

Gnome Sort

Simple sorting algorithm

Group member

1910001

1931102

2030087

Abu Ahmed Rafi

Monjurul Hasan Emon

Jannat Un Nayeem Iqra

Introduction and History of the Algorithm:

- ❖ Gnome sort is also known as Stupid sort. It was introduced by Iranian computer scientist Hamid Sarbazi Azad in 2000 and later explained by Dick Grune.
- ❖ Gnome Sort is a comparison based sorting algorithm which is kind of similar to bubble sort and insertion sort. It is based on the technique used by Garden gnome to sort flower pots.

Method of gnome sort is :

- At start he is at the first pot so he will move forward to the next pot
 - After moving to the next pot he will compare the current pot with previous pot.
 - If the pot are in correct order he will move forward
 - If the order is not correct then he will swap the two pot and will go backward(previous pot)
 - After reaching the last pot there will be no pot left next to him so he stops
- ❖ Gnome sort is very simple as it does not require any nested loop and the code is also very easy to understand

Pseudocode:

Gnomesort(a[]):

index=0

while index<a.length:

 if (index ==0 or a[index] >= a[index-1]):

 index= index + 1

 else:

 temp=a[index]

 a[index] = a[index-1]

 a[index-1] = temp


 index= index - 1

Stimulation:

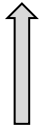
3	5	2	8	1
---	---	---	---	---

index=0

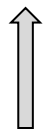
Stimulation:

Position of index shown by 

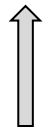
3	5	2	8	1
---	---	---	---	---



index==0
index++

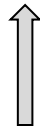


index=1



$a[1] \geq a[0]$
Index++

3	5	2	8	1
---	---	---	---	---



Index=2

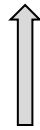


$a[\text{index}] \geq a[\text{index}-1]$

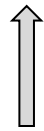
$a[2] \geq a[1]$



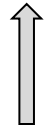
Condition is not true else statement is going to be executed



Swapping between $a[i]$ and $a[i-1]$



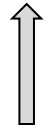
index--
index=1



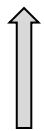
$a[1] \geq a[0]$



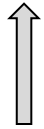
Else statement gonna be executed



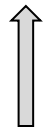
Swapping between $a[i]$ and $a[i-1]$



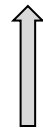
index--
index=0



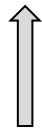
index==0
index++



index=1



$a[i] \geq a[i-1]$
index++



index=2



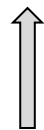
$a[i] \geq a[i-1]$
index++



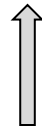
index=3



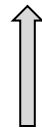
$a[i] \geq a[i-1]$
index++



index=4



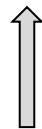
Else statement is going to be
executed



Swapping

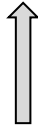


Index decremented
Index=3

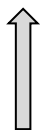


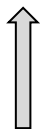


Swapping between $a[i]$ and $a[i-1]$

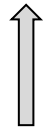


Index decremented
Index=2

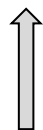


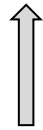


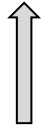
Swapping



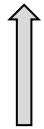
Index decremented
Index=1



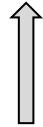




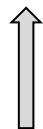
Index decremented
Index=0



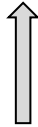
Index == 0
Index++



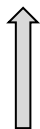
Index =1



Index ++



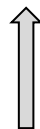
Index =2



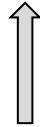
Index ++



Index =3



Index ++



Index =4



Index ++

1	2	3	5	8
---	---	---	---	---

Index =5
While loop ended
Array is sorted

Best Case Analysis:

If the array is already sorted then the while loop will only run n (size of array) times as there is no need of swapping so the index will be only incremented .

	Cost	Times
Gnomesort(a[]):	c1	1
index=0	c2	n
while index<a.length:	c3	n-1
if (index ==0 or a[index] >=	c4	n-1
a[index-1]):		
index= index + 1		
else:	c6	0
temp=a[index]	c7	0
a[index] = a[index-1]	c8	0
a[index-1] = temp	c9	0
index= index - 1		

Best Case Analysis:

$$T(n) = c_1(1) + c_2(n) + c_3(n-1) + c_4(n-1) + c_6(o) + c_7(o) + c_8(o) + c_9(o)$$

$$T(n) = c_1 + c_2n + c_3n - c_3 + c_4n - c_4$$

$$T(n) = (c_2 + c_3 + c_4)n + (c_1 - c_3 - c_4)$$

$$T(n) = a(n) + b$$

$$T(n) = n$$

Worst case scenario

$x = \text{value}$
 $\text{Count}(x) = \text{Number of time the loop runs to bring } x \text{ to its correct position and to bring index in the same position}$



$\text{count}(x) = 0$

Worst case scenario



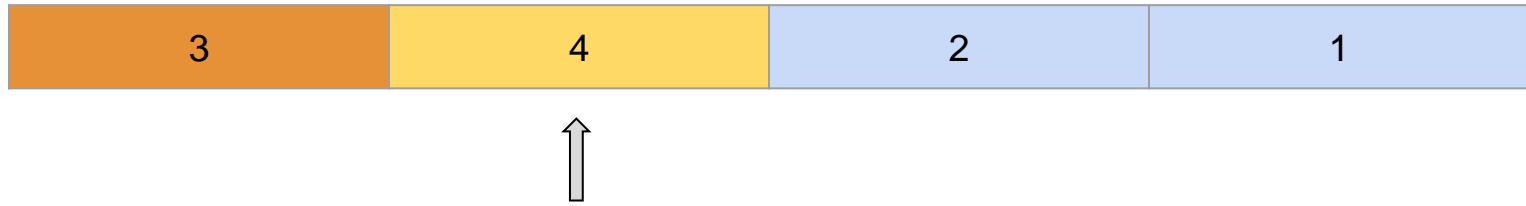
While loop run 1 time to bring 4 to its correct position
 $\text{Count}(4)=1$

Worst case scenario



Swap
count(3)=1

Worst case scenario



$\text{count}(3)=1$

Worst case scenario



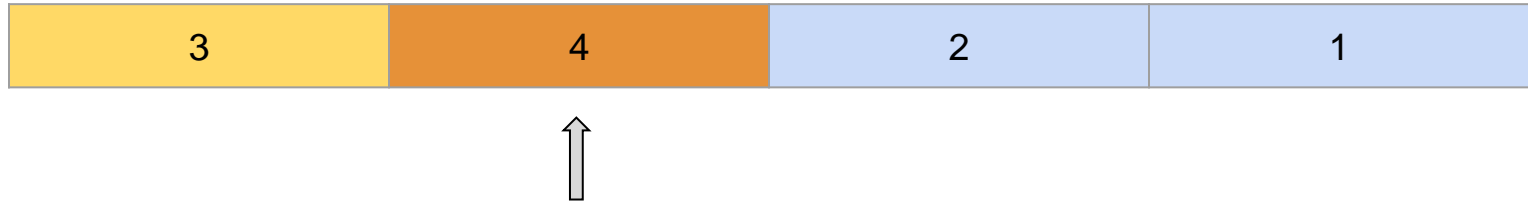
Index decremented
 $\text{count}(3)=1$

Worst case scenario



Index incremented
 $\text{count}(3)=2$

Worst case scenario



$\text{count}(3)=2$

While loop runs 2 times to bring 3 to its correct position

Worst case scenario



Index incremented

Worst case scenario



swap
count(2)=1

Worst case scenario



swap
count(2)=1

Worst case scenario



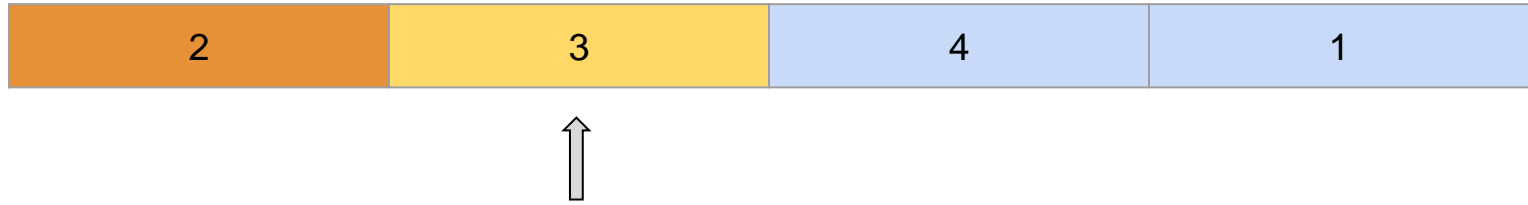
Index decremented
 $\text{count}(2)=2$

Worst case scenario



Swap
count(2)=2

Worst case scenario



Swap
count(2)=2

Worst case scenario



Index decremented
 $\text{count}(2)=2$

Worst case scenario



Index incremented
 $\text{count}(2)=3$

Worst case scenario



Index incremented
 $\text{count}(2)=3$

Worst case scenario



Index incremented
 $\text{count}(2)=4$

Worst case scenario



$\text{count}(2)=4$

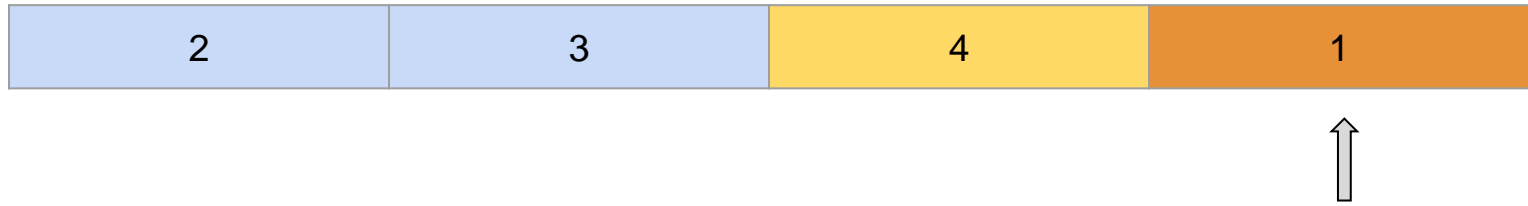
While loop runs 4 time to bring 2 to its correct position

Worst case scenario



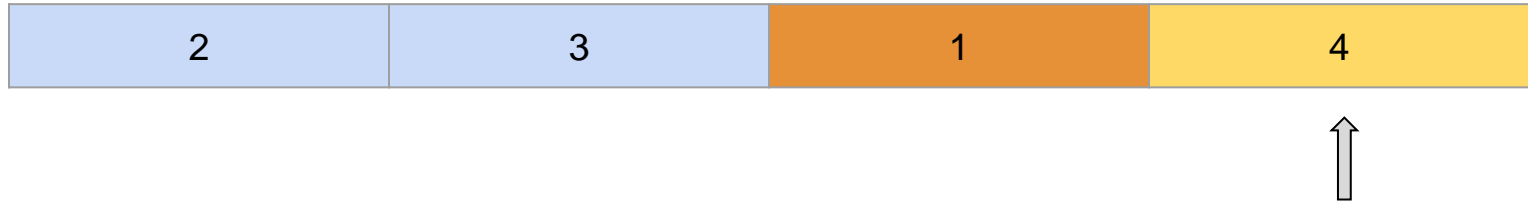
Index++

Worst case scenario



count(1)=1
swap

Worst case scenario



$\text{count}(1)=1$

Worst case scenario



count(1)=1
Index--

Worst case scenario



count(1)=2
swap

Worst case scenario



count(1)=2
swap

Worst case scenario



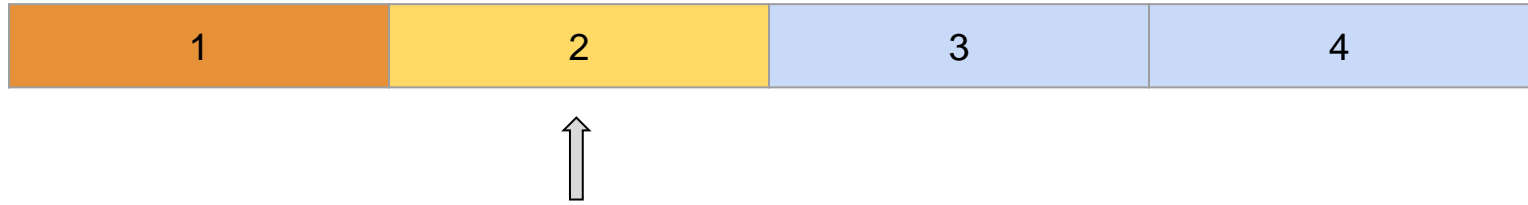
count(1)=2
Index--

Worst case scenario



count(1)=3
swap

Worst case scenario



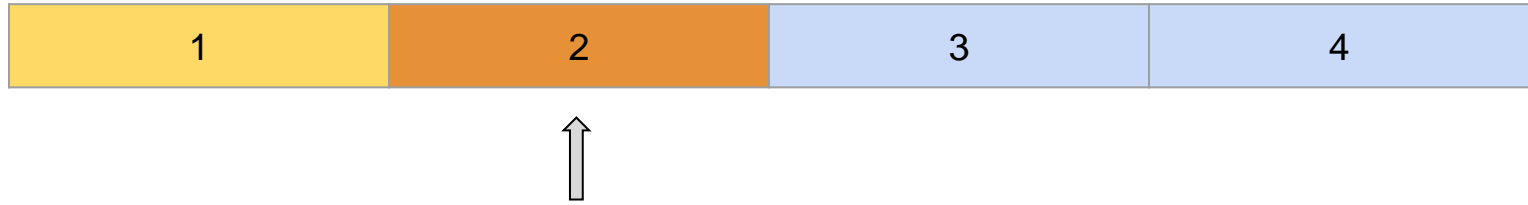
$\text{count}(1)=3$

Worst case scenario



count(1)=3
Index--

Worst case scenario



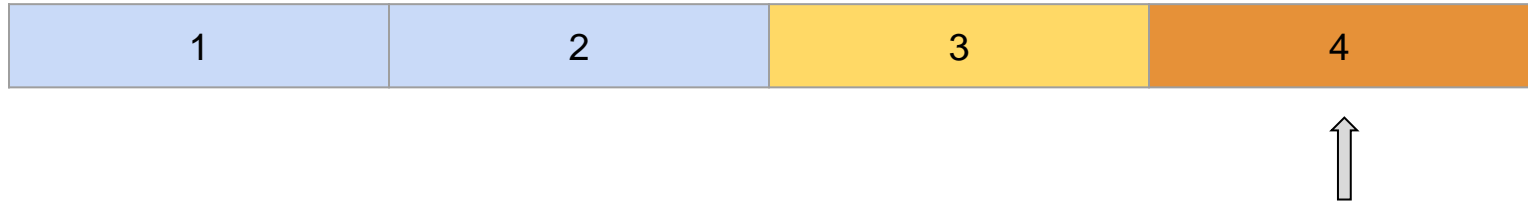
count(1)=4
Index++

Worst case scenario



count(1)=5
Index++

Worst case scenario



$\text{count}(1)=6$

While loop runs 6 time to bring 1 to its correct position

Worst Case Analysis:

So now,

Didn't took
the false
check

While loop runs total= count(4)+ count(3)+count(2)+count(1)

$$= 1+ 2+ 4+ 6$$

$$= 1 + 2(1 + 2 + 3)$$

So for n number = $1+ 2(1+2+3+ \text{-----} +n)$

$$= 1+ 2(n(n-1)/2)$$

To bring a number to its correct place swapping is done half-time of the loop and the index is also incremented half-time of the loop

Else condition runs half time= $2(n(n-1)/2)/2= n(n-1)/2$

Worst Case Analysis:

Gnomesort(a[]):

index=0

while index<a.length:

 if (index ==0 or a[index] >= a[index-1]):

 index= index + 1

 else:

 temp=a[index]

 a[index] = a[index-1]

 a[index-1] = temp

 index= index - 1

Cost

c1

c2

c3

c4

c5

c6

c7

c8

Times

1

$n(n-1)$

$n(n-1)$

$n(n-1)/2$

$n(n-1)/2$

$n(n-1)/2$

$n(n-1)/2$

$n(n-1)/2$

Worst Case Analysis:

$$T(n) = c_1(1) + c_2(n(n-1)) + c_3(n(n-1)) + c_4(n(n-1))/2 + c_5(n(n-1))/2 + c_6(n(n-1))/2 + c_7(n(n-1))/2 + c_8(n(n-1))/2$$

$$T(n) = c_1 + c_2n^2 - c_2n + c_3n^2 - c_3n + c_4n^2/2 - c_4n/2 + c_5n^2/2 - c_5n/2 + c_6n^2/2 - c_6n/2 + c_7n^2/2 - c_7n/2 + c_8n^2/2 - c_8n/2$$

$$T(n) = (c_2 + c_3 + c_4/2 + c_5/2 + c_6/2 + c_7/2 + c_8/2) n^2 - (c_2 + c_3 + c_4/2 + c_5/2 + c_6/2 + c_7/2 + c_8/2) n + c_1$$

$$T(n) = an^2 + bn + c$$

$$T(n) = n^2$$

Average Case Analysis:

In case of average case the array is partially sorted , so while loop will run half the time taken of worst case

Gnomesort(a[]):	Cost	Times
index=0	c1	1
while index<a.length:	c2	$n(n-1)/2$
if (index ==0 or a[index] >= a[index-1]):	c3	$n(n-1)/2$
index= index + 1	c4	$n(n-1)/4$
else:		
temp=a[index]	c5	$n(n-1)/4$
a[index] = a[index-1]	c6	$n(n-1)/4$
a[index-1] = temp	c7	$n(n-1)/4$
index= index - 1	c8	$n(n-1)/4$

Average Case Analysis:

$$T(n) = c_1(1) + c_2(n(n-1))/2 + c_3(n(n-1))/2 + c_4(n(n-1))/4 + c_5(n(n-1))/4 + c_6(n(n-1))/4 + c_7(n(n-1))/4 + c_8(n(n-1))/4$$

$$T(n) = c_1 + c_2n^2/2 - c_2n/2 + c_3n^2/2 - c_3n/2 + c_4n^2/4 - c_4n/4 + c_5n^2/4 - c_5n/4 + c_6n^2/4 - c_6n/4 + c_7n^2/4 - c_7n/4 + c_8n^2/4 - c_8n/4$$

$$T(n) = (c_2/2 + c_3/2 + c_4/4 + c_5/4 + c_6/4 + c_7/4 + c_8/4) n^2 - (c_2/2 + c_3/2 + c_4/4 + c_5/4 + c_6/4 + c_7/4 + c_8/4) n + c_1$$

$$T(n) = an^2 + bn + c$$

$$T(n) = n^2$$

Attributes:

Online:

- ❖ During the running time, even if more elements are added , the algorithm will still sort the array as it does not depend on the whole input. It works on every index individually.

Stable:

- ❖ Gnome sort is stable as two element with same value appear in same order in both the case (sorted and unsorted array) as we can see in the pseudocode if two elements have same value the index is incremented (no swapping) so the order is maintained.

Adaptive:

- ❖ Adaptive as the order of element does affect the time complexity . If input is already sorted it takes $O(n)$ time. If it is partially/ unsorted then it takes $O(n^2)$. Therefore the number of steps (how many times the while loop run) depends on the way the element are arrange.

In-Place:

- ❖ Gnome sort is in-place as it does not take help of any other array for sorting.

Additional information:

Space complexity of gnome sort is $O(1)$.

Can work with positive and negative integer number and also with positive and negative floating point numbers.

Pros and Cons:

Pros:

- It does not use any extra array for sorting (less space)
- also conceptually simple as it requires no nested loop and the code is also very easy to understand .
- Linear time complexity in case of best case.

Cons:

- Not a very efficient algorithm.Its pretty slow compared to other algorithm as in case of worst case it take $O(n^2)$.In case of insertion sort index is only incremented but here index is both incremented and decremented.

Practical Use:

- Can be used to sort array in ascending order but compare to other algorithm it's running time is $O(n^2)$ in case of average and worst case which is very slow. So, if the input is nearly sorted then gnome sort can be used as its time complexity is linear $O(n)$.
- Can be used in educational purpose as the code is simple and easy to understand