geeksforgeeks.org

Comb Sort - GeeksforGeeks

5-6 minutes

Comb Sort is mainly an improvement over Bubble Sort. Bubble sort always compares adjacent values. So all inversions are removed one by one. Comb Sort improves on Bubble Sort by using gap of size more than 1. The gap starts with a large value and shrinks by a factor of 1.3 in every iteration until it reaches the value 1. Thus Comb Sort removes more than one inversion counts with one swap and performs better than Bublle Sort.

The shrink factor has been empirically found to be 1.3 (by testing Combsort on over 200,000 random lists) [Source: Wiki]

Although, it works better than Bubble Sort on average, worst case remains $O(n^2)$.

Below is C++ implementation.

- C++
- Java
- Python

C++

```
#include<bits/stdc++.h>

using namespace std;

int getNextGap(int gap)

{
```

```
    gap = (gap*10)/13;
    if (gap < 1)
        return 1;
    return gap;
}
void combSort(int a[], int n)
{
    int gap = n;
    bool swapped = true;
    while (gap != 1 || swapped == true)
    {
        gap = getNextGap(gap);
        swapped = false;
        for (int i=0; i<n-gap; i++)
        {
            if (a[i] > a[i+gap])
            {
                swap(a[i], a[i+gap]);
                swapped = true;
            }
        }
    }
}
int main()
```

```
{
    int a[] = {8, 4, 1, 56, 3, -44, 23, -6, 28,
0};
    int n = sizeof(a)/sizeof(a[0]);
    combSort(a, n);
    printf("Sorted array: \n");
    for (int i=0; i<n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Java

```
class CombSort
{
    int getNextGap(int gap)
    {
        gap = (gap*10)/13;
        if (gap < 1)
            return 1;
        return gap;
    }
    void sort(int arr[])
    {
        int n = arr.length;
```

```
        int gap = n;
        boolean swapped = true;
        while (gap != 1 || swapped == true)
        {
            gap = getNextGap(gap);
            swapped = false;
            for (int i=0; i<n-gap; i++)
            {
                if (arr[i] > arr[i+gap])
                {
                    int temp = arr[i];
                    arr[i] = arr[i+gap];
                    arr[i+gap] = temp;
                    swapped = true;
                }
            }
        }
    }
    public static void main(String args[])
    {
        CombSort ob = new CombSort();
        int arr[] = {8, 4, 1, 56, 3, -44, 23, -6,
28, 0};
        ob.sort(arr);
```

```
        System.out.println("sorted array");
        for (int i=0; i<arr.length; ++i)
            System.out.print(arr[i] + " ");
    }
}
```

Python

```
def getNextGap(gap):
    gap = (gap * 10)/13
    if gap < 1:
        return 1
    return gap
def combSort(arr):
    n = len(arr)
    gap = n
    swapped = True
    while gap !=1 or swapped == 1:
        gap = getNextGap(gap)
        swapped = False
        for i in range(0, n-gap):
            if arr[i] > arr[i + gap]:
                arr[i], arr[i + gap]=arr[i + gap],
arr[i]
                swapped = True
```

```
arr = [ 8, 4, 1, 3, -44, 23, -6, 28, 0]

combSort(arr)

print ("Sorted array:")

for i in range(len(arr)):

    print (arr[i]),
```

Output :

```
Sorted array:
-44 -6 0 1 3 4 8 23 28 56
```

Illustration:

Let the array elements be

```
8, 4, 1, 56, 3, -44, 23, -6, 28, 0
```

Initially gap value = 10

After shrinking gap value => 10/1.3 = **7**;

```
 8  4  1  56  3  -44  23  -6  28  0
-6  4  1  56  3  -44  23   8  28  0
-6  4  0  56  3  -44  23   8  28  1
```

New gap value => 7/1.3 = **5**;

```
-44  4  0  56  3  -6  23  8  28  1
-44  4  0  28  3  -6  23  8  56  1
-44  4  0  28  1  -6  23  8  56  3
```

New gap value => 5/1.3 = **3**;

```
-44  1   0  28  4  -6  23  8  56  3
-44  1  -6  28  4   0  23  8  56  3
-44  1  -6  23  4   0  28  8  56  3
-44  1  -6  23  4   0   3  8  56  28
```

New gap value => 3/1.3 = **2**;

```
-44  1  -6  0  4  23  3  8  56  28
-44  1  -6  0  3  23  4  8  56  28
```

```
-44 1 -6 0 3 8 4 23 56 28
```

New gap value => 2/1.3 = **1**;

```
-44 -6 1 0 3 8 4 23 56 28
-44 -6 0 1 3 8 4 23 56 28
-44 -6 0 1 3 4 8 23 56 28
-44 -6 0 1 3 4 8 23 28 56

no more swaps required (Array sorted)
```

Time Complexity : Worst case complexity of this algorithm is $O(n^2)$ and the Best Case complexity is O(n).

Auxiliary Space : O(1).

Quiz on Comb Sort

This article is contributed by **Rahul Agrawal**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Snapshots:

Let the array elements be

8	4	1	56	3	-44	23	-6	28	0

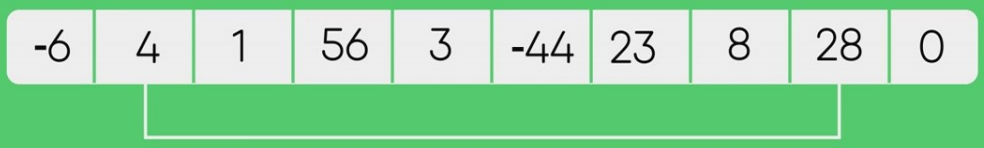
Gap value	Run No	Comments
10	1	No change

Let the array elements be

8	4	1	56	3	-44	23	-6	28	0

Gap value	Run No	Comments
10/1.3 = 7	2	Compare and swap values

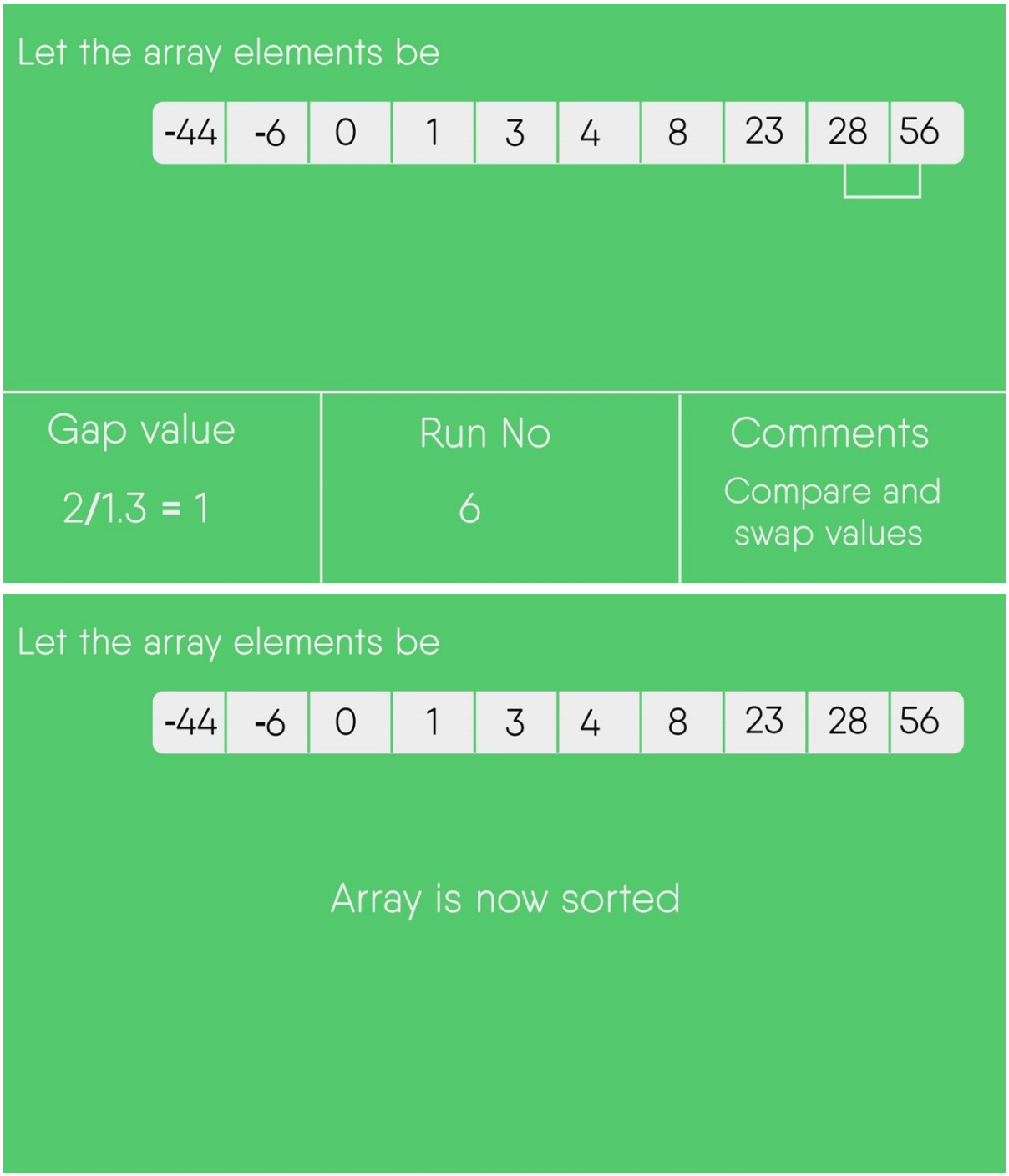
Let the array elements be



Gap value	Run No	Comments
10/1.3 = 7	2	Compare and swap values

Let the array elements be

Gap value	Run No	Comments
2/1.3 = 1	6	Compare and swap values



Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz
Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Pigeonhole Sort

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.