Ash Sorting: Easy & Less Time Consuming Sorting Algorithm

Dr. Leena Bhatia

Assistant Professor, MCA Institute, S.S.Jain Subodh P.G. College, Jaipur

Dr. Bindu Jain

Assistant Professor, Maharani's College, University of Rajasthan Jaipur

Summary

Sorting is the basic building block around which many other algorithms are built. By understanding sorting, we obtain an amazing amount of power to solve other problems. Sorting is the most thoroughly studied problem in computer science. Literally dozens of different algorithms are known, most of which possess some advantage over all other algorithms in certain situations. To keep these in mind we are presenting **Ash Sorting algorithm** which is comparison based less time consuming simple algorithm.

Key Words:

Ash assorting, algorithm, linear sort

1. Introduction

Sorting is a computational building block of fundamental importance and is one of the most widely studied algorithmic problems. Many algorithms rely on the availability of efficient sorting routines as a basis for their own efficiency. Historically, it has proved that computers spend more time in sorting than doing anything else. As we know that a quarter of all the mainframe cycles are spent in sorting the data. Although it is unclear whether this remains true on smaller computers, sorting remains the most universally accepted combinatorial algorithm problem in practice. Sorting is the most thoroughly studied problem in computer science. Literally dozens of different algorithms are known, most of which possess some advantage over all other algorithms in certain situations. To keep these in mind we are presenting Ash Sorting algorithm which is comparison based less time consuming simple algorithm.

The study of sorting techniques has a long history and countless algorithmic variants have been developed [1, 5]. Many important classes of algorithms rely on sort or sortlike primitives. Database systems make extensive use of sorting operations [3]. The construction of spatial data structures that are essential in computer graphics and geographic information systems is fundamentally a sorting process. Efficient sort routines are also a useful building block in implementing algorithms like sparse matrix multiplication and parallel programming patterns like MapReduce [2, 4].

Manuscript received December 5, 2010
Manuscript revised December 20, 2010

2. Ash Sorting: Concept

Ash sorting is based on the very simple real life smoke concept that is ***when we burn the coal the smoke which is lighter, fly in the air and the heaviest ash remains at the ground***. Ash sorting is also comparison based sorting but with less number of comparison as compared to selection or linear sort.

Concept of ash sorting

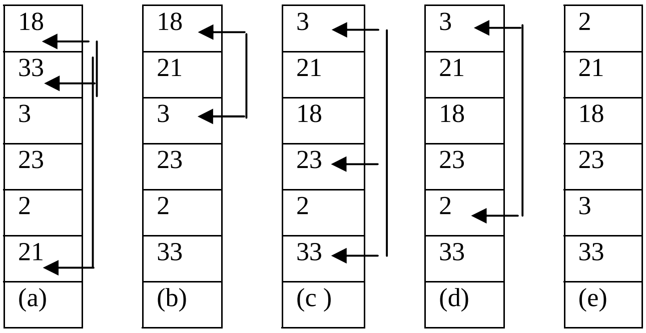


Fig 1: I Pass

In ash sorting we start from first element and compare it with next element (i.e., 2nd element) as well as with last element (i.e. 6th in the above example) and put least value at first position, mid value at 2nd position and highest value at 6th position or the last position.

Now, the 1st element will be compared with next element (i.e., 3rd). After comparison there might be three basic options:

Case I.	1st element >3rd element
Case II.	3rd element >1st element
Case III.	both are equal

In Case I, if 1st element >3rd element, there is no need to compare the 3rd element with last element (as last element is already greater to 1st element) and just swapping of 1st element with 3rd element is required.

But in Case II, 3rd element must also be compare with last element) as it could be greater than last element). If the 3rd

element is also greater than the last element then we have to swap the values of 3rd element and last element.
In the last case i.e., Case III no swapping or further comparison is required.

In the present example, First comparison of Pass I (Fig:1 a) would be among 1st (i.e, 18), 2nd (i.e., 33) and 6th (i.e., 21) elements and finally we will get least value at first position (i.e, 18), at 2nd position value 21 will be placed and highest value i.e, 33 at 6th position.
During second comparison of Pass I (Fig:1 b), we compare 1st element (i.e, 18) with 3rd element (i.e, 3), swapping would be performed and no comparison between 3rd and 6th is required (Case I).

Next comparison(Fig:1 c) will be made among 1st (i.e, 3), 4th (i.e, 23) and 6th (i.e., 33) elements but initially between 1st and 4th. As 1st element is less than the 4th element (Case II) comparison between 4th and 6th is also needed. Although, no swapping is required in the above example.
In the last comparison (Fig:1 d), only 1st element would be compared with 5th element (Case I) and swapping will be performed.

I pass is now completed and after this least and highest elements will be placed at correct positions (Fig:1 e).

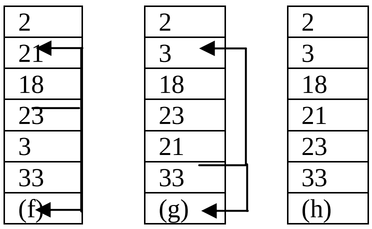


Fig 2: II Pass

II pass: At the start of II pass, the same procedure will be followed with 2nd, 3rd and second last element i.e., 5th in the present example (Fig 2: f) and the values are 21, 18 and 3. After the first comparison, 2nd position will be occupied by 3, 3rd position will be occupied by 18 and 21 will be stored at 5th position.
Next comparison would be among 2nd, 4th and 5th elements (Fig: 2 g). As 3 (2nd element) is less than 23 (4th element) that's why 23 will also be compared with 5th element i.e, 21 (Case II). And swapping would be performed between 4th and 5th elements.

II pass is now completed and after this 2nd least and 2nd highest elements will be placed at correct positions (Fig: 2 h).

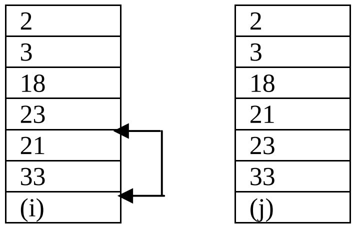


Fig 3: III Pass

III pass: in the third pass only 3rd and 4th elements would be compared (Fig: 3 i) and positioned at correct places.
After III pass all the elements get sorted and placed at right positions (Fig 3 j).

3. Algorithm of Ash Sort:

```
Procedure Ash (Array arr, Number initial_index, Number lst)
Begin
For i= initial_index to lst/2
Begin
Flag=0
For j=initial_index+1 to lst
Begin
        If flag=0 then
          Sort (arr[i],arr[j], arr[lst])
          flag=1
        else
        if arr[i]>arr[j] then            Rem  Case I
          tmp=arr[i]
          arr[i]=arr[j]
          arr[j]=tmp
        else
                if arr[j]>arr[lst] then          Rem: Case II
                        tmp=arr[j]
                        arr[j]=arr[lst]
                        arr[lst]=tmp
                 end if
                end if
          end if
        end loop
                lst=lst-1
      end loop
   end procedure
```

4. 'C' Program of Ash Sort:

```
#include<stdio.h>
void main()
{
        int ar[30],flag=0;
/* flag is used to specify whether comparing 3 values or two at a time */
```

```
            int i,j,tmp,f,l,lst=29999,s;
/* Initially lst will store last value of array index which will
be decremented afterwards */
            clrscr();
            /* Assigning values to the array: worst case */
            for(i=29,j=0;i>=0;i--,j++)
             ar[j]=i;
            /* Ash Sorting outer loop which will be executed
for half number of times to the total number of elements */
            for(i=0;i<15;i++)
            {      flag=0;
                        for(j=i+1;j<=lst;j++) /* inner loop starts
from i+1 th element to the lst */
                        {
                              if(flag==0)
                  /* every 1st comparison of each pass to place
3 values at proper places: Sorting of three elements*/
{
     if(ar[i]>ar[j])
    {
     if(ar[i]>ar[lst])
     {
            if(ar[j]>ar[lst])
            {
               f=ar[lst];
               s=ar[j];
               l=ar[i];
             }
            else
            {
              f=ar[j];
              s=ar[lst];
              l=ar[i];
             }
           }
           else
          {
            f=ar[j];
            s=ar[i];
            l=ar[lst];
         }
      }
else
  if(ar[j]>ar[lst])
   {
     f=ar[i];
    s=ar[lst];
    l=ar[j];
 }
else
{
  f=ar[i];
 s=ar[j];
 l=ar[lst];
  }
             ar[i]=f;
            ar[j]=s;
            ar[lst]=l;
             flag=1;
 }
        else
       {
            /* all comparisons except 1st of each Pass */
            if(ar[i]>ar[j])               /* Case I */
            {
                tmp=ar[i];
               ar[i]=ar[j];
               ar[j]=tmp;
            }
            else
              if(ar[j]>ar[lst]) /* Case II */
              {
                        tmp=ar[j];
                        ar[j]=ar[lst];
                        ar[lst]=tmp;
              }
     }
}
            lst=lst-1;
}
// To Clear The screen and Display the sorted Array
clrscr();
for(i=0;i<30;i++)
printf("%d\t",ar[i]);
getch();
}
```

5. Observations and Results:

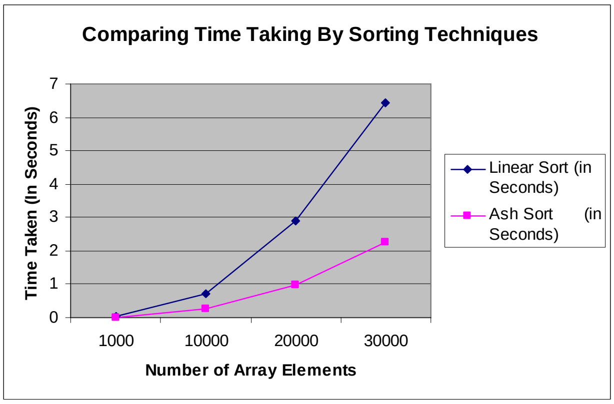
We have conducted linear sort and ash sort on different number of elements arranged in descending order. In the first set of experiment we have taken array of 1000 elements having values 0-999 arranged in descending order (ar[0..999]=999..0) and executed both the programs for five times and calculated the mean time taken by the program to sort the array of 1000 elements in ascending order. Same results were recorded on array of 10000, 20000 and 30000 elements respectively. Table 1 shows the observations recorded. As the results clearly show that the ash sorting is consuming less time as compared to linear sort in all the cases. As we increase the number of elements, the difference also gets increased.

6. Conclusion

Ash sorting is comparison based algorithm which is very simple in writing and in implementation too. Results indicate that it is about two – three times faster (depending on the number of elements being sorted) algorithm than linear sort. In ash sorting, the number of comparisons get reduced thus enhance the speed of the sorting. This could be beneficial algorithm where one wants to sort a large number of elements in less time in comparatively easy manner.

Table 1: Comparison with Linear sort

No of elements in array				
Mean Time → Taken in Sorting	**1000**	**10000**	**20000**	**30000**
Linear Sort (in Seconds) ↓	0.025	0.72	2.9	6.43
Ash Sort (in Seconds)	0	0.28	0.99	2.27



Graph 1: Comparing Time Taken By Sorting Techniques

References:

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. MIT Press, Second edition, Sept. 2001.

[2] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In Sixth Symposium on Operating System Design and Implementation, Dec. 2004.

[3] G. Graefe. Implementing sorting in database systems. ACM Comput. Surv., 38(3):10, 2006.

[4] B. He, W. Fang, N. K. Govindaraju, Q. Luo, and T. Wang. Mars: A MapReduce framework on graphics processors. Technical Report HKUST-CS07-14, Department of Computer Science and Engineering, HKUST, Nov. 2007.

[5] D. E. Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching. Addison-Wesley, Boston, MA, Second edition, 1998.