# Gnome Sort- A Stupid One

**Program Name-** Gnome Sort/ Stupid Sort

**Project Category-** Sorting

**Explanation-**

One of the simplest sorting algorithm- Gnome Sort is based on the concept of a Garden Gnome sorting his flower pots.

A garden gnome sorts the flower pots by the following method-

He looks at the flower pot next to him and the previous one; if they are in the right order he steps one pot forward, otherwise he swaps them and steps one pot backwards.
If there is no previous pot (he is at the starting of the pot line), he steps forwards; if there is no pot next to him (he is at the end of the pot line), he is done.

Hence our algorithm is as follows-

We assume that our input array is – arr[] and there are ‘n’ elements in it.

1) If you are at the start of the array then go to the right element (from arr[0] to arr[1]).

2) If the current array element is larger or equal to the previous array element then go one step right

*Equivalent Code-*

```
if (arr[i] >= arr[i-1])
i++;
```

3) If the current array element is smaller than the previous array element then swap these two elements and go one step backwards

*Equivalent Code-*

```
if (arr[i] < arr[i-1])
{
swap (arr[i], arr[i-1]);
i--;
}
```

4) Repeat steps 2) and 3) till ‘i’ reaches the end of the array (i.e- ‘n-1’)

5) If the end of the array is reached then stop and the array is sorted.

**Example-**

Let us take the below example to show how Gnome Sort works-

We take the below array as our example

30 87 72 5 48 15 80 45 59 86

The array elements which are underlined are the pair under consideration.
The array elements which are coloured “**red**” are the pair which needs to be swapped.
The result of the swapping of each of the “**red**” pair is coloured as “**blue**”

**30 87** 72 5 48 15 80 45 59 86

30 **87 72** 5 48 15 80 45 59 86

30 **72 87** 5 48 15 80 45 59 86

**30 72** 87 5 48 15 80 45 59 86

30 72 **87 5** 48 15 80 45 59 86

30 72 **5 87** 48 15 80 45 59 86

30 **72 5** 87 48 15 80 45 59 86

30 **5 72** 87 48 15 80 45 59 86

**30 5** 72 87 48 15 80 45 59 86

**5 30** 72 87 48 15 80 45 59 86

5 30 72 **87 48** 15 80 45 59 86

5 30 72 **48 87** 15 80 45 59 86

5 30 **72 48** 87 15 80 45 59 86

5 30 **48 72** 87 15 80 45 59 86

5 **30 48** 72 87 15 80 45 59 86

5 30 48 72 **87 15** 80 45 59 86

5 30 48 72 **15 87** 80 45 59 86

5 30 48 **72 15** 87 80 45 59 86

5 30 48 **15 72** 87 80 45 59 86

5 30 **48 15** 72 87 80 45 59 86

5 30 **15 48** 72 87 80 45 59 86

5 **30 15** 48 72 87 80 45 59 86

5 **15 30** 48 72 87 80 45 59 86

**5 15** 30 48 72 87 80 45 59 86

5 15 30 48 72 **87 80** 45 59 86

5 15 30 48 72 **80 87** 45 59 86

5 15 30 48 **72 80** 87 45 59 86

5 15 30 48 72 80 **87 45** 59 86

5 15 30 48 72 80 **45 87** 59 86

5 15 30 48 72 **80 45** 87 59 86

5 15 30 48 72 **45 80** 87 59 86

5 15 30 48 **72 45** 80 87 59 86

5 15 30 48 **45 72** 80 87 59 86

5 15 30 **48 45** 72 80 87 59 86

5 15 30 **45 48** 72 80 87 59 86

5 15 **30 45** 48 72 80 87 59 86

5 15 30 45 48 72 80 **87 59** 86

5 15 30 45 48 72 80 **59 87** 86

5 15 30 45 48 72 **80 59** 87 86

5 15 30 45 48 72 **59 80** 87 86

5 15 30 45 48 **72 59** 80 87 86

5 15 30 45 48 **59 72** 80 87 86

5 15 30 45 **48 59** 72 80 87 86

5 15 30 45 48 59 72 80 **87 86**

5 15 30 45 48 59 72 80 **86 87**

5 15 30 45 48 59 72 **80 86** 87

5 15 30 45 48 59 72 80 86 87 (Sorted Output)

**Point to Note-**

Gnome Sorting is also called as “Stupid Sort”.

**Time Complexity-**

As there are no nested loop (only one while) it may seem that this is a linear $O(N)$ time algorithm. But the time complexity is $O(N^2)$. This is because the variable – ‘index’ in our program doesn’t always gets incremented, it gets decremented too.

However this sorting algorithm is adaptive and performs better if the array is already/partially sorted.

**Auxiliary Space-**

This is an in-place algorithm. So $O(1)$ auxiliary space is needed.

**References-**

https://en.wikipedia.org/wiki/Gnome_sort