Performance Analysis of Enhanced Selection Sort Algorithm Based on Even-Odd Elements for Positive Integers

Sarvjeet Singh
(M.Tech. Scholar)
Dept. of Computer Science and Engineering
Lovely Professional University
Phagwara, Punjab, India

Surmeet Kaur
(Assistant Professor)
Dept. of Computer Science and Engineering
Lovely Professional University
Phagwara, Punjab, India

Abstract—One of the most problems for the computer to mapping the data into databases. Nowadays many algorithms are introduced for the sorting. Generally Sorting is the operation that arranged the logically data into the database. There are the application of the sorting is searching, used a different method like linear, based on indexed sequentially to make access fast and efficiently. In this research paper introduced an enhancement of performance of the selection sort using the classification of data on the basis of even and odd elements. We conclude the result of the existing algorithm with the proposed sorting algorithm with the help of mathematics and practically implementation of the algorithms. Time complexity of enhanced selection sort in the best case is big \(O(n \log n)\).

Keywords—Big O Notation, CPU, RAM, Complexity.

I. INTRODUCTION
Algorithm is the vital part of the operational research methodology. Basically algorithms involved the step by step procedures to solve the problem and it is terminated after the when statement either true or false vice-versa. So there are the two method to analysis of the sort algorithms based on the mathematical analysis and second is the empirical analysis[2]. There are the two factor based for the algorithms complexity:-
1. Time (how much time taken to execution)
2. Space (how much space algorithm require)
There are several factor which affected the algorithms complexity:-
1. Number of comparisons.
2. Number of swaps
3. Number of arithmetic operations.
4. Number of iterations.

II. EXISTING SORTING ALGORITHMS
1) Bubble Sort Algorithm:- This is simplest sorting algorithm, in which elements scan from starting index of an array and the compare with adjacent element if its greater than it swap if it’s not than second element compare with third and so on till array is not equal to null, and this process repeated until list is being sorted. The time complexity is \(O(n)\) in best case and \(O(n^2)\) in worst case and as well as space complexity is depends upon the number of inputs. Bubble Sort Algorithm is as follows[8]:-

```plaintext
 BubbleSortAlgorithm( A , n)
 1. for i ← 0 to n – 1 do
 2.       for j ← 0 to n – i
 5.             end for of step 2
 6.         end for of step 1
```

Figure 1.1 Bubble Sorting Algorithm

2) Selection Sorting Algorithm:- In this method of sorting, selection method is being used, basic formula to conclude the algorithm is that smallest element is selected from the array and placed in the proper location of the array in
particular pass, this is repeated until list is being sorted. The time complexity of the algorithm is $O(n^2)$ in best and worst case of the input data size[2].

![SelectionSortAlgorithm(A,n)
1. i ← 0
2. while i < n do
3. j ← i+1
4. while j < n do
6. Swap(A[i],A[j])
7. j ← j+1
8. End if of step 5
8. End While of step 4
9. i = i+1
10 end-while

Figure 1.2 Selection Sorting Algorithm

III. PROPOSED ENHANCED SELECTION SORTING TECHNIQUE

In the proposed sort algorithms we can take the input from the user equal number of the even and odd elements and after that we can take the temporary array of the four size and after the selection of the elements there are we can extract the even values as well as odd values from the inputted array. After the pick values from it store in temporary array and compare to each other and fixes into a fixed sized array of n number, where n equal to number of inputs.

Algorithms:-
Step 1.-Input the array.
Step 2.-Assign proper index to the array.
Step 3.-Assign temporary array of fixed size of four elements.
Step 4.- Select the minimum even and minimum odd value from the array and store into the temp array of fixed size four.
Step 5.- go to step 4 till temp array full.
Step 6.- Compare the values in temp array and store in the different array till temp array n/2!=0.
Step 7.- Go to step 4.
Step 8.-End.

Pseudocode of fetch values from the algorithms:-
1. For Fetch Even Values From List:-
   for(int k=0;k<evenl;k++)
   {
     for(int p=k+1;p<evenl;p++)
     {
       if(even[p]<even[k])
       {
         int sw=even[k];
         even[k]=even[p];
         even[p]=sw;
       }
     }
   }
2. For Fetch Odd Values From List:-
   for(int k1=0;k1<oddl;k1++)
   {
     for(int p1=k1+1;p1<oddl;p1++)
     {
       if(odd[p1]<odd[k1])
       {
         int sw1=odd[k1];
         odd[k1]=odd[p1];
         odd[p1]=sw1;
       }
     }
   }
After fetch the values from list then we compare the values which is temporary stored in the values in the four size fixed array and then compare with all an freeze it after the comparison of values from fixed size array, there are in third separated list of an elements which which used to store the values comes from after comparisons.

3. Comparisons of fetched Even and Odd Values:

```java
for(int jl=0;jl<length;jl++)
{
    if((even[e2]<odd[o2] && e2!=e1)||o2>=o1)
    {
        final1[jl]=even[e2];
        e2++;
        continue;
    }
    if((even[e2]>odd[o2] && o2!=o1) || e2>=e1)
    {
        final1[jl]=odd[o2];
        o2++;
        continue
    }
}
```

Example with Proposed Algorithm:

<table>
<thead>
<tr>
<th>14</th>
<th>8</th>
<th>15</th>
<th>21</th>
<th>18</th>
<th>7</th>
<th>3</th>
<th>12</th>
</tr>
</thead>
</table>

After first Pass :

<table>
<thead>
<tr>
<th>14</th>
<th>8</th>
<th>15</th>
<th>21</th>
<th>18</th>
<th>7</th>
<th>3</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>7</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After Second Pass:

<table>
<thead>
<tr>
<th>14</th>
<th>8</th>
<th>15</th>
<th>21</th>
<th>18</th>
<th>7</th>
<th>3</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12</td>
<td>14</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After Third Pass:

<table>
<thead>
<tr>
<th>14</th>
<th>8</th>
<th>15</th>
<th>21</th>
<th>18</th>
<th>7</th>
<th>3</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After Fourth Pass:

| 14 | 8 | 15 | 21 | 18 | 7 | 3 | 12 |
Table 1.1 Comparisons on the basis of complexity and other factors

<table>
<thead>
<tr>
<th>Time Complexity:</th>
<th>Bubble Sort Algorithm</th>
<th>Selection Sort Algorithm</th>
<th>Insertion Sort Algorithm</th>
<th>Merge Sort Algorithm</th>
<th>Enhanced Selection Sort Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Case</td>
<td>O(n)</td>
<td>O(n²)</td>
<td>O(n)</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>Average Case</td>
<td>O(n)</td>
<td>O(n²)</td>
<td>O(n)</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>Worst Case</td>
<td>O(n)</td>
<td>O(n²)</td>
<td>O(n)</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
</tr>
</tbody>
</table>

| Passes used in above Example | 7 | 6 | 6 | 5 | 7 |

| Space Complexity | O(1) | O(1) | O(1) | O(n) | O(1) |

<table>
<thead>
<tr>
<th>Method used</th>
<th>Exchange</th>
<th>Selection</th>
<th>Incremental</th>
<th>Merging</th>
<th>Selection</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>In Place Algorithm</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Stable Algorithm</th>
<th>Yes</th>
<th>No</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Type of Memory used</th>
<th>Internal Memory</th>
<th>Internal Memory</th>
<th>Internal Memory</th>
<th>Internal and External Memory</th>
<th>Internal Memory</th>
</tr>
</thead>
</table>
Practical Results:
In the comparisons of the other algorithms execution time is calculated. We conclude the result from the selection, bubble, insertion, freeze sorting algorithm. The algorithm efficiency is measured on the basis of the CPU time based on the time taken by the running process at background area of the n number of inputs data. The algorithm was obtained results from the C Language with the help of in built function Clock(). It runs on Sony Vaio E-Series laptop with the following specification:

- Intel(R) Core™ i5-2450M CPU at 2.50GHz with 4 GB RAM.

In first table result will shows of the Clock() function[2].

1) Practical Results:
In the comparisons of the other algorithms execution time is calculated. We conclude the result from the selection, bubble, insertion, freeze sorting algorithm. The algorithm efficiency is measured on the basis of the CPU time based on the time taken by the running process at background area of the n number of inputs data. The algorithm was obtained results from the C Language with the help of in built function Clock(). It runs on Sony Vaio E-Series laptop with the following specification:

- Intel(R) Core™ i5-2450M CPU at 2.50GHz with 4 GB RAM. In first table result will shows of the Clock() function[2].
IV. CONCLUSIONS

In this proposed algorithms, There are O (n log n) time complexity. The number of element is select by n/2 inputs after comparisons in particular pass is occur, and it is lesser than the Selection sort algorithm. It is called as stable and in place algorithm also because it selects the element from left to right in array. It needs only O (1) space complexity. So the performance analysis is based upon the two different scenario one is mathematically and second practical implementation. In different term of the sort it support the stable sorting because selection of the elements is directly linear form.

REFERENCES

[16] Sareen Pankaj, “Comparison of Sorting Algorithms (On the Basis of Average Case)”, March 2013 IJRCSSSE.