AN EFFICIENT SORTING ALGORITHM: INCRECOMPARISION SORT

Ms. Geeta¹, Ms. AnubhootiPapola²

¹M.Tech, 2nd year, Dept. Computer Science and Engineering, Uttarakhand Technical University, India
²Assistant Professor, Dept. Computer Science and Engineering, Uttarakhand Technical University, India

ABSTRACT

Data Structure is a logical organization of data which subsume of a number of operations, sorting is one of them. Sorting facilitate some other operation such as searching, merging and normalization.[6] There are many sorting algorithm that has being used in practical life as well as in computation. A sorting algorithm includes comparison, swapping, and assignment operations. There are numerous of the sorting algorithm like Selection Sort, Double Ended Selection Sort, Max-Min Sort which takes $O(n^2)$ time in its best case and perform unnecessary number of comparisons.[2,3,4,9] In this paper we proposed an algorithm Increcomparision Sort (ICS) which does not execute unnecessary number of comparisons. This paper shows enhancement of those algorithm which has complexity $n^2$ in their best case.

Keywords: Algorithm, Max-Min Sort, Double-Ended Selection Sort, Complexity, Increcomparision Sort.

INTRODUCTION

An algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function In this computational world, sorting plays an important role. Sorting is an arrangement of data items either in

1. Statistical Order:- Order the data based on their numerical values. This is performed in the form-
   a) Ascending Order:- Smallest item is in first place and largest item is in last place. Or we can say smallest item sorted first.
   b) Descending Order:- Largest item is in first place and smallest item is in last place. Or we can say largest item sorted first.
2. Lexicographical Order:- Alphabetical value like addressee key

There are many well known sorting algorithms for unsorted list, such as selection sort, Quick sort, Insertion sort, Merge sort, Bubble sort and so on. There are many factors to evaluate the performance of the sorting algorithms which are given below. [1]

1. Computational complexity (worst, average and best behavior) (n).
2. Usage of memory and other computer resources.
3. Stability
4. Number of swaps (for in-place algorithms).
5. Number of comparisons

The common sorting algorithms can be divided into two classes by the difficulty of their algorithms. There is a direct correlation between the complexity of an algorithm and its relative effectiveness. Complexity of an algorithm is calculated by asymptotic notations and in three cases. There are three types of asymptotic notations-

1. **O** notation: $O(g(n)) = \{f(n) : \exists$ positive constants $c$ and $n_0$, such that $\forall n \geq n_0$, we have $0 \leq f(n) \leq cg(n) \}$

2. **Θ** notation: $\Theta(g(n)) = \{f(n) : \exists$ positive constants $c_1$, $c_2$, and $n_0$, such that $\forall n \geq n_0$, We have $0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \}$

3. **Ω**-notation: $\Omega(g(n)) = \{f(n) : \exists$ positive constants $c$ and $n_0$, such that $\forall n \geq n_0$, we have $0 \leq cg(n) \leq f(n) \}$

The three cases are-

1. Best case:- minimum time taken by an algorithm.
2. Average case:- average time or mean time taken by an algorithm.
3. Worst Case:- Maximum time taken by an algorithm.

Double ended selection sort and max-min sort perform unnecessary comparisons and swaps in case of sorted array. To improve this we proposed an algorithm in which there is no need of unnecessary swapped which improve the complexity of algorithm.

**Increcomparision Sort:** Sorting is one of the profound research area. For any sorting algorithm its minimum complexity shows its good performance. Increcomparision Sort does not perform nonessential number of comparisons in its best case. Increcomparision Sort comprises the steps-

1. i=0;
2. while(i<n-1)
3. if(a[i]<=a[i+1])
4. i=i+1;
5. else count++;
6. for j=n-1 to 0
7. if(a[j]<a[j-1])
8. exchange a[j] to a[j-1];
9. else
10. if(i=j) or (count=j)
11. break;
12. i++;

In this algorithm first we initialize i as 0. n is number of element in given array.
Working of Increcomparision Sort:- In this sorting algorithm it is based on value of array whether simply increase the index value or swap the array value. If the previous array value less than next one, we simply increase the index value. Otherwise start swapping from backward.

Suppose we have an array A which have 8 elements.

```
1 2 3 8 7 6 4 5
```

After 1\textsuperscript{st} iteration

```
1 2 3 8 7 6 4 5
```

After 2\textsuperscript{nd} iteration

```
1 2 3 8 7 6 4 5
```

After 3\textsuperscript{rd} iteration

```
1 2 3 8 7 6 4 5
```

After 4\textsuperscript{th} iteration

```
1 2 3 4 8 7 6 5
```

After 5\textsuperscript{th} iteration

```
1 2 3 4 5 8 7 6
```

After 6\textsuperscript{th} iteration

```
1 2 3 4 5 6 8 7
```

After 7\textsuperscript{th} iteration

```
1 2 3 4 5 6 8 7
```

The sorted array is

```
1 2 3 4 5 6 8 7
```

Fig 1: Working of Increcomparision Sort
**Complexity of Increcomparision Sort:** Complexity of any algorithm is represents as - Time Complexity, which calculates time taken by any algorithm where as Sapce complexity defines, how much space or memory required for any algorithm.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=0</td>
<td>C1</td>
<td>1</td>
</tr>
<tr>
<td>while(i&lt;n-1)</td>
<td>C2</td>
<td>n</td>
</tr>
<tr>
<td>if(a[i]&lt;a[i+1])</td>
<td>C3</td>
<td>n-1</td>
</tr>
<tr>
<td>i=i+1</td>
<td>C4</td>
<td>(n-1).ti</td>
</tr>
<tr>
<td>else count++</td>
<td>C5</td>
<td>te</td>
</tr>
<tr>
<td>for j=n-1 to 0</td>
<td>C6</td>
<td>te.n</td>
</tr>
<tr>
<td>if(a[j]&lt;a[j-1])</td>
<td>C7</td>
<td>te.(n-1)</td>
</tr>
<tr>
<td>exchange a[j] to a[j-1]</td>
<td>C8</td>
<td>te.(n-1).tj</td>
</tr>
<tr>
<td>Else if(count=j) or(i=j)</td>
<td>C9</td>
<td>tee</td>
</tr>
<tr>
<td>break;</td>
<td>C10</td>
<td>tee.tci</td>
</tr>
<tr>
<td>i++</td>
<td>C11</td>
<td>te</td>
</tr>
</tbody>
</table>

In above table cost defines a constant amount of time is required to execute each line of our algorithm. One line may take a different amount of time than another line but we shall assume that each execution of the i-th line takes time c_i, where c_i is a constant. Time defines how much time each line executes in the given algorithm.[16] We can say complexity of any algorithm is sum of multiplication of cost with time of each line.

**Complexity of Algorithm**

\[
T(n) = C1*1+C2*n+C3*(n-1)+C4*(n-1).ti+ C5*te+C6* te.n+C7* te.(n-1)+C8*te.(n-1).tj+ C9* tee+C10* tee.tci+C11* te \
\]

Where

\[
t_e=(n-1)-(n-1). t_i \\
t_{ee}= t_e.(n-1)-t_e.(n-1).t_j \\
t_i=1 if condition is true otherwise 0. \\
\]

Similarly,

\[
t_j,t_{ci}=1 if condition is true otherwise 0. \\
\]

**In Best case:-**

\[
t_e=0 and ti=1 so \\
T(n)=C1.1+C2.n+C3.(n-1)+ C4.(n-1)+ C5.0+ C6.0+ C7.0+ C8.0+ C9.0+ C10.0+ C11.0 \\
T(n)=C1 + nC2 + nC3 +C3 + nC4 - C4 \\
T(n)= n(C2 + C3+ C4 ) – (C3+C4) + C1 \
\]
So complexity in best case  
T(n)=O(n) 

In Worst case:-  
t_i=0 and t_e=n-1

T(n)=C1*1+C2*n+C3*n-1+C4*(n-1)0+ C5*(n-1)+ C6*(n-1)n +C7* (n-1)(n-1) +C8. (n-1).(n-1),t_j+C9*(n-1).(n-1),(n-1).t_i+C10.((n-1)(n-1)-(n-1)(n-1).t_j)+C11.(n-1)

For the worst case complexity of any algorithm depends on its inner most loop. In this sorting  
algorithm complexity is depends on C6*(n-1)n .  
So worst case complexity 
T(n)=O(n²)

Comparison With other algorithm:- Increcomparision sort takes minimum time in its best case. It  
is better than selection sort, merge sort, heap sort etc. in its best case. But in its worst case it also  
produces O(n²) time like some other algorithms insertion sort, cocktail sort.[7,8]

Table 2: Comparison with different sorting algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Best Case</th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Sort</td>
<td>O(n²)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Bubble Sort</td>
<td>O(n²)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>O(n)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>O(nlogn)</td>
<td>O(nlogn)</td>
<td>O(nlogn)</td>
</tr>
<tr>
<td>Heap Sort</td>
<td>O(nlogn)</td>
<td>O(nlogn)</td>
<td>O(nlogn)</td>
</tr>
<tr>
<td>Quick Sort</td>
<td>O(nlogn)</td>
<td>O(nlogn)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Cocktail Sort</td>
<td>O(n)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Max Min Sort</td>
<td>O(n²)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Double Ended Selection Sort</td>
<td>O(n²)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Increcomparision Sort</td>
<td>O(n)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
</tbody>
</table>

CONCLUSION AND FUTURE SCOPE

In computer science, a sorting is an efficient way which is used to puts elements of a list in a  
certain order or arranges a collection of items into a particular order. Sorting data has been developed  
to arrange the array values in various ways for a database. In this paper we introduce the  
Increcomparision sort, how it is work, what its complexity and how better it is than other algorithms.  
Increcomparision Sort is an algorithm which is good for small amount of data. It produces faster  
result on that data and it is easy to understand for everyone. It is a straightforward approach and  has  
a lot of future scope. Sorting algorithm based on divide and conquers approach produces very fast  
result. Most of these algorithm takes O(nlogn) time. So there is a chance we can improve  
Increcomparision Sort. We can employ this approach in Quick Sort to improve its worst case  
complexity in case when array is already sorted. This approach is problem oriented so it is a  
possibility to make it global oriented. This algorithm is implemented using static data structure, so it  
may be prospect we will implement using dynamic data structure.
At last we want to say there is a vast scope of sorting algorithm in future and to find an optimal sorting algorithm in future.

REFERENCES


ABOUT AUTHOR

Ms. Geeta student of M.Tech in Uttarakhand Technical University. She did her B.tech. from Uttar Pradesh Technical University in 2007. She did M.Techs from Karnataka State Open University in 2011. She did teaching in different institution. Her area of interest is data structure, design and analysis of algorithms and image processing.

Ms. Anubhooti Papola presently working as an assistant professor in Uttarakhand Technical University. Before that she works in different institutions. She did her B.tech. from H.N.B Garhwal University in 2009 and M.Tech from Graphic Era University, Dehradun in 2012. Her area of interest is data structure and image processing.