Cognitive Sorting

Pankaj Kumar G
Dept. of Computer Science & Engineering
FISAT,Angamaly
Cochin,Kerala.

Prasad J C
Dept. of Computer Science & Engineering
FISAT, Angamaly
Cochin, Kerala.

ABSTRACT
Sorting or ordering a list of items is one of the tasks that occur frequently in majority of computer programs. In our work we incorporated certain aspects of human cognition into sorting algorithms that will improve their performance. This paper presents two new variants of Selection sort. The first algorithm has a time complexity of O(1) in the best case. The second algorithm outperforms Selection sort and its variants when sorting a list of items that has a large number of duplicates present in it.

Keywords
Algorithms, Bingo Sort, Cognition, Selection Sort, Time Complexity.

1. INTRODUCTION
Sorting is any process that involves arranging of items into different sets or in some specific order. Sorting of items is not a problem that is specific to computer science. Sorting had always been an integral part in human life from an early period. The sorting process that occurs in other fields involves sorting of sediments by running water, extraction of gold from ore etc. However the underlying principles involved in performing the sorting are not always essentially the same.

Sorting algorithms have their importance in undergraduate classes to complex engineering problems. For many of the applications the performance is mainly determined by the underlying sorting algorithms. The earliest works on sorting algorithms can be traced back from 1956 [1][2] and still researchers are trying to optimize and develop new algorithms that can outperform the traditional ones[3][4].

A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine [5]. Humans have the remarkable ability to dynamically adapt and improve methods used to solve problems depending on the situation [6]. If we do not impose strict constraints and ask a human to perform some task, he will try to perform the task efficiently by adapting to the situations. The improvements presented in this paper are inspired by human cognition.

2. BACKGROUND
The major sorting algorithms used today are bubble sort, insertion sort, selection sort, quick sort, merge sort etc [7][8].

Bubble sort is a simple sorting algorithm that starts by comparing two elements in the start and performs swapping if the condition is evaluated as false. It then proceeds by comparing two adjacent elements till the end and the process is repeated till no swapping occurs.

Selection sort is a simple sorting algorithm that has advantages over other algorithms in certain situations. Selection sort proceeds by finding the minimum element in the list and moves the element to the correct position by performing a swap operation.

Insertion sort is efficient in sorting small lists or list that has a high degree of sorted data present in them. It proceeds by selecting each item in the list and inserting them into their correct position.

Merge sort creates a number of sub lists of the given input list having one or two elements by a process of continuous partitioning. It then proceeds by sorting sub-lists by comparison operation. It then takes advantage of these sorted sub lists by merging them to form the final sorted array.

Quick sort uses the divide and conquer approach. It first selects a pivot element and places all elements having value lesser than the pivot element to one side and all elements having value larger than it to the other side. Then it recursively performs the operation on the newly formed sub-lists till all the elements are sorted.

There are a number of sorting algorithms and their variants that is not covered under the scope of this paper [8]. Major variants of sorting algorithms include cocktail sort [10], bingo sort, shell sort [11] and enhanced shell sort [12].

Cocktail sort is a variant of bubble sort that performs sorting in both directions. The advantage of this bi-directional variant is that it effectively deals with the turtles present in the input.

Bingo sort is a variant of selection sort that has two passes. In the first pass it finds the minimum element. In the next pass it moves all elements equivalent to the minimum element and moves them to their correct position. Bingo sort is very efficient for sorting inputs containing large number of duplicate elements.

3. COGNITIVE SELECTION SORT

3.1 Inspiration from Human Cognition
Humans when asked to perform sorting using the concept of selection sort on a list of numbers using a paper and pencil can outperform a computer that performs algorithmic implementation of selection sort, if computers and humans take same time to perform same operation. Our superiority here arises due to our ability to make use of already known facts. Consider an example in which a person is performing sorting on an array in which the element a[i+p] is the smallest element and element a[i] is the smallest element occurring before a[i+p] where i and p are positive integers. After moving a[i+p] to its correct position we set the smallest element as a[i] and proceed to look for the smallest element from the position i+p because we already know that the smallest element between 0 and i+p is a[i].Hence there is no need to check for the minimum element in that range.

3.2 Procedure

- Insert the elements into the array and call the function cognitive selection sort with the array and size as parameters.
- Find the first increasing sequence in the array and store the positions of elements in the sequence in a new array b.
- Move the last element in the sequence to its correct place by swapping, if the length of new array is same as the number of elements algorithm have to sort at that point exit from the function.
- Find the new increasing sequence and modify the new array and return to the previous step till the program exits.

3.3 Pseudocode

function cognitive selection sort (array, size)

```
1   var b(size),  i, j, y, temp
2   y:=-1
3   i:=0
4   while i<size do
5       b(++y):=i
6       b(y):=i+1;
7       while y>=0 do
8           j:=b(y+1)
9           while j<size do
10              if a(j)>=a(b(y)) then
11                  b(++y):=j
12              end if
13          end while
14          if x-1 == y
15              return array
16          end if
17          temp:=a(x-1)
18          a(x-1):=a(b(y))
19          a(b(y)):=temp
20          y:=y-1
21          x:=x-1
22      end while
23  end while
```

3.4 Analysis

The algorithm works by finding the first increasing sequence in the input and stores the positions of elements in the sequence in a new array. The first element in the sequence will always be the first element in the input array. It then checks whether the number of elements in the new array is same as the number of elements to be sorted. If the condition is evaluated to true the algorithm returns from the function otherwise it moves the last element in the sequence to its correct place by performing a swap and removes the last element from the new array. After each swap the number of elements to be sorted gets reduced by one. It tries to find the first sequence in the modified array and continues till the entire input is sorted. The algorithm can also be called as first sequence sorting since the sorting algorithm works by modifying the first sequence present in the input. To find the first sequence in the modified array we need only to extend the previous sequence with last element removed. Calculating the first increasing sequence for the modified array from the beginning will waste CPU cycles. Since the algorithm also makes uses information of already known minimal elements and avoids unnecessary calculations it can also be called as selection sort using dynamic programming.

The algorithm performs no swapping operation if the given input is already sorted. If the given input is in descending order the algorithm performs n/2 swapping operations. In general depending on the input the number of swaps performed can vary from 0 to n-1.The reason for performing no swap operation when the given input is already sorted is because the length of the first sequence will be same as number of elements to be sorted. The reason for taking only n/2 swap operations in case of an input in descending order is because for each swap the elements get ordered in the begining. Consider the given sequence.

10	9	8	7	6	5	4	3	2	1

After the first swap the sequence becomes as shown below.

1	9	8	7	6	5	4	3	2	10

After the second iteration the sequence becomes as shown below.

1	2	8	7	6	5	4	3	9	10

The worst case for the algorithm will be an algorithm similar to the one shown below.

10	1	2	3	4	5	6	7	8	9

In a sequence similar to the above one the algorithm has to perform n-1 swaps. If the elements in the array is already sorted from second to last position and if the first element have the largest value it can nullify any advantage present in the input data.

4. COGNITIVE BINGO SORT

4.1 Inspiration from Human Cognition

Humans when asked to perform sorting using the concept of bingo sort on a list of numbers written in paper using a pencil and eraser can outperform a computer performing an algorithmic implementation like the previous case. Humans are capable of performing multiple tracking operations simultaneously. We will try to find the maximum and minimum element in the input at the same time. In the next step we will try to move all the occurrence of maximum and minimum element to their correct places. While performing this we will also try to find the elements with minimum and maximum values for the usage in next step. Consider an example in which a person is performing sorting on an array. In the first step he finds the maximum and minimum element let the elements be min1 and max1.On the next pass he will try to move all the elements with value min1 and max1 to their current position. While performing this he can also search for min2 and max2 where min2 and max2 are the minimum and maximum elements in the array respectively if min1 and max1 are removed from the array.

4.2 Procedure

- Insert the elements into the array and call the function cognitive bingo sort with passing the array and size as its parameters.
- Find the smallest and largest element.
- Move all the elements with minimum and maximum value to their correct place and while performing this find the next smallest and largest element.
- Repeat the above step till the array is fully sorted.

4.3 Pseudocode

function cognitive bingo sort (array,size)

```
1    var i,  j,  min, max, cmin, cmax, temp
2    cmin:=a(0)
3    cmax:=a(0)
4    i:=1
5    while i<size do
6        if a(i)>cmax then
7            cmax:=a(i)
8        end if
9        if a(i)<cmin then
10           cmin:=a(i)
11       end if
12   end while
13   i:=0
14   while i<size do
15       min:=cmin
16       max:=cmax
17       cmin:=max
18       max:=cmin
19       j:=i
20       while j<size do
21           if a(j)==max then
22               temp:=a(x-1)
23               a(x-1):=a(j)
24               a(j):=temp
25               size:=size-1
26           else if a(j)==min then
27               temp:=a(i)
28               a(i):=a(j)
29               a(j):=temp
30               i:=i+1
31               j:=j+1
32           else if a(j)>cmax then
33               cmax:=a(j)
34               j:=j+1
35           else if a(j)<cmin then
36               cmin:=a(j)
37               j:=j+1
38           end if
39       end while
40   end while
41   return array
```

4.4 Analysis

The algorithm works by finding the minimum element and maximum element before entering into the main part. It then reads each element from the beginning of the array and checks whether it is equivalent to the minimum element or maximum element. If it is equivalent to the minimum element or maximum element the element is moved to their correct position via swapping and checking is continued. The algorithm do not consider the elements that have been moved to their correct positions in any future operations. If the element that is under consideration is not the minimum element or maximum element algorithm checks whether it is the minimum element or maximum element from the elements that are not swapped. The number of swap operations performed by the algorithm in all case remains same.

5. EXPERIMENTATION

In order to evaluate the efficiency of the proposed algorithms were implemented in C. The program was compiled using GCC 4.4 and the program was run with various set of input. Since the performance of CSS varies with the sequence present in it we considered seven inputs.

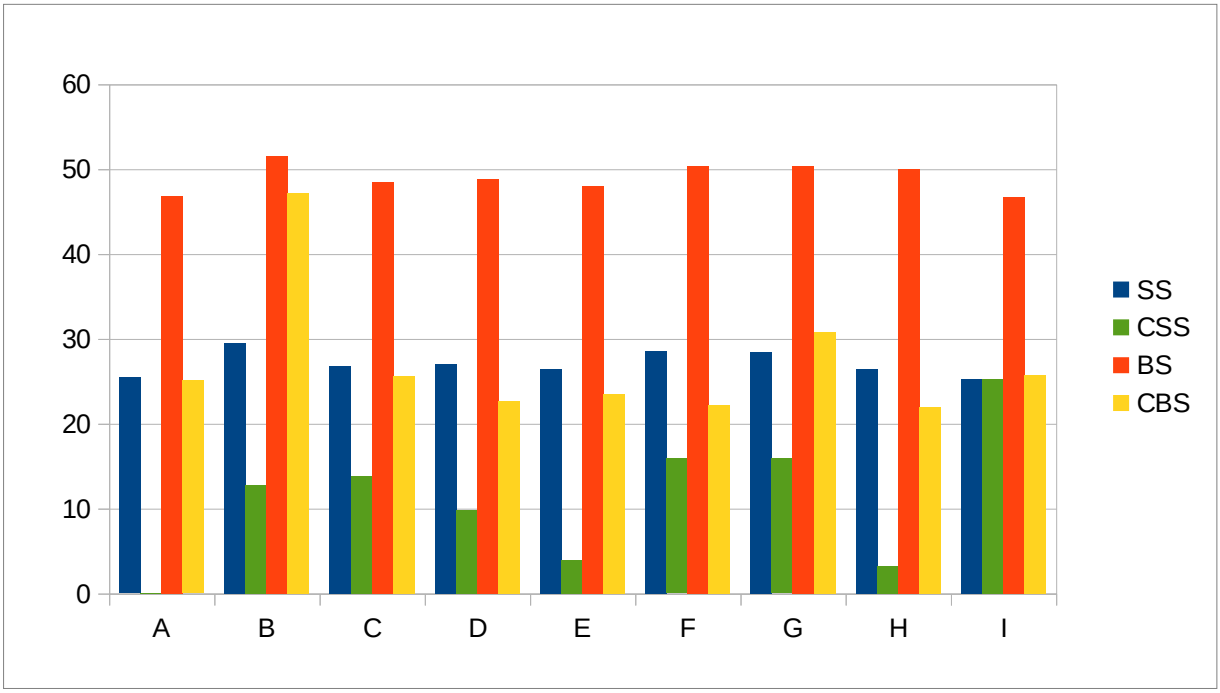

Figure 1. Results obtained on input with no duplicates

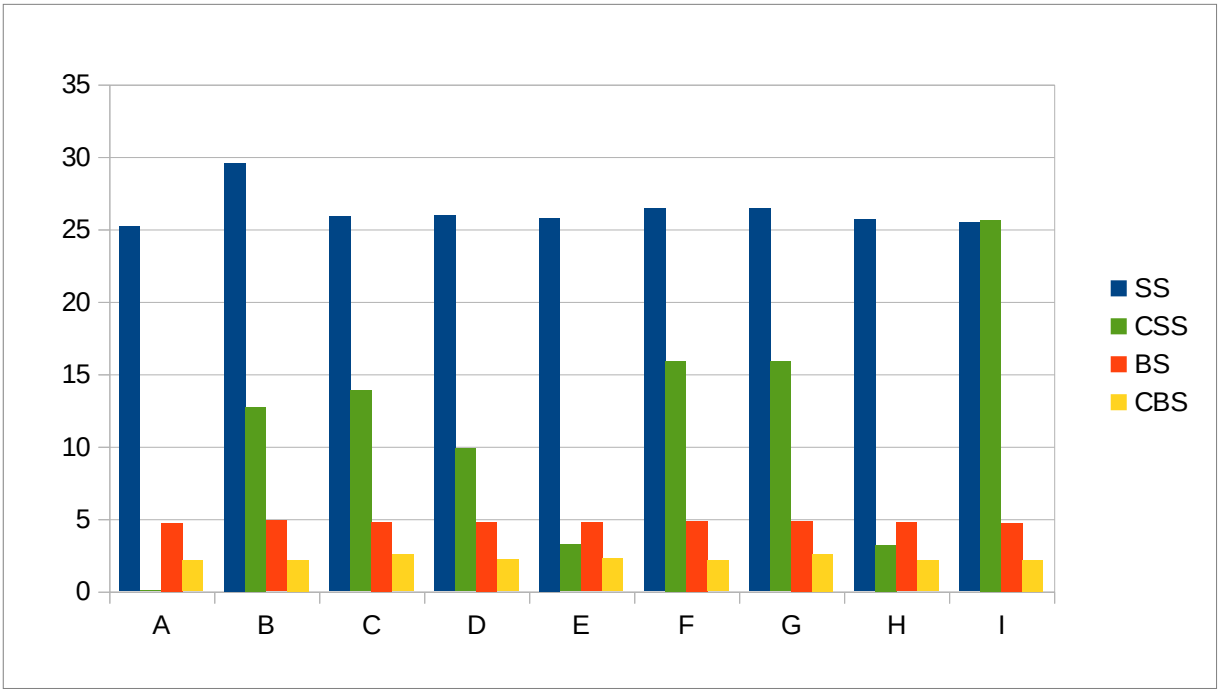

Figure 2.Result obtained on inputs with duplicates.

Table 1. Details about the input test cases.

Case	Description
A	Input is already sorted
B	Input is in descending order
C	Elements in the odd position of the input constitutes an ascending sequence and elements in the even position constitutes a descending sequence and every element in the ascending sequence is smaller than elements in other sequence
D	Elements are ordered as in case C but every element in the ascending sequence is larger than elements in other sequence
E	The first half of the input is in ascending order and the second half is in descending order and every element in the ascending sequence is smaller than elements in other sequence
F	Elements are ordered as in case E but every element in the ascending sequence is larger than elements in other sequence
G	The first half of the input is in descending order and the second half is in ascending order and every element in the ascending sequence is smaller than elements in other sequence
H	Elements are ordered as in case G but every element in the ascending sequence is larger than elements in other sequence
I	Worst case for CSS,

For testing the efficiency of CBS we considered the test cases mentioned in Table 1 with 10 percentage duplicates. The number of elements in the test cases was 100,000.

6. RESULT ANALYSIS

The results obtained after running experiments are shown in the graphs. The X-axis of the graph shows the running time taken by the various implementations. The Y-axis contains various algorithms that were considered.

From the graph we can see that in both cases CSS offers better performance when the input is completely sorted. The performance of CSS degrades to that of normal selection sort only in the worst case scenarios. The reason for the better performance of CSS is because it can take advantages of the ordering present in the inputs.

Selection sort offers a consistent performance in all the cases. Selection sort cannot take any advantage of the patterns inherent in the inputs. Selection sort shows the worst performance in all the cases. Bingo sort shows the worst performance in case where input does not contain any duplicates. It took almost twice the time taken by the selection sort in its worst case. But in case of the inputs with larger number of duplicate elements bingo sort showed better performance than CSS and SS.

Performance of CBS was similar to that of selection sort in the first case where input had no duplicate elements. In the second case CBS outperformed all other algorithms. The time taken by CBS in the second case was nearly half of the time taken by bingo sort.

7. CONCLUSION

In this paper two new sorting algorithms were presented. CSS has a time complexity of O(1) in the best case. In the worst and average case its time complexity is same as selection sort. CSS outperforms selection sort in majority of cases.

CBS offers better performance by performing multiple operations in a single iteration. It is the fastest among the four algorithms for sorting a list containing high concentration of duplicate elements.

These proposed algorithms were implemented and experiments were conducted to measure their efficiency in varying environments.

8. ACKNOWLEDGMENTS

We would like to thank Computer Society of India for financially supporting this research project under minor research grant.

9. REFERENCES

[1] Demuth, H. 1956. Electronic Data Sorting. PhD thesis, Stanford University.

[2] Astrachanm O.,2003. Bubble Sort: An Archaeological Algorithmic Analysis, Duk University.

[3] Bender, M. A., Farach-Colton, M., and Mosteiro M. 2006. Insertion Sort is O(n log n). Theory of Computing Systems Volume 39, Number 3.

[4] Jehad Alnihoud and Rami Mansi. An Enhancement of Major Sorting Algorithms. The International Arab Journal of Information Technology, Vol. 7, No. 1, January 2010.

[5] Turing, A. M., 1948, Intelligent machinery.

[6] Mańdziuk, J ,"Towards Cognitively Plausible Game Playing Systems" , IEEE Computational Intelligence Magazine vol.6 no.4 pp. 38-51, 2011.

[7] Donald Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89685-0.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7.

[9] Wikipedia -The free encyclopedia [online] Available: http://en.wikipedia.org/wiki/Sorting_ algorithm.

[10] Nyhoff L., An Introduction to Data Structures, Nyhoff Publishers, Amsterdam, 2005.

[11] Shahzad B. and Afzal M., "Enhanced Shell Sorting Algorithm," Computer Journal of Enformatika, vol. 21, no. 6, pp. 66-70, 2007

[12] Shell D., "A High Speed Sorting Procedure,"Computer Journal of Communications of theACM, vol. 2, no. 7, pp. 30-32, 1959.