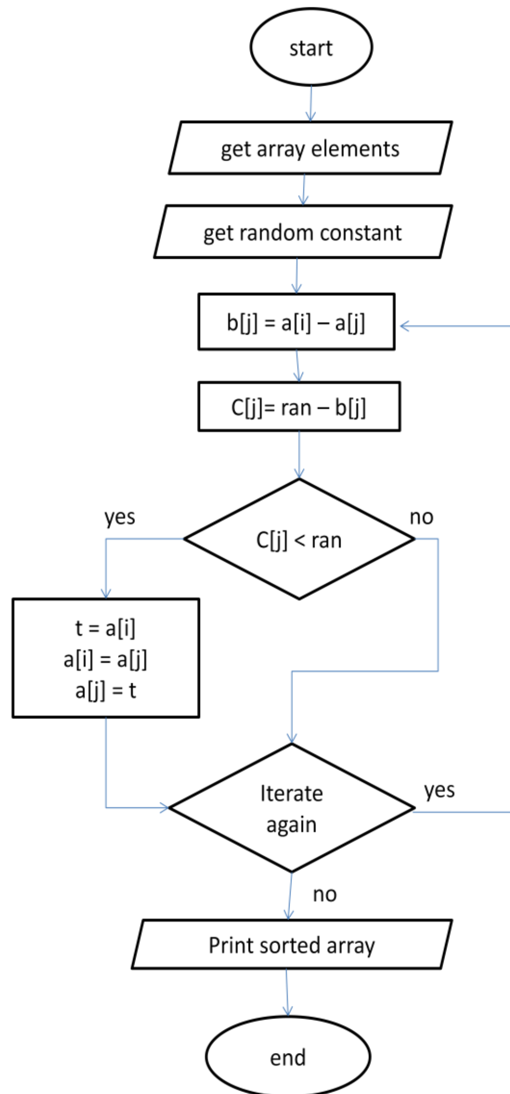
6-Apply the following conditions.

-If the result of step (4) is less than the original value of (ran), then swap the elements.

-If the result of step (4) is equal to the original value of (ran), then do not swap the elements.

-If the result of step (4) is greater than the original value of (ran), then retain the elements in their respective positions.

7-Repeat steps 3, 4, 5 and 6 until the entire sequence of number is sorted.



III. IMPLEMENTATION

The algorithm can be coded in any existing programming languages.

1-The random constant is either obtained from the user or it is generated by random function.

2-The difference between the elements in position I and J that is I + 1 is made. The result is stored in a variable say B [J].

B [J] =a [I]-a [J];

3-The result is then subtracted from the constant (ran). The result is then stored in a variable say C[j]

C [J] =ran-B [J];

4-Then the resultant value c [J] is compared with the constant (ran).

If (C [J] < ran)

5-The above loop implements the following constraint.

- If the value of C [J] is greater than (ran), then do retain the elements.
- If the value of C [J] is equal to (ran), then do not swap the elements.
- If the value of C [J] is less than (ran), then swap the elements.

This can be implemented as follows.

```
if(c[j]<ran)
{
    t=a[i];
    a[i]=a[j];
    a[j]=t;
}
```

IV. ALGORITHM IN PRACTICE :

Let us consider that the user wants to sort the following numbers

-> 3, 8, 5, 8

And the user choice of constant (ran) is “10”.

-Initial value of I=0

<u>Iteration 1:</u>

1 -Take first two elements 3 and 8. (I and I+1 respectively)

2 –Perform difference, the result is -5.

3- The resultant value “-5” is then subtracted from the random constant 10. The result will be 15 which is greater than the random constant “10”. So retain the element in its own position.

4-take the first element 3 (position I) and the element 5 (position I+2)

5-Perform difference, the result is -2.

6- The resultant value “-2” is then subtracted from the random constant 10. The result will be 12 which is greater than the random constant “10”. So retain the element in its own position.

7-Consider the elements “3” and “8”. (I and I+3 respectively)

8 –Perform difference, the result is -5.

9- The resultant value “-5” is then subtracted from the random constant 10. The result will be 15 which is greater than the random constant “10”. So retain the element in its own position.

10- At the end of first iteration the result will be 3, 8, 5, 8.

<u>Iteration 2:</u>

1 - Take second and third element 8 and 5 elements at position I+1 and I+2 respectively.

2 - Perform difference, the result is 3.

3- The resultant value “3” is then subtracted from the random constant 10. The result will be “7” which is less than the random constant “10”. So swap the elements.

4-now consider the element 5 and 8 at positions I+1 and I+3 respectively.

5 - Perform difference, the result is -3.

6- The resultant value “-3” is then subtracted from the random constant 10. The result will be “13” which is greater than the random constant “10”. So retain the elements.

7- At the end of second iteration the result will be 3, 5, 8, 8.

<u>Iteration 3:</u>

1 –Take third and fourth element 8 and 8(position I+2 and I+3 respectively).

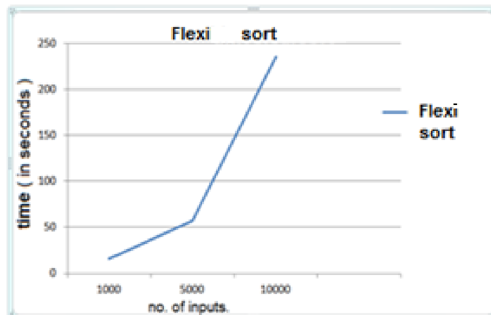
2 –Perform difference, the result is 0.

3- The resultant value “0” is then subtracted from the random constant 10. The result will be 10 which is equal to the random constant. So retain the element in its own position.

4- At the end of third iteration the result will be 3, 5, 8, 8.
5- If there are "n" numbers then "n-1" iterations are required to sort them. In the above case the value of n is 4. And hence the number of iterations required to sort them is n-1 i.e. is 3.

V. PERFORMANCE

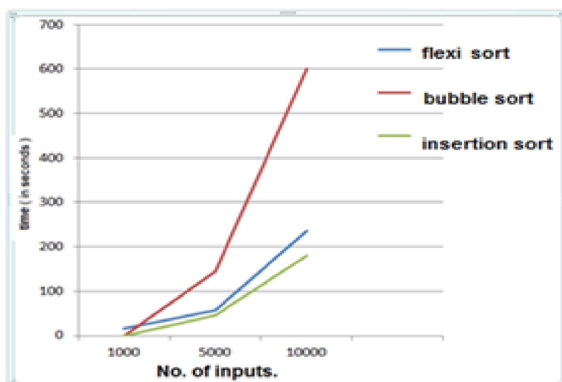
The algorithm holds good for any number of elements (independent of the number of elements to be sorted). This algorithm can sort a wide range of numbers both positive and negative numbers. The algorithm will be more effective when compared to the other sorts with comparative operator. Here the user is free to choose the "random constant" which makes the algorithm more interactive.



VI. COMPARISON

We compare to sorting algorithms bubble sort and insertion sort with "Flexi sort". The algorithm is basically compared by their clock time (execution time). The algorithm is provided with 500 computer generated random numbers whose range varies from 0-10000.

A. Comparison graph :



B. Complexity :

- Flexi sort :

The worst case complexity is O (n^2).Number of comparisons is vital in complexity calculation
– Worst Case can be calculated as (n-1) + (n-2)…. +1-> O (n^2).
– Best Case can be O (n).
Number of swapping operations performed
-- Worst Case O (n^2).
-- Best case O (n).

- Bubble sort :

-- Worst case: O (n^2).
-- Best case: O (n).

- Insertion sort :

Best case: O (n).
Worst case: O (n^2).
The clock time of "Flexi sort" is higher when compared to the other sorts, this is because of the additional feature of this algorithm as mentioned earlier (user interaction). A random constant of user choice is made a base mark in comparing the elements, this makes the execution time longer.

VII. FUTURE ENHANCEMENT

The algorithms execution time is little higher when compared to the other sorting algorithm. This is mainly because of the size of the constant (ran) entered by the user. If the number of digits in the constant (ran) is higher, then the execution time will also be higher. This is because the compiler uses the value of (ran) in all its iterations.
To make it more simple and effective the algorithm has to be enhanced so that the execution time is considerably reduced. The most possible way is by controlling the number of digits in the constant (ran).
To achieve this mathematical principle can be applied. The number of digits in any given number can be reduced by two operations the first one is the division operation and the second one is the modulo division operation. Among these two operations modulo division can be performed as it reduces the number of digits to a great extent. This quote can be proved by the following case
-Assume the user entered random constant (ran) is "500"
-Assume that the divisor is 2.
-case 1:
-Performing modulo division we get
- 500%2=0.
- Thus the user entered constant (ran) value is now being reduced to "0".
-case 2 :
-Performing division operation we get
- 500/2=250.
-The (ran) constant value is reduced to 250. Which is higher when compared to the previous operation.
The above case proves our claim and we would always use "2" as the divisor because the result would be either "0" or "1" that will help us in reducing the execution time of the algorithm.

VIII. CONCLUSION

This algorithm uses a basic mathematical principle to sort and hence the manual working out of this algorithm will be simple and less time consuming. This algorithm also teaches you about the hidden mathematical principle behind the "comparative operator" generally used in all algorithms and operations of our day today life. This algorithm will allow the user to experience a different working experience since it deals with both computational and mathematical science to a great extent.

REFERENCES

1. Sartaj Sahni, Data structures, Algorithms and Applications in Java, McG-Hill.
2. Robert Sedgewick, Algorithm in C++, Third edition, Addison Wesley.
3. Herbert Schildt, The complete Reference Java 2, Fifth edition, McG-Hill.