A New Approach To Sorting: Min-Max Sorting Algorithm

Aayush Agarwal  M.Tech, CDAC Noida
Vikas Pardesi  M.Tech, DTU, Delhi
Namita Agarwal  B.Tech, BITS, Sonepat

Abstract

Many algorithms are available for sorting the unordered elements. Most important of them are Bubble sort, Heap sort, Insertion sort and Quick sort. This paper presents the new algorithm for sorting the elements which is based on minimum and maximum elements of the array which results in placing the elements at appropriate position. This will reduce the number of passes in which the sorting takes place. We will examine this sorting technique and compare with the other available sorting algorithms in terms of complexity, memory and other factors.

2. PROPOSED ALGORITHM

The Min-Max Sorting Algorithm works on the minimum and maximum element of the array. It finds the minimum and maximum element from the array and set on the first and last position of the array. Then the array index increment from the first position and decrement from the last position to get the new array. From this new array, it again finds the minimum and maximum element and set on their respective positions. In this way, this sorting algorithm sorts the elements of the array. Here we provide the Pseudo-Code for this sorting algorithm.

Pseudo-Code for Min-Max Sorting

1. Set p:=0,q:=n-1    //n is the number of elements
2. while p<q do:
   Repeat steps 3 to 6
3. Minimum(a,p,q)     //pass array,p,q to find minimum element of current array
4. Maximum(a,p,q)     //pass array,p,q to find maximum element of current array
5. Set p:=p+1         //increment p
6. Set q:=q+1         //increment q
7. End while
8. Print sorted array //end of algorithm

Pseudo-Code for finding the minimum element from the array:

**Pseudo-Code for Minimum Function**

Minimum(int a[],int p,int q)     //Receive array,p,q
1. Set min:=a[p]       //set the first element of array to min
2. for l=p to q do:
   Repeat Steps 3 to 5
3. if a[l]<min then
4. swap(a[l],min)     //swapping is done and find the minimum element
5. Set l:=l+1;
6. End for
7. a[p]=min

Pseudo-Code for finding the maximum element from the array:

**Pseudo-Code for Maximum Function**

Maximum(int a[],int p,int q)     //Receive array,p,q
1. Set max:=a[p]       //set the first element of array to max
2. for l=p to q do:
   Repeat Steps 3 to 5
3. if a[l]>max then
4. swap(a[l],max)     //swapping is done and find the maximum element
5. Set l:=l+1;
6. End for
7. a[p]=max

This paper includes:

- Section 2 describes the proposed algorithm.
- Section 3 describes the comparative study with the other algorithms.
- Section 4 describes the conclusion.
Pseudo-Code for Maximum Function

Maximum(int a[],int p,int q) //Receive array,p,q
1. Set max:=a[q]             //set the last element of array
to min
2. for i=p to q do:
    Repeat Steps 3 to 5
3. if a[i]>max then
4. swap(a[i],max)          //swapping is done and find
the maximum element
5. Set i:=i+1;
6. End for
7. max=a[q]

We provide the source code of the Min-Max Sorting Technique:

Source Code in C

```
#include<stdio.h>
#include<conio.h>

void minimum(int a[],int p,int q)
{
    int min=a[p];
    for(int l=p;l<=q;l++)
    {
        if(a[l]<min)
        {
            int temp=min;
            min=a[l];
            a[l]=temp;
        }
    }
    a[p]=min;
}

void maximum(int a[],int p,int q)
{
    int max=a[q];
    for(int i=p;i<=q;i++)
    {
        int temp;
        if(a[i]>max)
        {
            temp=a[i];
            a[i]=a[q];
            a[q]=temp;
        }
        max=a[q];
    }
}

void main()
{
    Clrscr();
    int n,a[20];
    printf("How many elements you want to enter…?\n");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
        Scanf("%d",&a[i]);
    for(int j=0;j<n;j++)
        printf("nYour Array is :%d ",a[j]);
        printf("nSorted Array is :");
        int p=0,q=n-1;
        while(p<q)
        {
            minimum(a,p,q);
            maximum(a,p,q);
            p++;
            q--;
        }
    for(int k=0;k<n;k++)
        printf("%d ",a[k]);
    getch();
}
```

OUTPUT

How many elements you want to enter…?
6
23 43 12 54 27 5
Your Array is: 23 43 12 54 27 5
Sorted Array is: 5 12 23 27 43 54

3. COMPARATIVE STUDY WITH OTHER ALGORITHMS

In this section, we provide the comparison of our Min-Max sorting algorithm with others existing sorting algorithms in different cases with example.

A. Comparison in terms of Time Complexity with other Algorithms:

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>BEST CASE</th>
<th>AVERAGE CASE</th>
<th>WORST CASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN-MAX SORT</td>
<td>n^2</td>
<td>n^2</td>
<td>n^2</td>
</tr>
<tr>
<td>QUICKSORT</td>
<td>nlog n</td>
<td>nlog n</td>
<td>n^2</td>
</tr>
<tr>
<td>HEAP SORT</td>
<td>nlog n</td>
<td>nlog n</td>
<td>nlog n</td>
</tr>
<tr>
<td>MERGE SORT</td>
<td>nlog n</td>
<td>nlog n</td>
<td>nlog n</td>
</tr>
<tr>
<td>BUBBLE SORT</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>SELECTION SORT</td>
<td>n^2</td>
<td>n^2</td>
<td>n^2</td>
</tr>
<tr>
<td>INSERTION SORT</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>
```
B. Comparison in Number of Passes to Sort the Elements:

This comparison is shown by taking an example. In this, we have an array of 5 integers and we are differentiating sorting algorithms in terms of number of passes and proved that our algorithm takes minimum number of passes to sort an array. Here we are comparing only with two sorting i.e., Selection sort and bubble sort.

Let us take an example:
Array: 5 4 3 2 1

Min-Max Sorting:

2, 3, 5, 1, 4 (Given Array)
1, 3, 5, 2, 4 (After Minimum Function) \[\text{Pass 1}\]
1, 3, 4, 2, 5 (After Maximum Function)
1, 3, 4, 2, 5 (After Minimum Function) \[\text{Pass 2}\]
1, 2, 3, 4, 5 Resultant Sorted Array

It will take only two passes to sort the elements. In General, it takes \(\lfloor n / 2 \rfloor\) passes to sort the elements where \(n\) is the number of elements.

Bubble Sorting:

2, 3, 5, 1, 4 (Given Array)
2, 3, 5, 1, 4 \[\text{Pass 1}\]
2, 3, 1, 5, 4
2, 3, 1, 4, 5

2, 3, 1, 4, 5 (Intermediate Array after Pass 1)
2, 3, 1, 4, 5 \[\text{Pass 2}\]
2, 1, 3, 4, 5
2, 1, 3, 4, 5

2, 1, 3, 4, 5 (Intermediate Array after Pass 2)
1, 2, 3, 4, 5 \[\text{Pass 3}\]
1, 2, 3, 4, 5

2, 1, 3, 4, 5 (Intermediate Array after Pass 3)
1, 2, 3, 4, 5 \[\text{Pass 4}\]
1, 2, 3, 4, 5
1, 2, 3, 4, 5

2, 3, 5, 1, 4 (Final Sorted Array after Pass 4)

It will take four passes to sort the elements. In General, it takes \((n-1)\) passes to sort the elements where \(n\) is the number of elements. It takes just double passes to sort the elements compare to Min-Max sorting.

Selection Sort:

2, 3, 5, 1, 4 (Given Array)
1, 3, 5, 2, 4 \[\text{Pass 1}\]
1, 2, 5, 3, 4 \[\text{Pass 2}\]
1, 2, 3, 5, 4 \[\text{Pass 3}\]
1, 2, 3, 4, 5 \[\text{Pass 4}\]

It will also take \((n-1)\) passes just taking double time compare to Min-Max sorting algorithm.

C. Comparison in terms of other factors:

All the sorting algorithms have two properties on which they follow. Some sorting algorithms follow these properties but some are not. Our sorting algorithm follows these properties which are as follows:

- **Stable Sorting**: A sorting algorithm is called **stable** if it keeps elements with equal keys in the same relative order in the output as they were in the input [10].

For example, in the following input the two 4's are indistinguishable:

1, 4_a, 3, 4_b, 2

And so the output of a stable sorting algorithm must be:

1, 2, 3, 4_a, 4_b

- **Inplace Sorting**: In-place algorithm is an algorithm which transforms input using a data structure with a small, constant amount of extra storage space. The input is usually overwritten by the output as the algorithm executes. An algorithm which is not in-place is sometimes called **not-in-place** or **out-of-place** [11].
4. CONCLUSION

This paper describes the new sorting algorithm named Min-Max Sorting based on finding the minimum and maximum element of the array and sort the elements. In this way, we are able to reduce the number of passes required to sort the elements in comparison with other sorting algorithms and also able to remove the rabbit and turtle problem. In future work we can try to reduce its time complexity then it can be an efficient and effective sorting algorithm.

REFERENCES