

BUBBLE SORTING

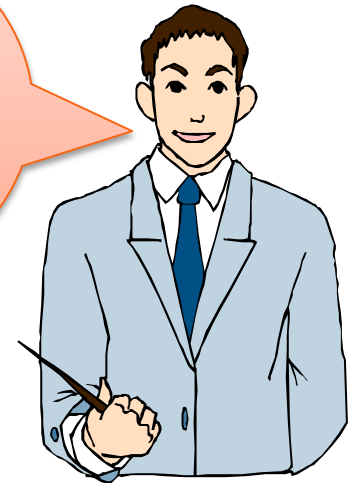
SORTING BY COMPARING ADJACENT ELEMENT

BUBBLE SORTING

- ✓ In Bubble Sorting, each element is compared with adjacent element and swap them if they are not in proper order
- ✓ In Bubble Sorting there are $N-1$ Passes and after every pass highest element is placed to its correct position
- ✓ In every pass start comparison from first element and compare and swap up to unsorted elements. So in each pass no. of comparison will be decreased by 1.
- ✓ Let us SORT using Bubble...

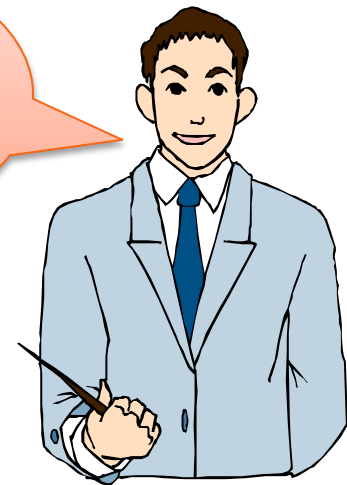
Index	value
0	17
1	6
2	12
3	1
4	8
5	4
6	21
7	10
8	7
9	5

LIST elements are
in random order
and we will SORT
it using BUBBLE
method



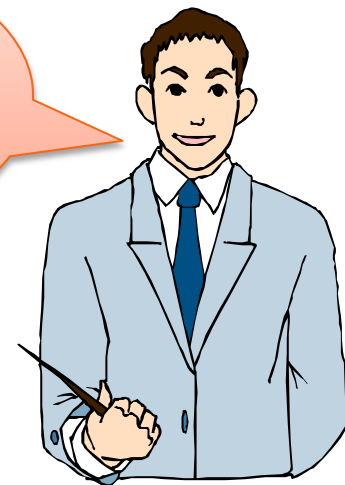
Index	value
0	17
1	6
2	12
3	1
4	8
5	4
6	21
7	10
8	7
9	5

We will first compare value at index 0 with value at index 1



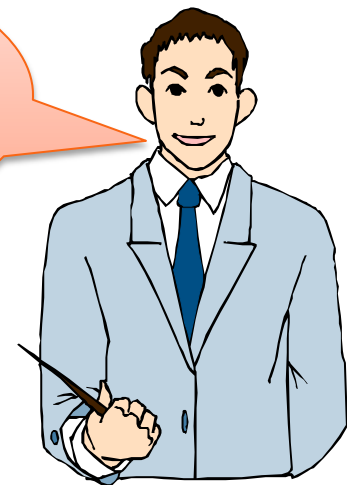
Index	value
0	17
1	6
2	12
3	1
4	8
5	4
6	21
7	10
8	7
9	5

We can see that they are not in proper order



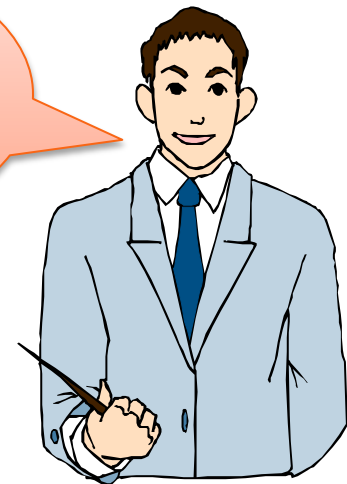
Index	value
0	17
1	6
2	12
3	1
4	8
5	4
6	21
7	10
8	7
9	5

So we will SWAP them



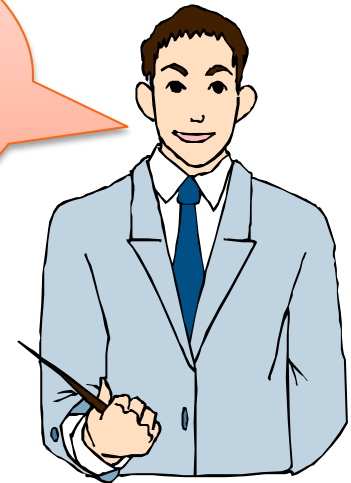
Index	value
0	6
1	17
2	12
3	1
4	8
5	4
6	21
7	10
8	7
9	5

Now we will compare value at index 1 with value at index 2



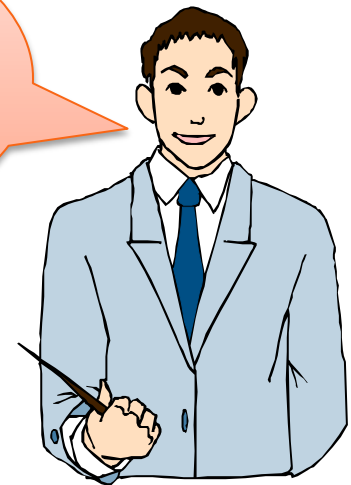
Index	value
0	6
1	17
2	12
3	1
4	8
5	4
6	21
7	10
8	7
9	5

We can see they are not in proper order so again we will SWAP them



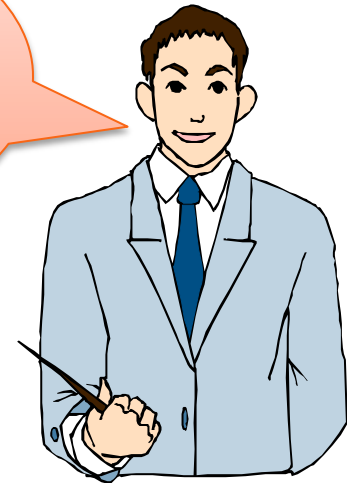
Index	value
0	6
1	12
2	17
3	1
4	8
5	4
6	21
7	10
8	7
9	5

Again we will
compare value at
index 2 with index
3



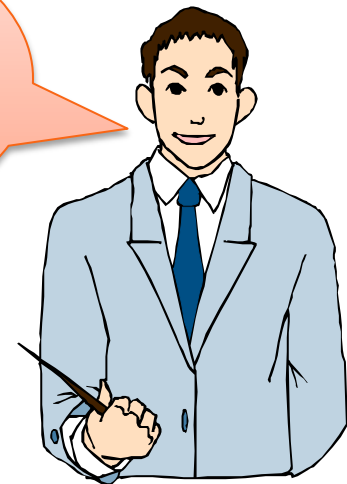
Index	value
0	6
1	12
2	17
3	1
4	8
5	4
6	21
7	10
8	7
9	5

SWAP them



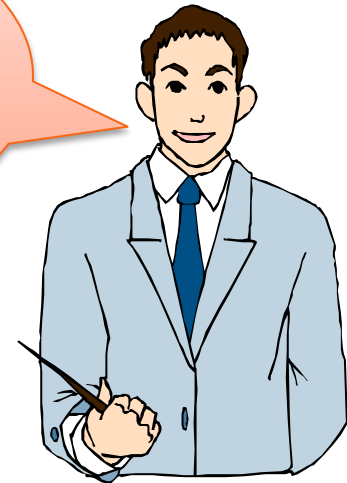
Index	value
0	6
1	12
2	1
3	17
4	8
5	4
6	21
7	10
8	7
9	5

Compare index 3
value with index 4



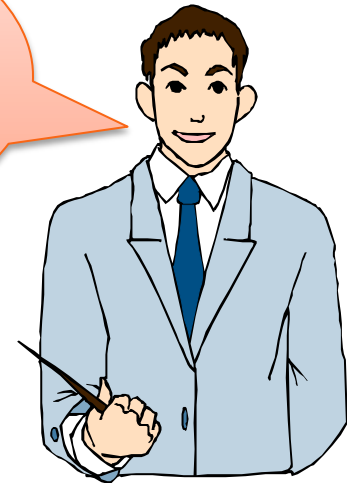
Index	value
0	6
1	12
2	1
3	17
4	8
5	4
6	21
7	10
8	7
9	5

SWAP them



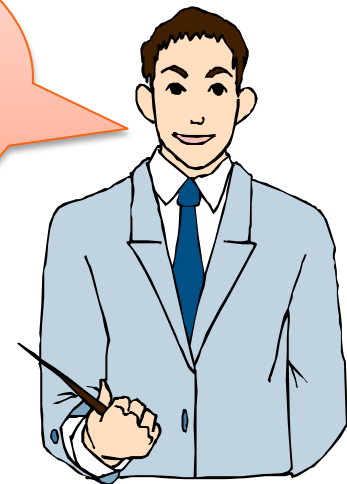
Index	value
0	6
1	12
2	1
3	8
4	17
5	4
6	21
7	10
8	7
9	5

Compare next adjacent elements



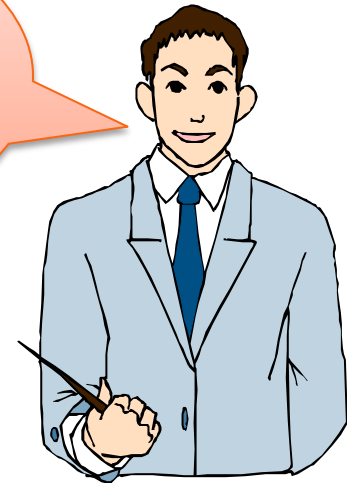
Index	value
0	6
1	12
2	1
3	8
4	17
5	4
6	21
7	10
8	7
9	5

Compare next adjacent elements



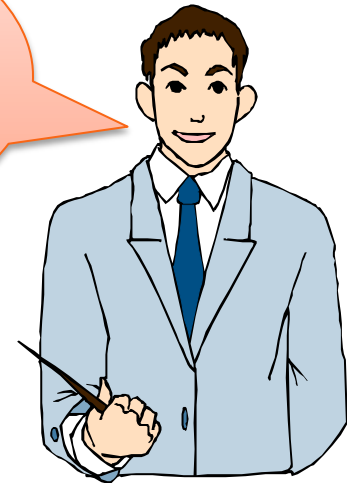
Index	value
0	6
1	12
2	1
3	8
4	17
5	4
6	21
7	10
8	7
9	5

SWAP them



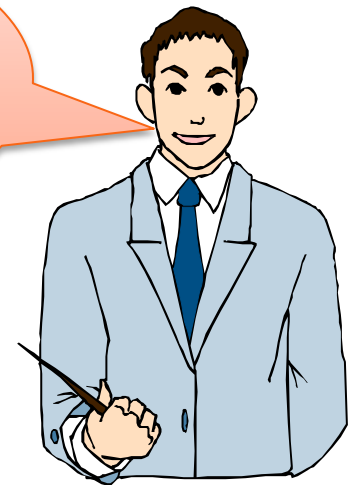
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	21
7	10
8	7
9	5

Compare next adjacent elements



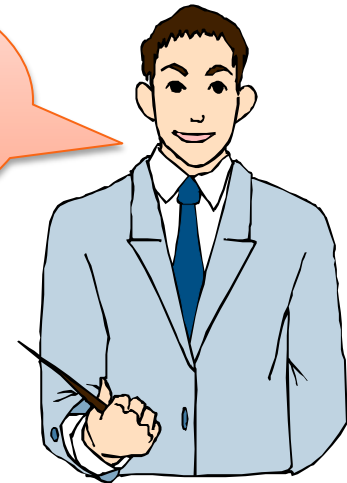
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	21
7	10
8	7
9	5

They are in proper order so we will not SWAP and continues with next adjacent elements



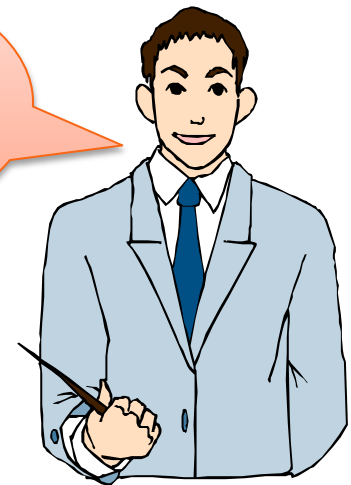
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	21
7	10
8	7
9	5

SWAP them



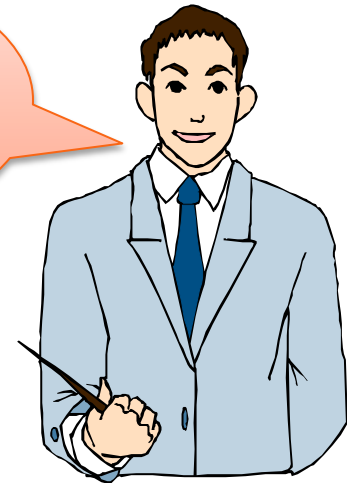
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	21
8	7
9	5

Compare next
adjacent
elements



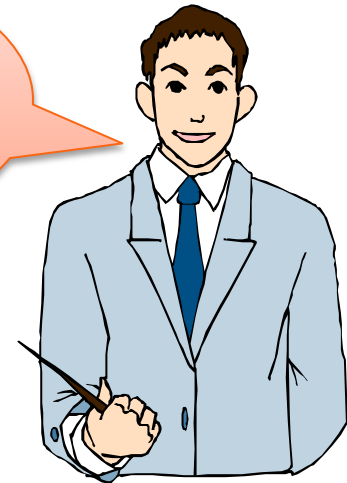
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	21
8	7
9	5

SWAP them



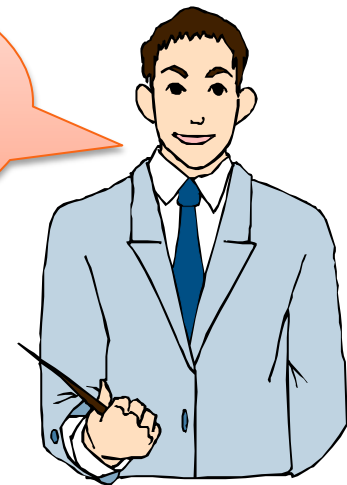
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	7
8	21
9	5

Compare next adjacent elements



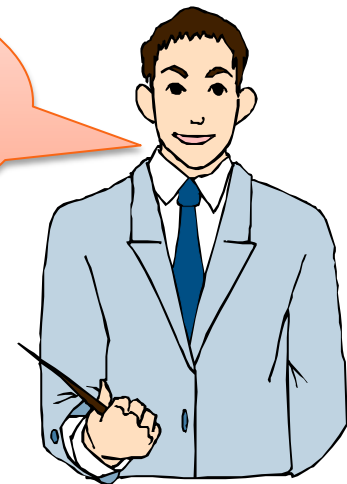
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	7
8	21
9	5

SWAP them



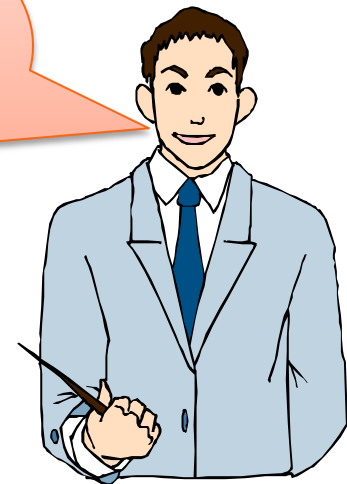
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	7
8	5
9	21

Now we can see that the highest element is placed at its correct position



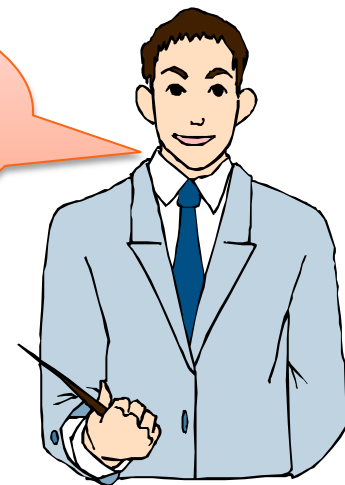
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	7
8	5
9	21

i.e. from index 0 to 8 is now unsorted part and from index 9 it is sorted, so in next pass we will compare only up to index 8



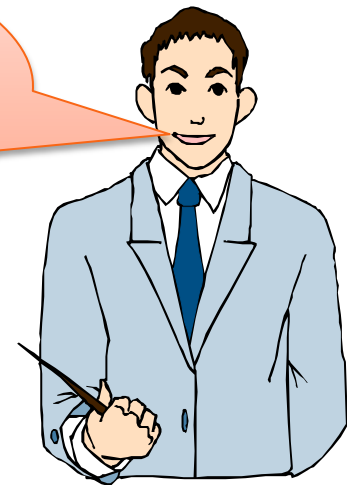
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	7
8	5
9	21

In Pass 1 total comparisons were $<N-1$ i.e. 9

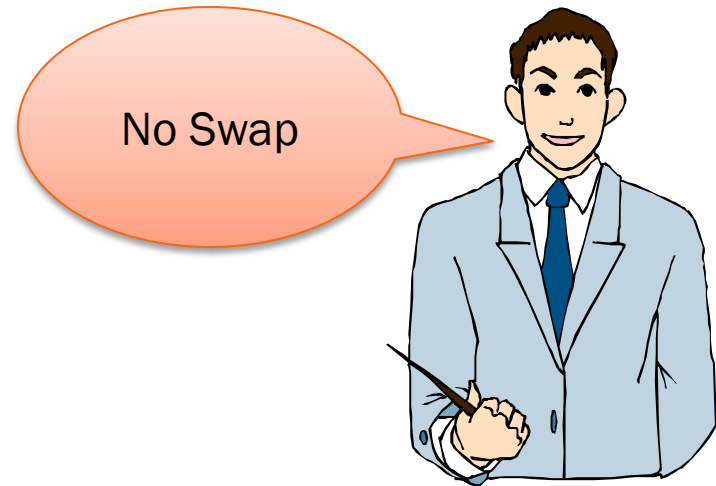


Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	7
8	5
9	21

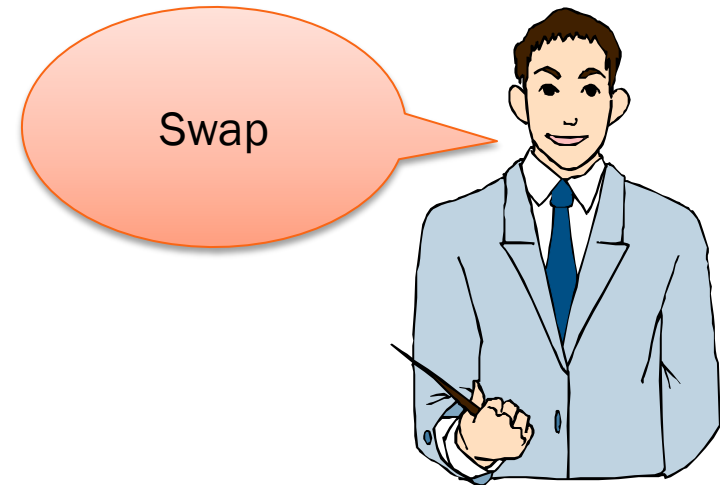
In Pass 2 total comparisons will be $<N-2$ because one element is already sorted



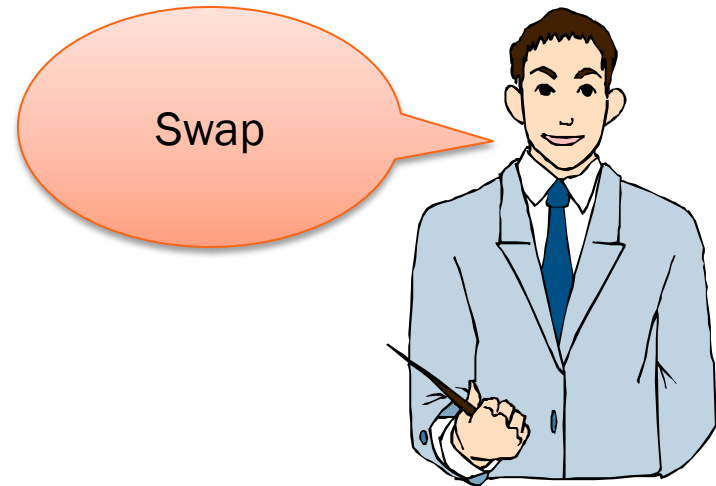
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	7
8	5
9	21



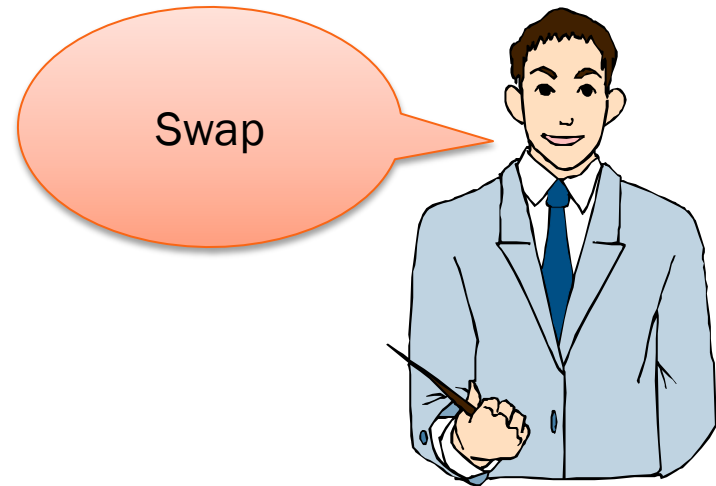
Index	value
0	6
1	12
2	1
3	8
4	4
5	17
6	10
7	7
8	5
9	21



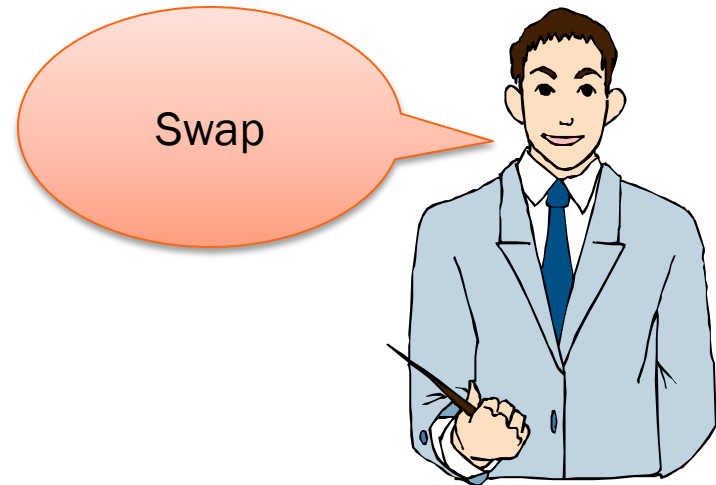
Index	value
0	6
1	1
2	12
3	8
4	4
5	17
6	10
7	7
8	5
9	21



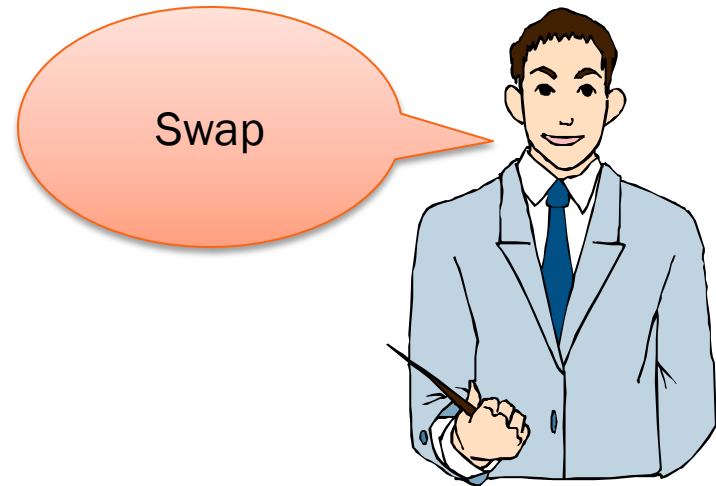
Index	value
0	6
1	1
2	12
3	8
4	4
5	17
6	10
7	7
8	5
9	21



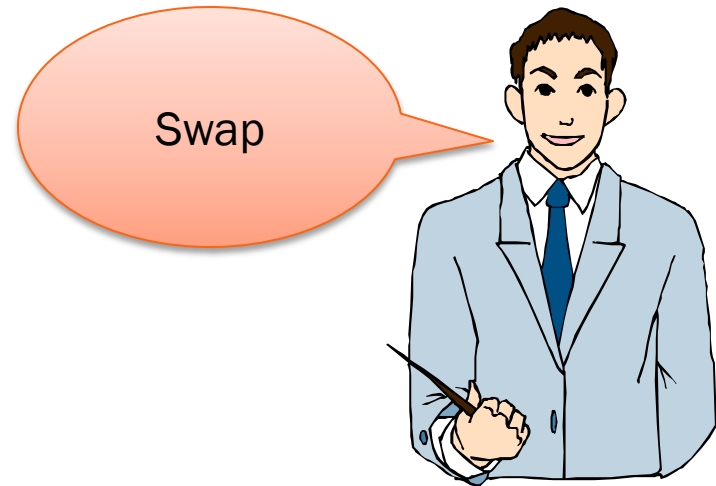
Index	value
0	6
1	1
2	8
3	12
4	4
5	17
6	10
7	7
8	5
9	21



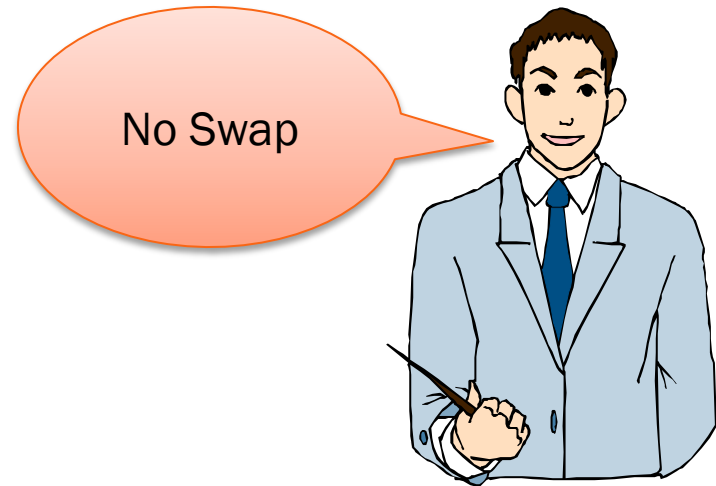
Index	value
0	6
1	1
2	8
3	12
4	4
5	17
6	10
7	7
8	5
9	21



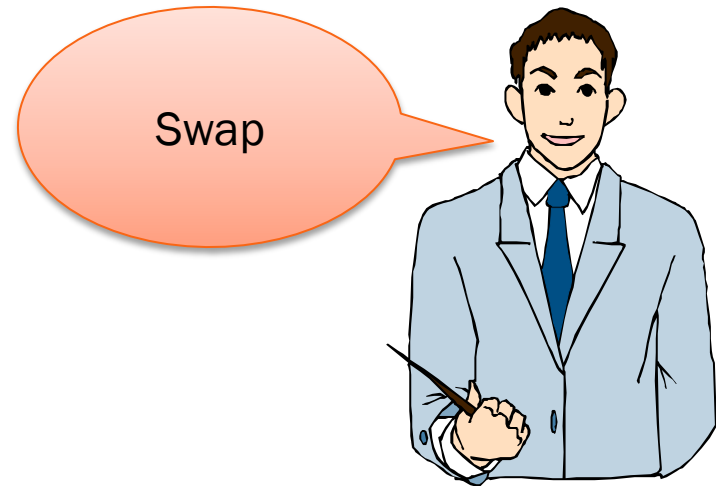
Index	value
0	6
1	1
2	8
3	4
4	12
5	17
6	10
7	7
8	5
9	21



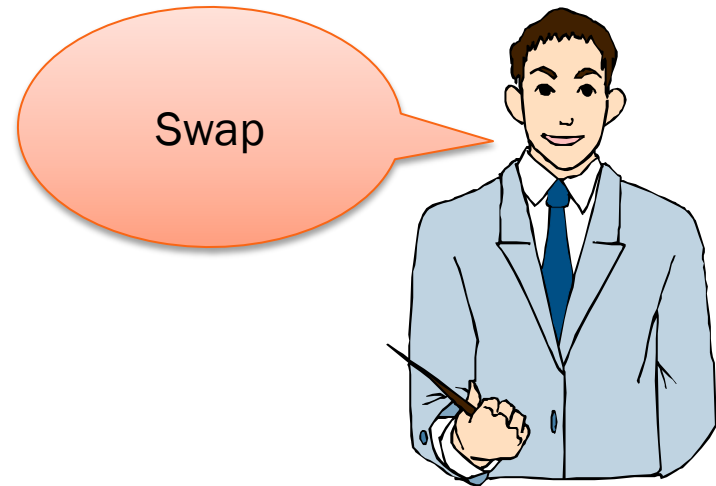
Index	value
0	6
1	1
2	8
3	4
4	12
5	17
6	10
7	7
8	5
9	21



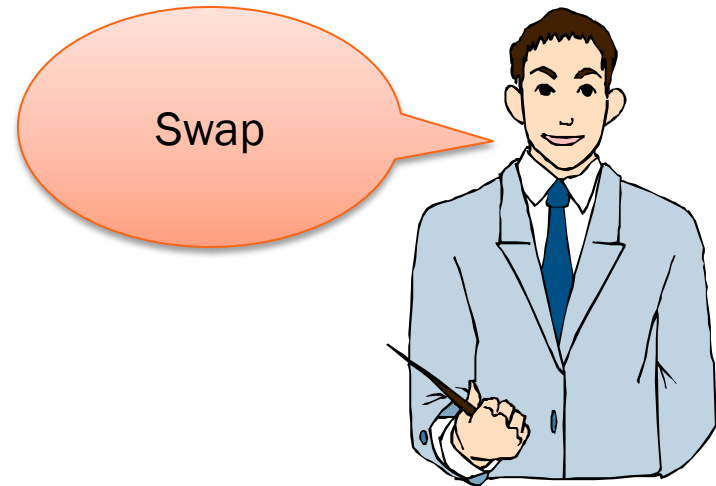
Index	value
0	6
1	1
2	8
3	4
4	12
5	17
6	10
7	7
8	5
9	21



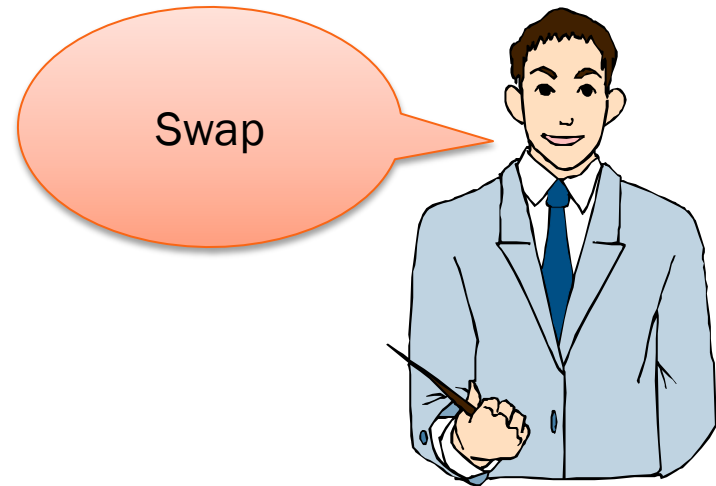
Index	value
0	6
1	1
2	8
3	4
4	12
5	10
6	17
7	7
8	5
9	21



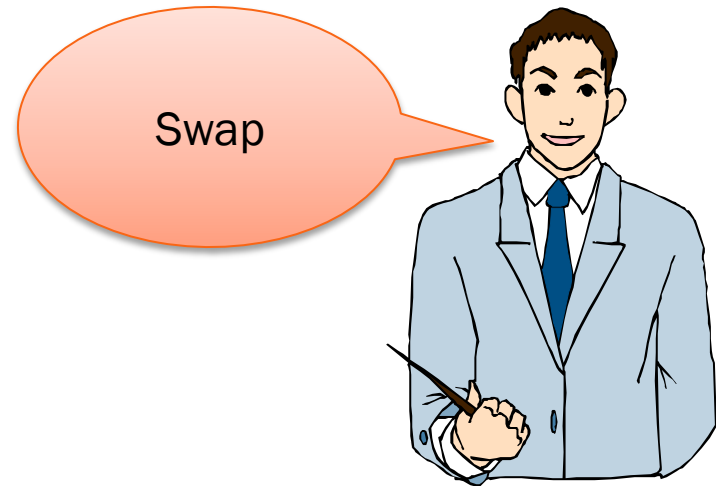
Index	value
0	6
1	1
2	8
3	4
4	12
5	10
6	17
7	7
8	5
9	21



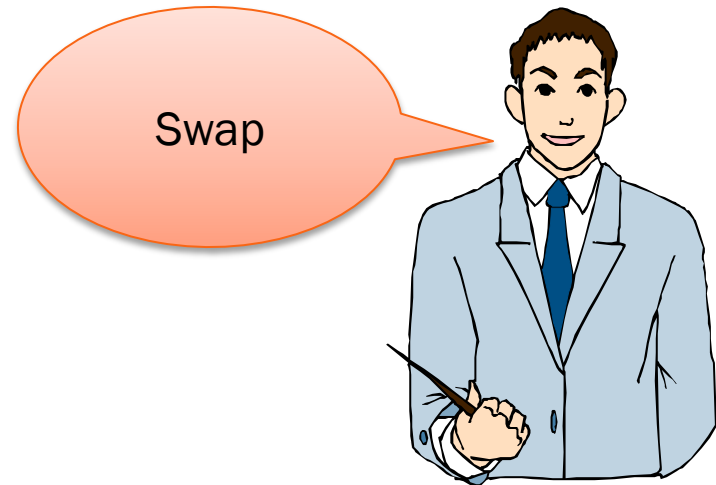
Index	value
0	6
1	1
2	8
3	4
4	12
5	10
6	7
7	17
8	5
9	21



Index	value
0	6
1	1
2	8
3	4
4	12
5	10
6	7
7	17
8	5
9	21

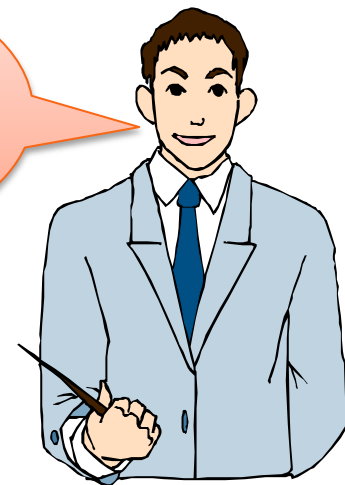


Index	value
0	6
1	1
2	8
3	4
4	12
5	10
6	7
7	5
8	17
9	21



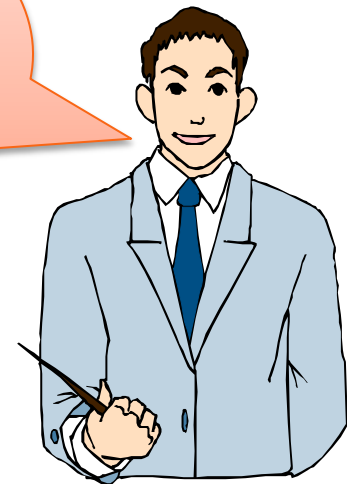
Index	value
0	6
1	1
2	8
3	4
4	12
5	10
6	7
7	5
8	17
9	21

Now 17 is at
its correct
position



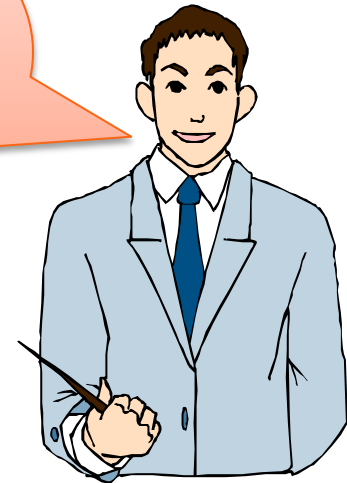
Index	value
0	6
1	1
2	8
3	4
4	12
5	10
6	7
7	5
8	17
9	21

From index 8 to 9
elements are
sorted, so next
sorting will be from
0 to 7

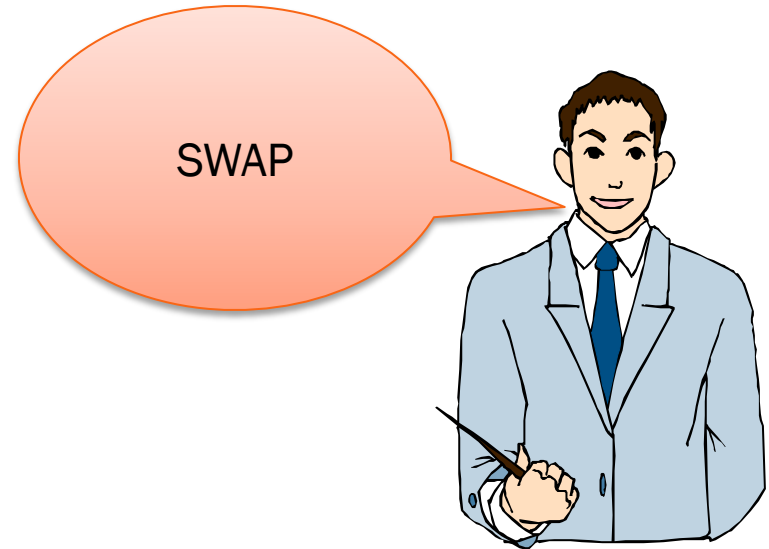


Index	value
0	6
1	1
2	8
3	4
4	12
5	10
6	7
7	5
8	17
9	21

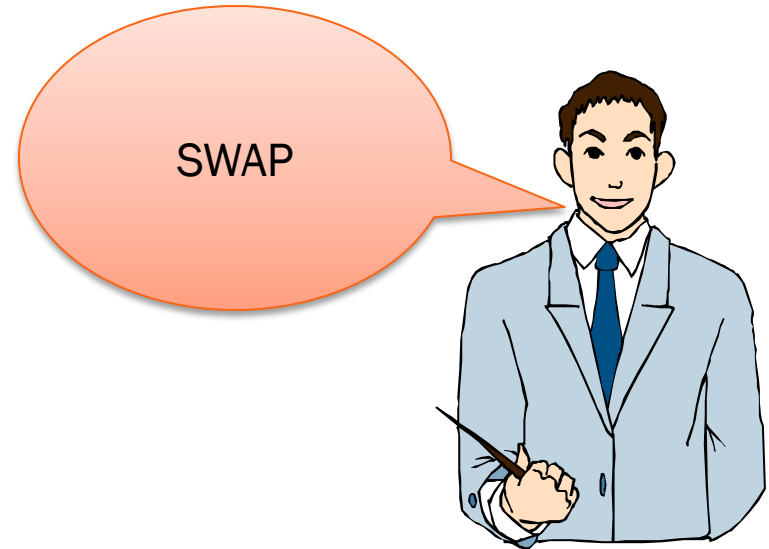
In Pass 2 total
comparison were
 $<N-2$ i.e. 7



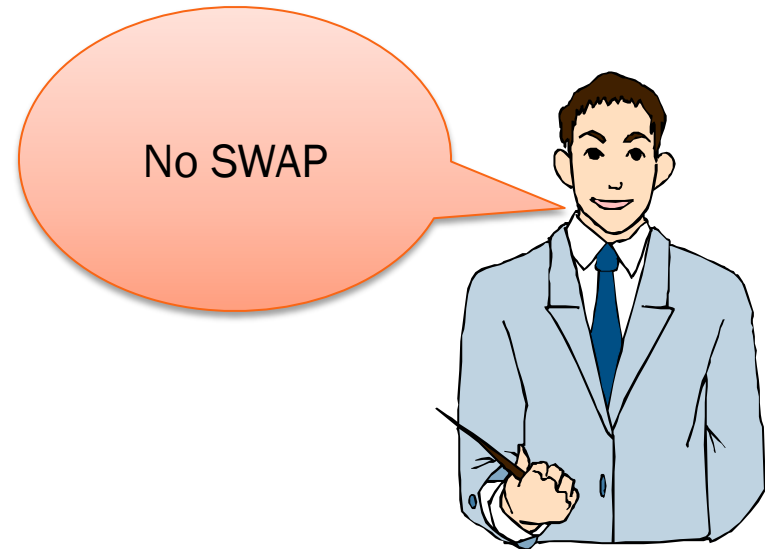
Index	value
0	6
1	1
2	8
3	4
4	12
5	10
6	7
7	5
8	17
9	21



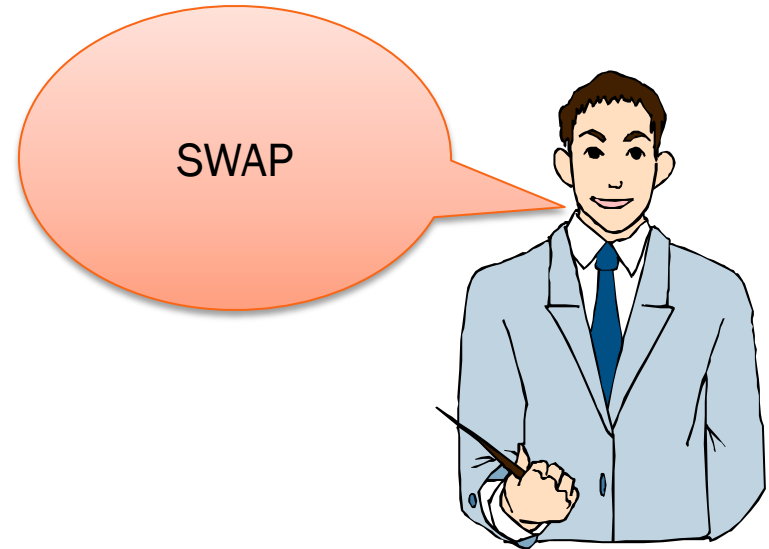
Index	value
0	1
1	6
2	8
3	4
4	12
5	10
6	7
7	5
8	17
9	21



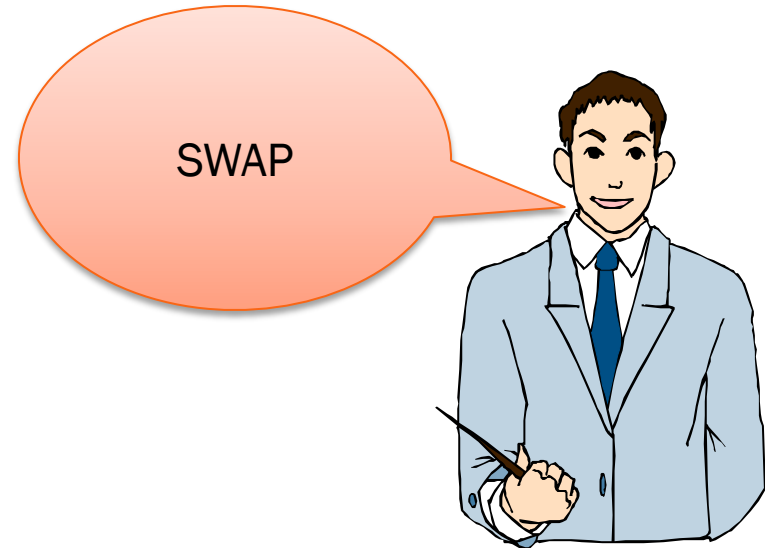
Index	value
0	1
1	6
2	8
3	4
4	12
5	10
6	7
7	5
8	17
9	21



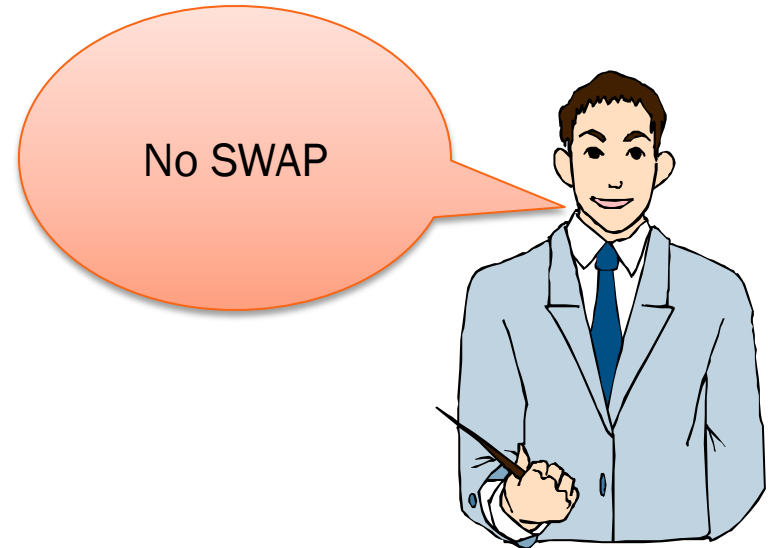
Index	value
0	1
1	6
2	8
3	4
4	12
5	10
6	7
7	5
8	17
9	21



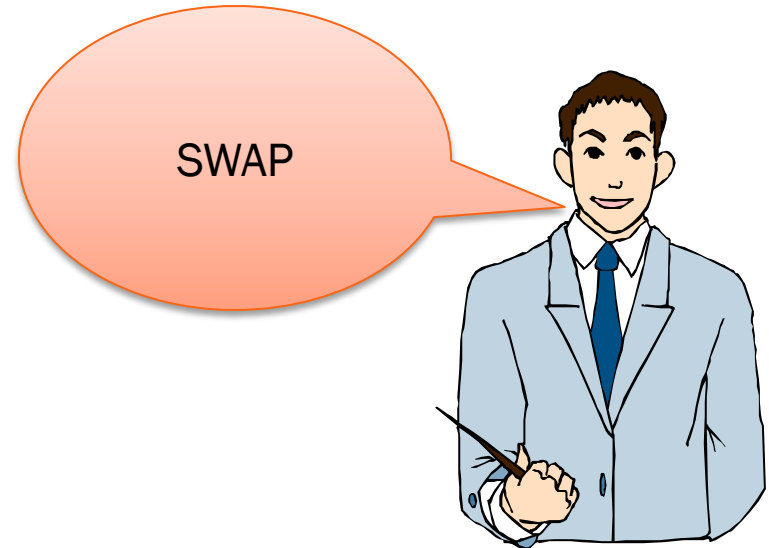
Index	value
0	1
1	6
2	4
3	8
4	12
5	10
6	7
7	5
8	17
9	21



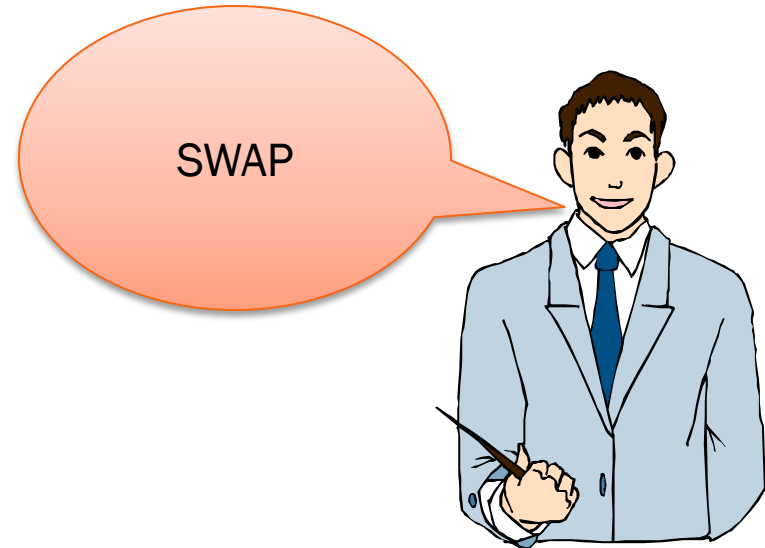
Index	value
0	1
1	6
2	4
3	8
4	12
5	10
6	7
7	5
8	17
9	21



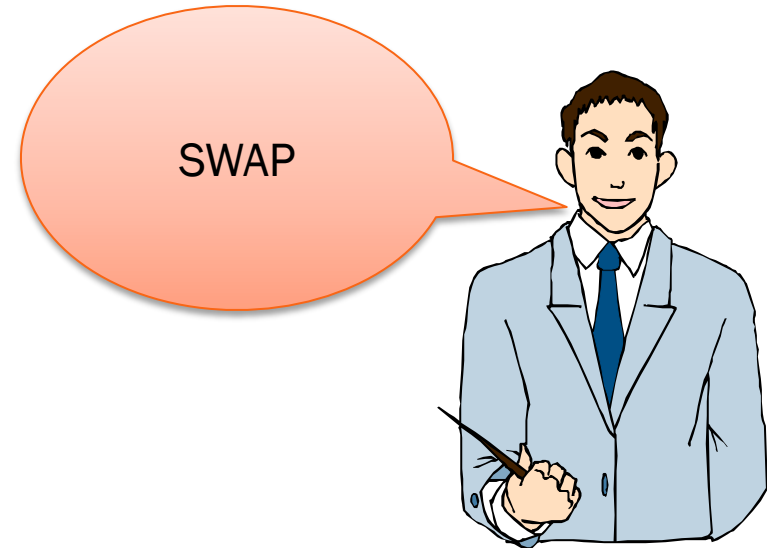
Index	value
0	1
1	6
2	4
3	8
4	12
5	10
6	7
7	5
8	17
9	21



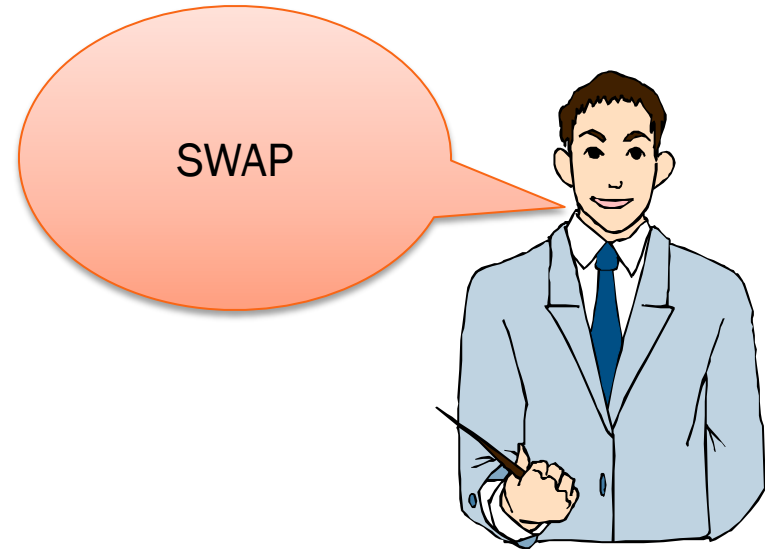
Index	value
0	1
1	6
2	4
3	8
4	10
5	12
6	7
7	5
8	17
9	21



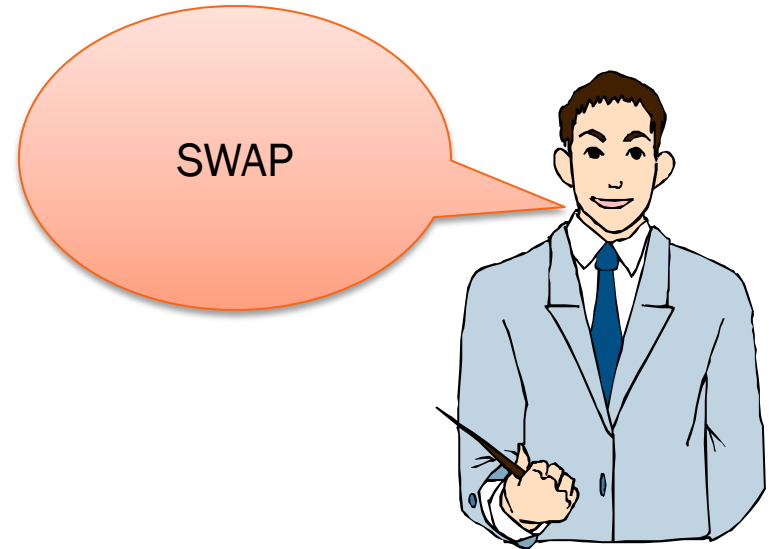
Index	value
0	1
1	6
2	4
3	8
4	10
5	12
6	7
7	5
8	17
9	21



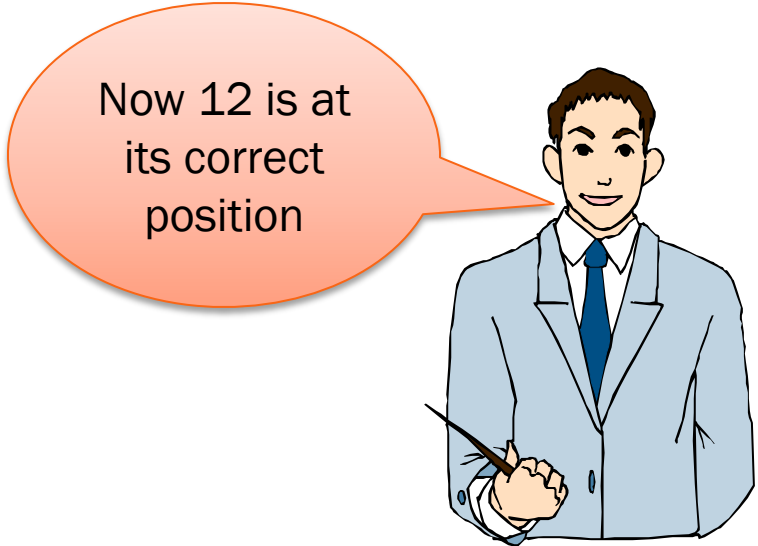
Index	value
0	1
1	6
2	4
3	8
4	10
5	7
6	12
7	5
8	17
9	21



Index	value
0	1
1	6
2	4
3	8
4	10
5	7
6	12
7	5
8	17
9	21

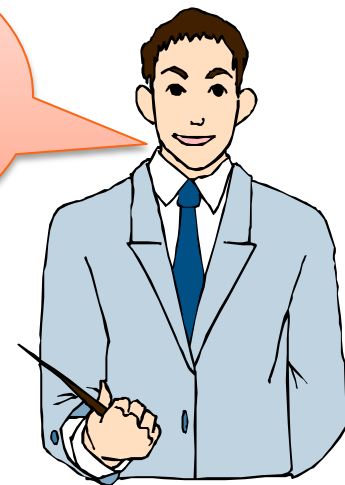


Index	value
0	1
1	6
2	4
3	8
4	10
5	7
6	5
7	12
8	17
9	21

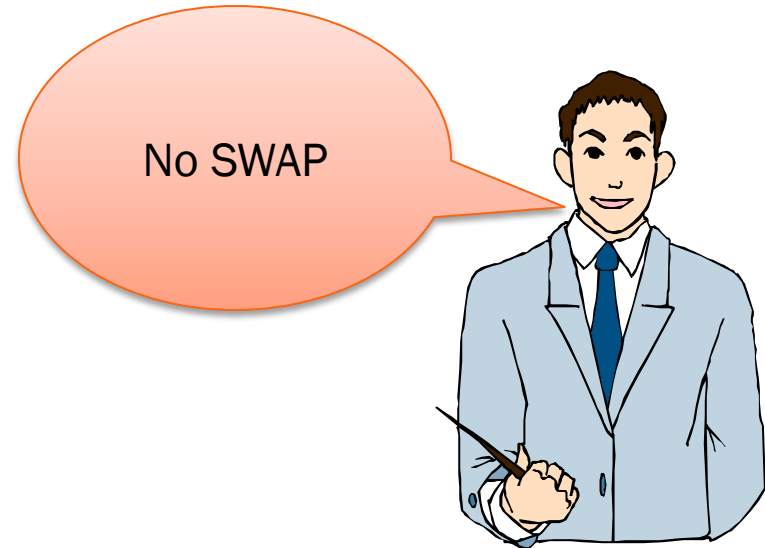


Index	value
0	1
1	6
2	4
3	8
4	10
5	7
6	5
7	12
8	17
9	21

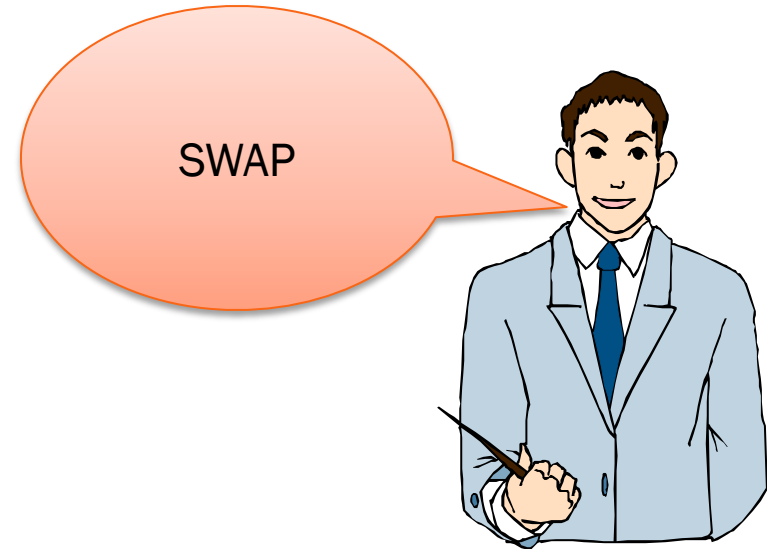
Now 12 is at
its correct
position



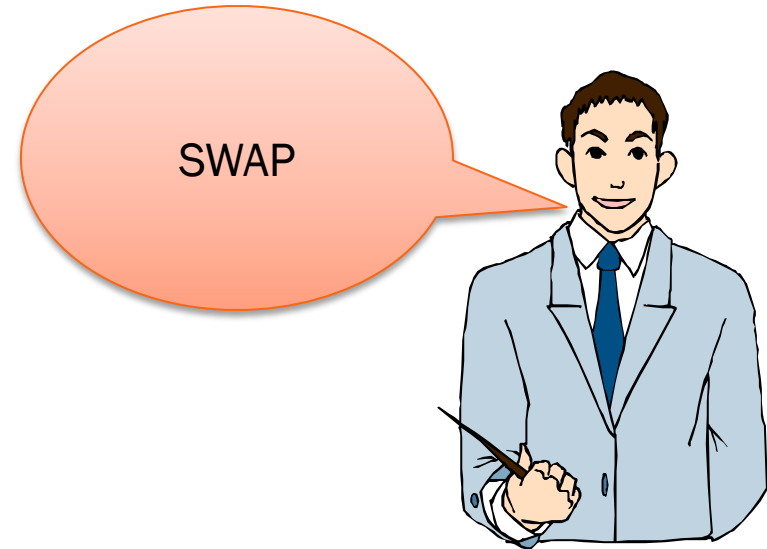
Index	value
0	1
1	6
2	4
3	8
4	10
5	7
6	5
7	12
8	17
9	21



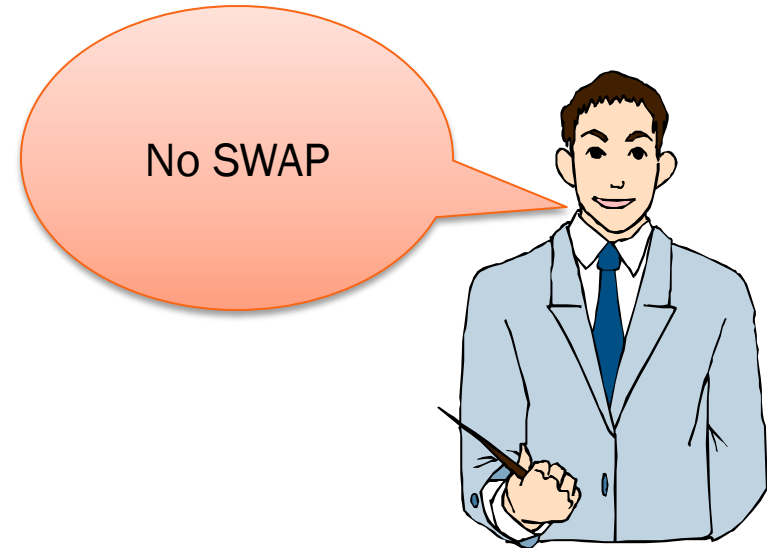
Index	value
0	1
1	6
2	4
3	8
4	10
5	7
6	5
7	12
8	17
9	21



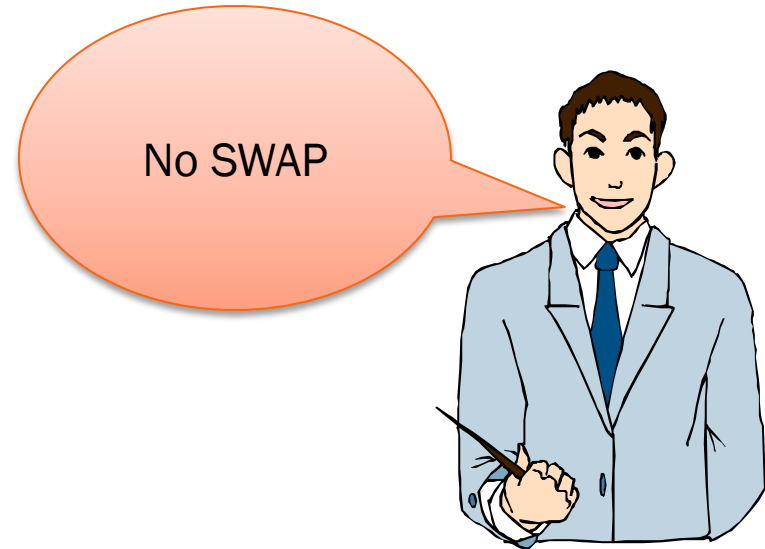
Index	value
0	1
1	4
2	6
3	8
4	10
5	7
6	5
7	12
8	17
9	21



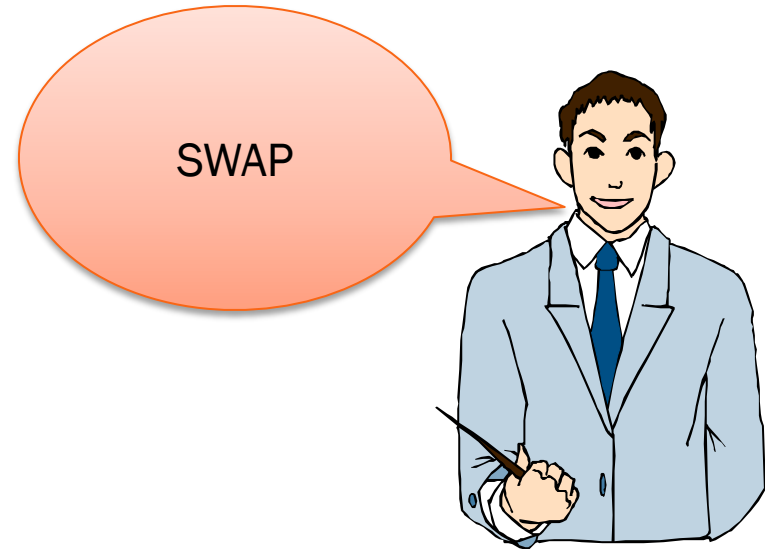
Index	value
0	1
1	4
2	6
3	8
4	10
5	7
6	5
7	12
8	17
9	21



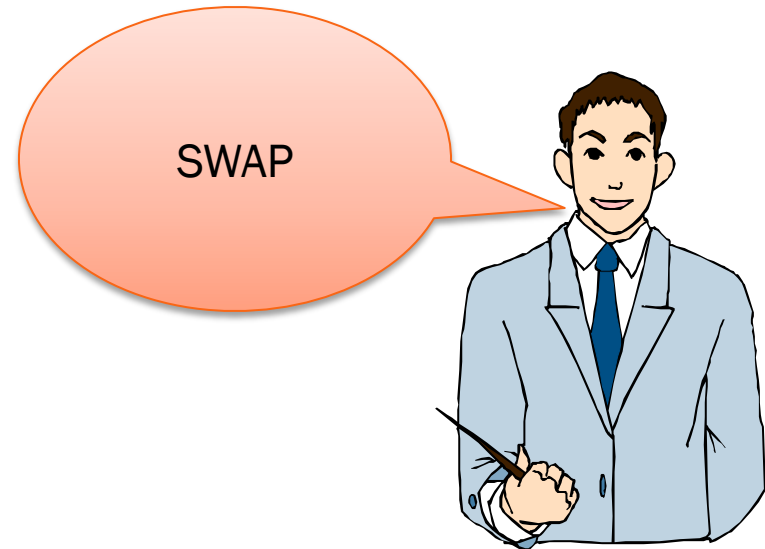
Index	value
0	1
1	4
2	6
3	8
4	10
5	7
6	5
7	12
8	17
9	21



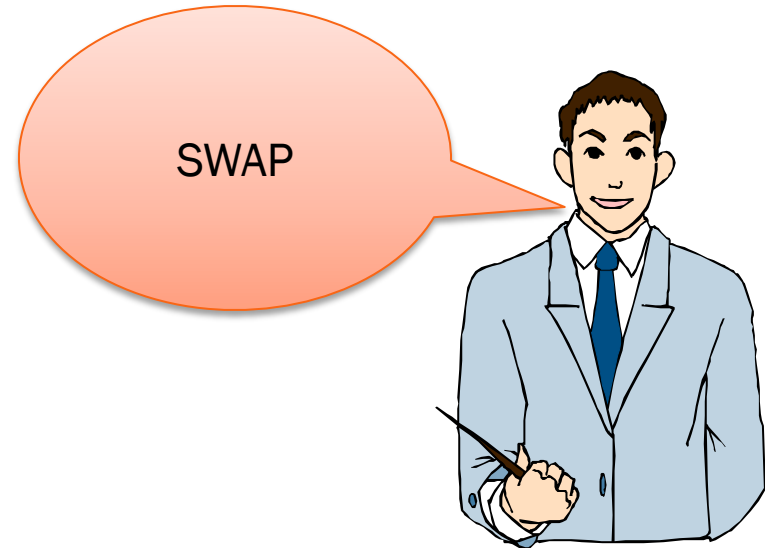
Index	value
0	1
1	4
2	6
3	8
4	10
5	7
6	5
7	12
8	17
9	21



Index	value
0	1
1	4
2	6
3	8
4	7
5	10
6	5
7	12
8	17
9	21

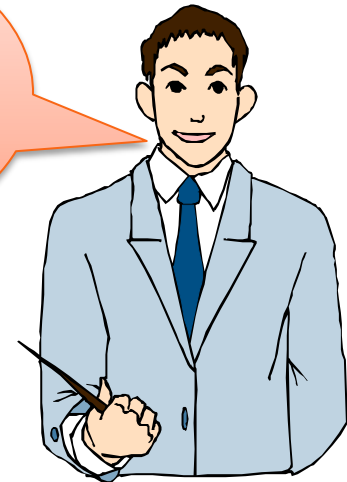


Index	value
0	1
1	4
2	6
3	8
4	7
5	10
6	5
7	12
8	17
9	21



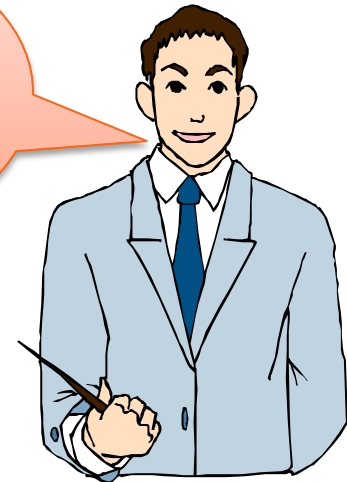
Index	value
0	1
1	4
2	6
3	8
4	7
5	5
6	10
7	12
8	17
9	21

No 10 is at its correct position

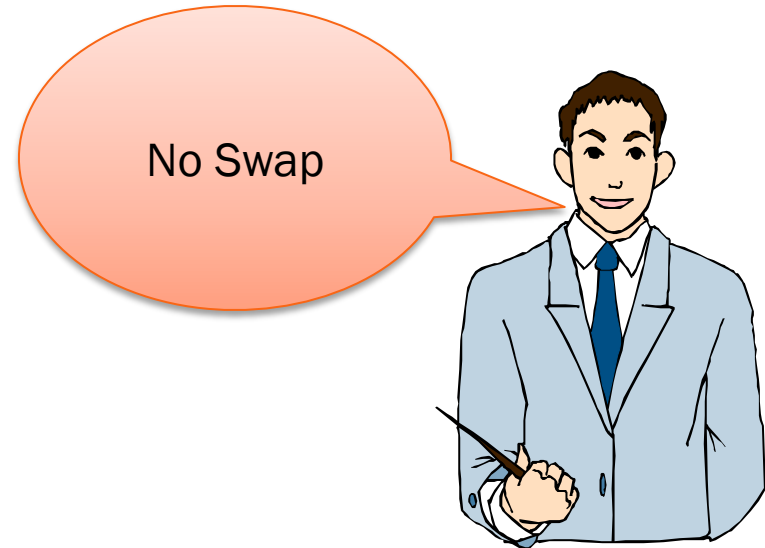


Index	value
0	1
1	4
2	6
3	8
4	7
5	5
6	10
7	12
8	17
9	21

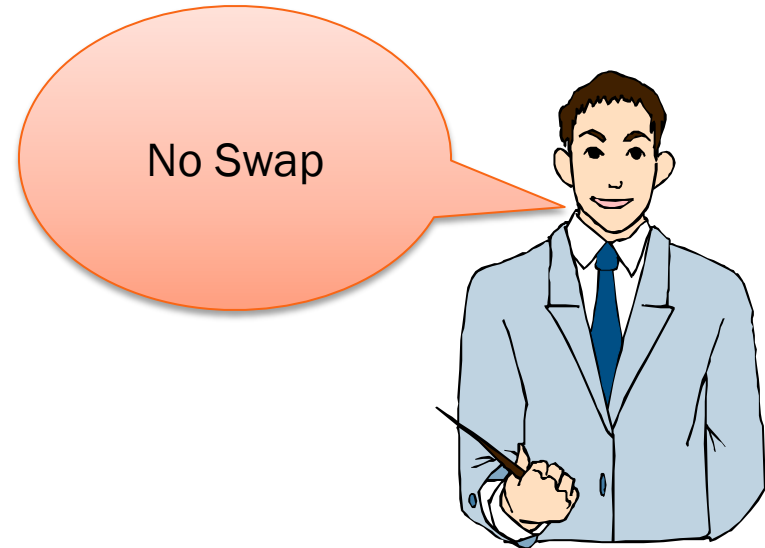
No 10 is at its correct position



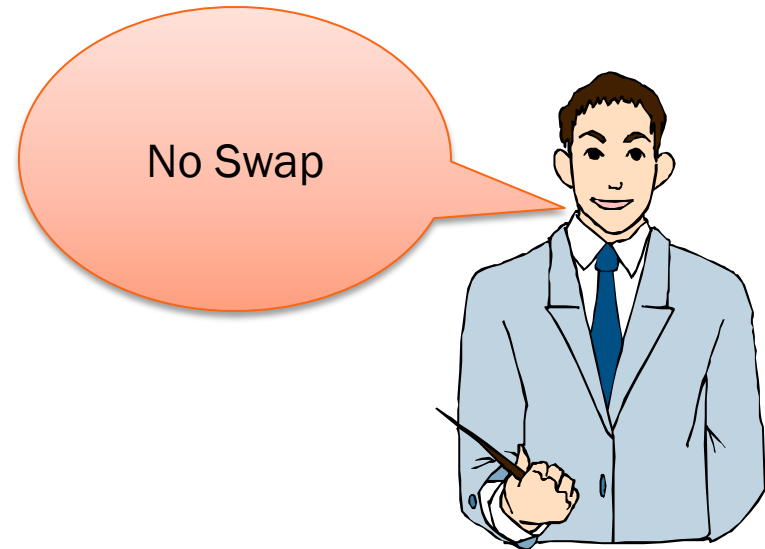
Index	value
0	1
1	4
2	6
3	8
4	7
5	5
6	10
7	12
8	17
9	21



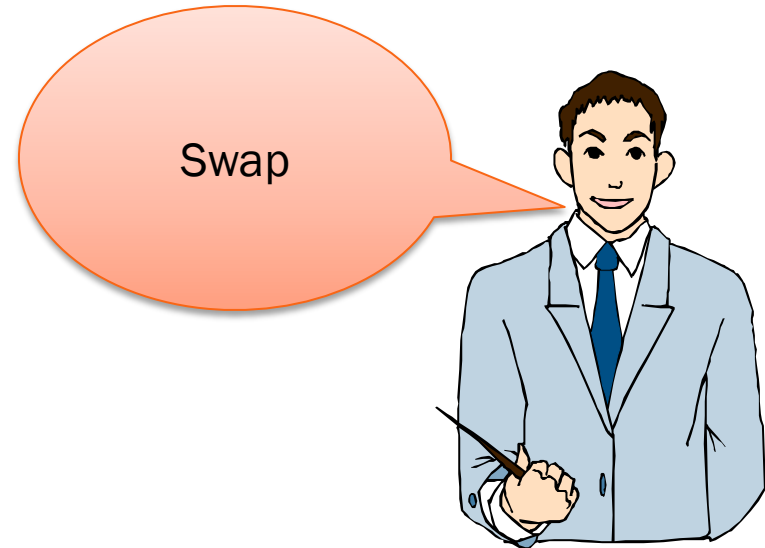
Index	value
0	1
1	4
2	6
3	8
4	7
5	5
6	10
7	12
8	17
9	21



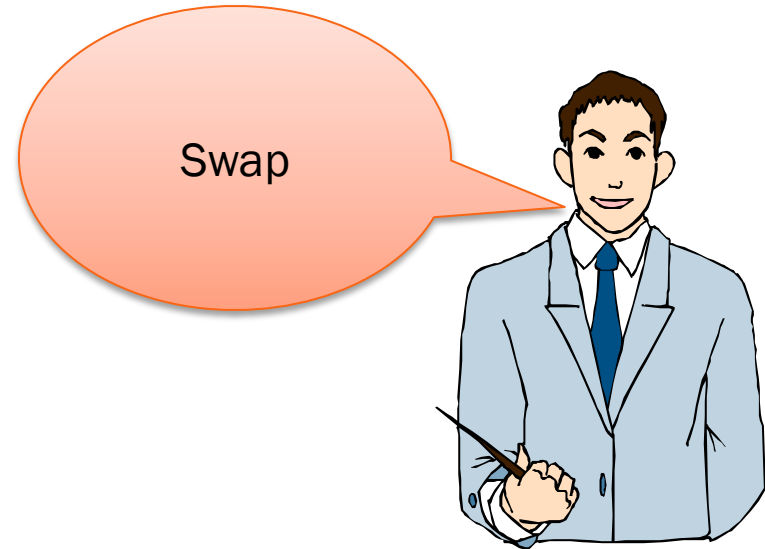
Index	value
0	1
1	4
2	6
3	8
4	7
5	5
6	10
7	12
8	17
9	21



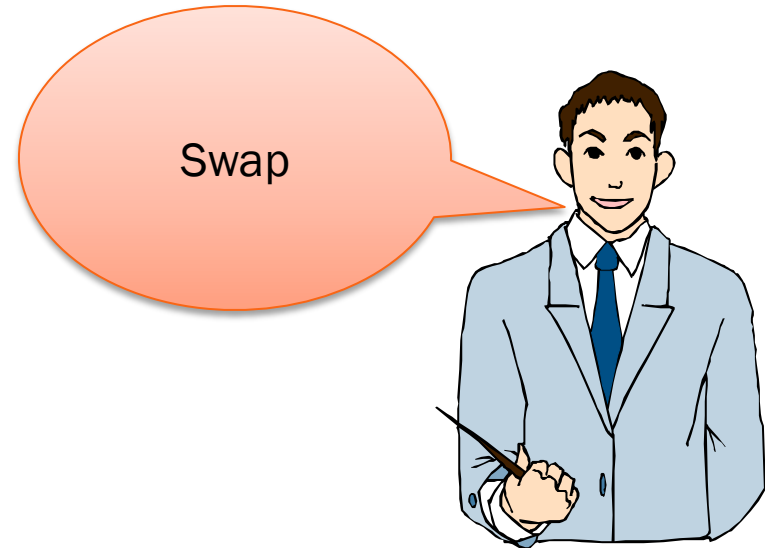
Index	value
0	1
1	4
2	6
3	8
4	7
5	5
6	10
7	12
8	17
9	21



Index	value
0	1
1	4
2	6
3	7
4	8
5	5
6	10
7	12
8	17
9	21

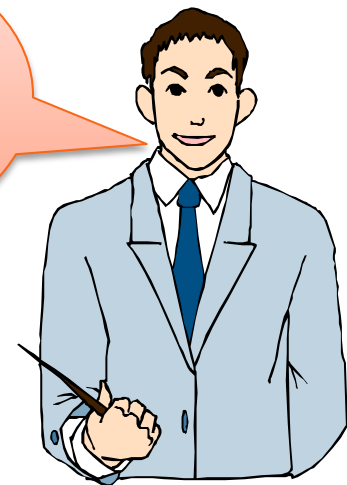


Index	value
0	1
1	4
2	6
3	7
4	8
5	5
6	10
7	12
8	17
9	21



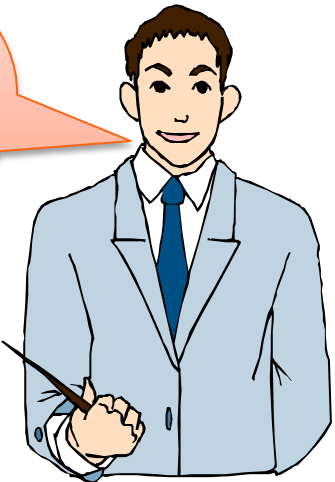
Index	value
0	1
1	4
2	6
3	7
4	5
5	8
6	10
7	12
8	17
9	21

Now 8 is at its correct position

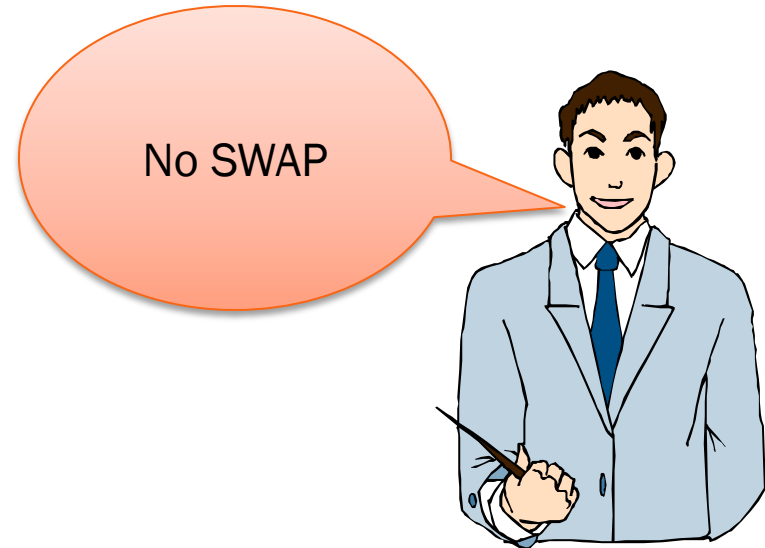


Index	value
0	1
1	4
2	6
3	7
4	5
5	8
6	10
7	12
8	17
9	21

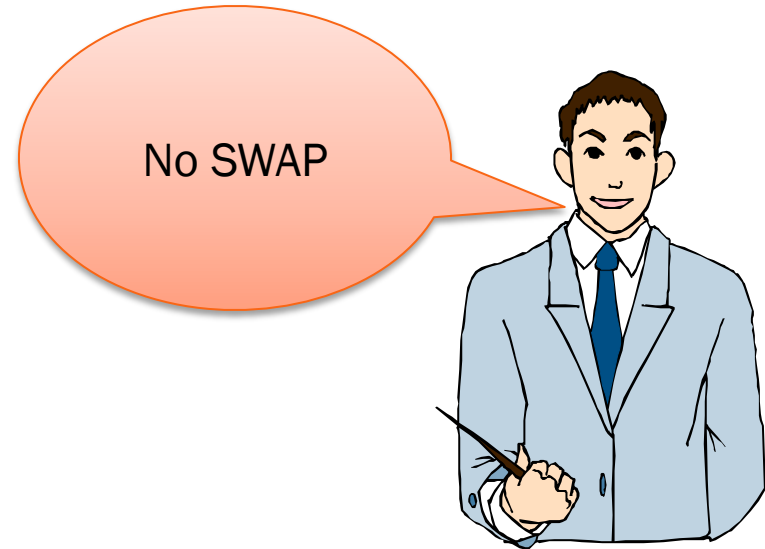
Now 8 is at its correct position



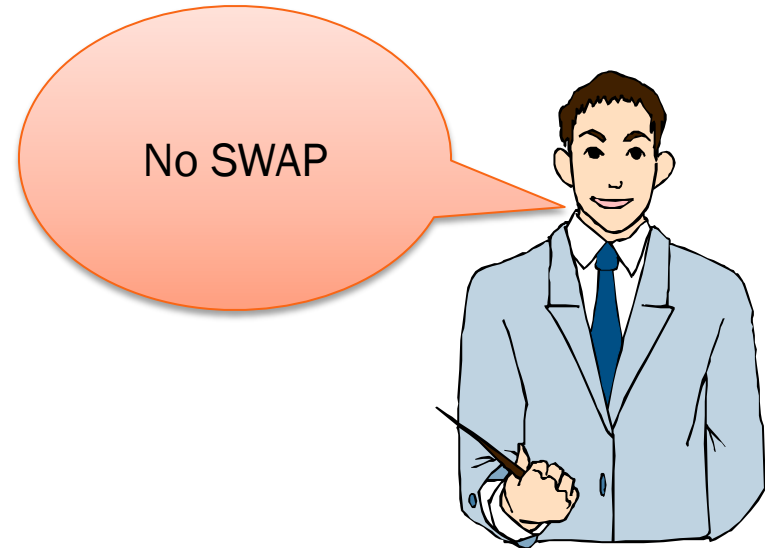
Index	value
0	1
1	4
2	6
3	7
4	5
5	8
6	10
7	12
8	17
9	21



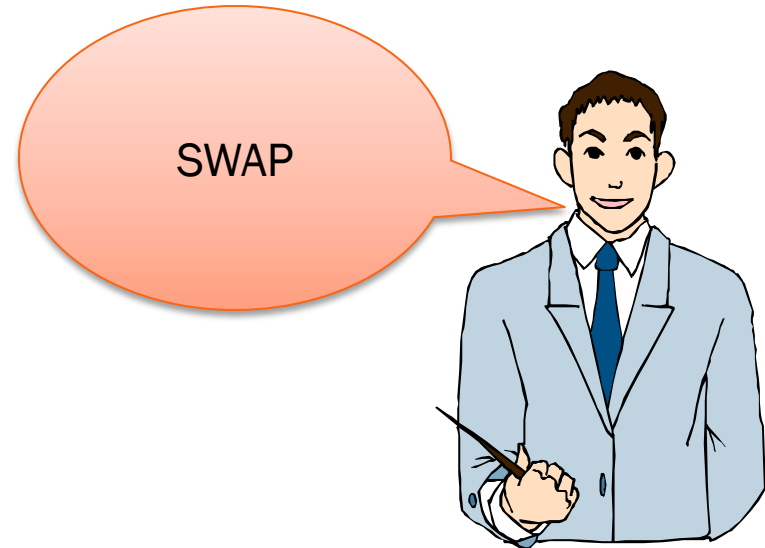
Index	value
0	1
1	4
2	6
3	7
4	5
5	8
6	10
7	12
8	17
9	21



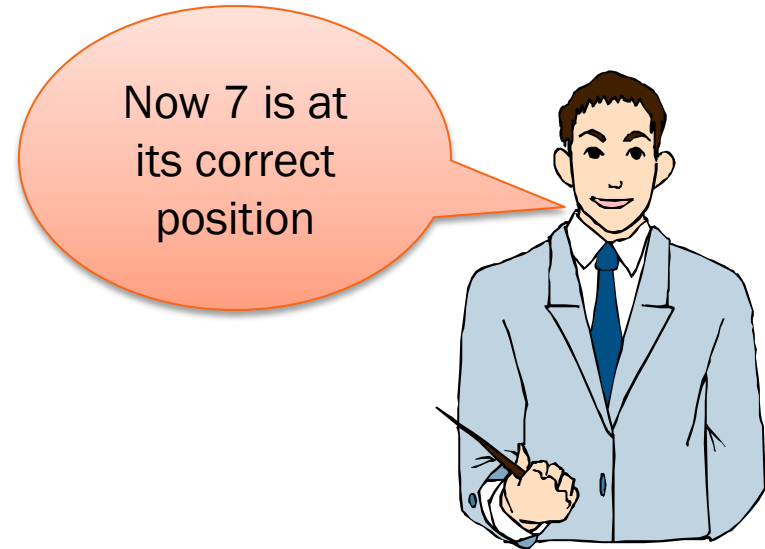
Index	value
0	1
1	4
2	6
3	7
4	5
5	8
6	10
7	12
8	17
9	21



Index	value
0	1
1	4
2	6
3	7
4	5
5	8
6	10
7	12
8	17
9	21

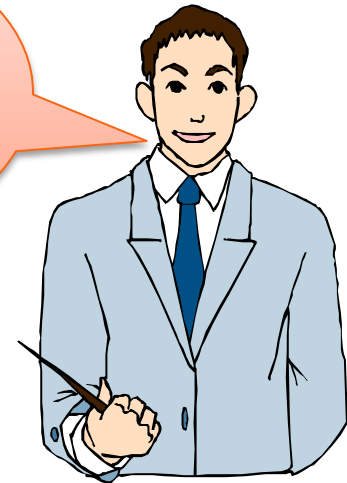


Index	value
0	1
1	4
2	6
3	5
4	7
5	8
6	10
7	12
8	17
9	21

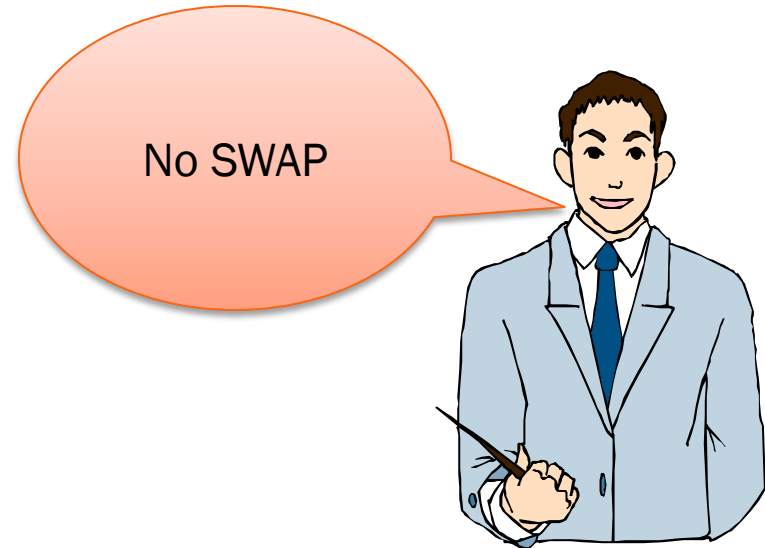


Index	value
0	1
1	4
2	6
3	5
4	7
5	8
6	10
7	12
8	17
9	21

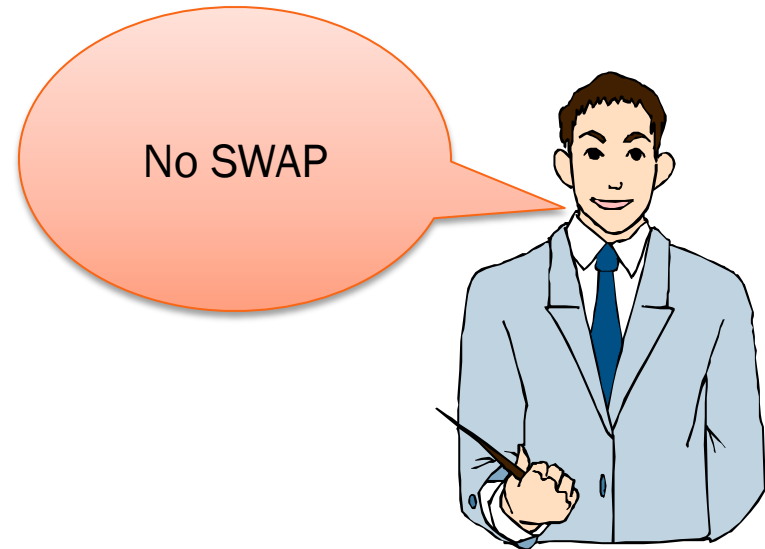
Now 7 is at its correct position



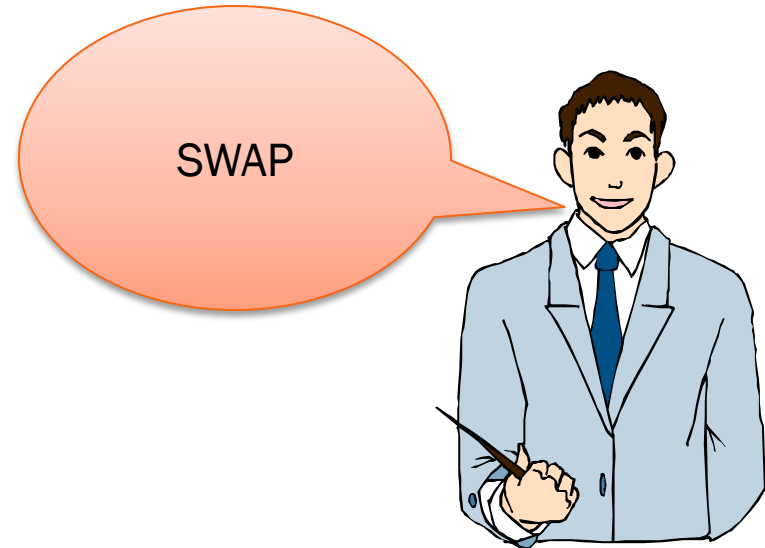
Index	value
0	1
1	4
2	6
3	5
4	7
5	8
6	10
7	12
8	17
9	21



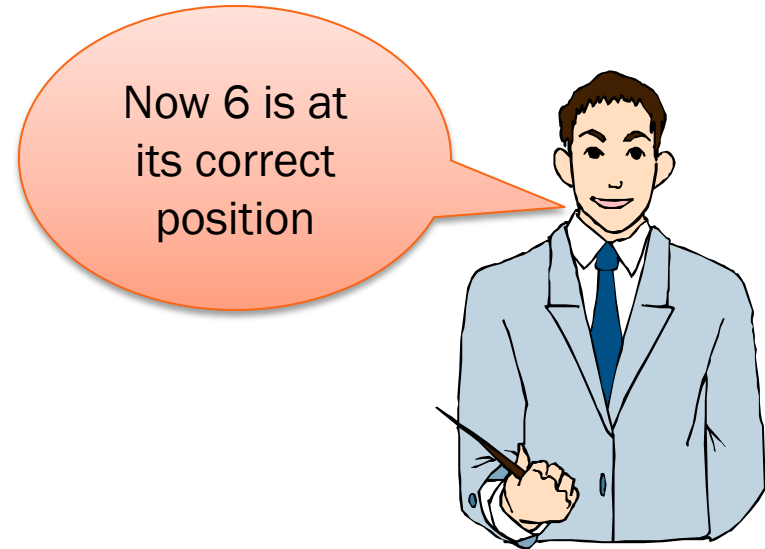
Index	value
0	1
1	4
2	6
3	5
4	7
5	8
6	10
7	12
8	17
9	21



Index	value
0	1
1	4
2	6
3	5
4	7
5	8
6	10
7	12
8	17
9	21

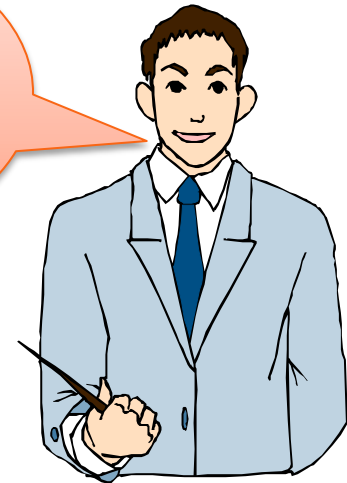


Index	value
0	1
1	4
2	5
3	6
4	7
5	8
6	10
7	12
8	17
9	21

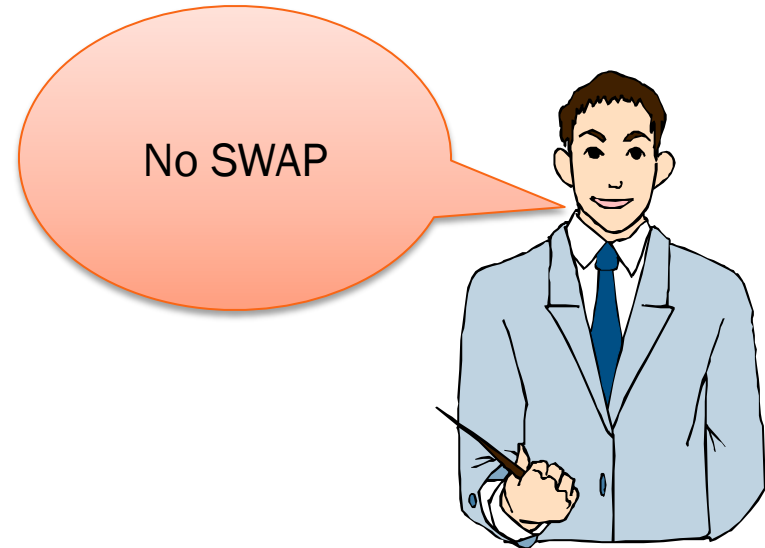


Index	value
0	1
1	4
2	5
3	6
4	7
5	8
6	10
7	12
8	17
9	21

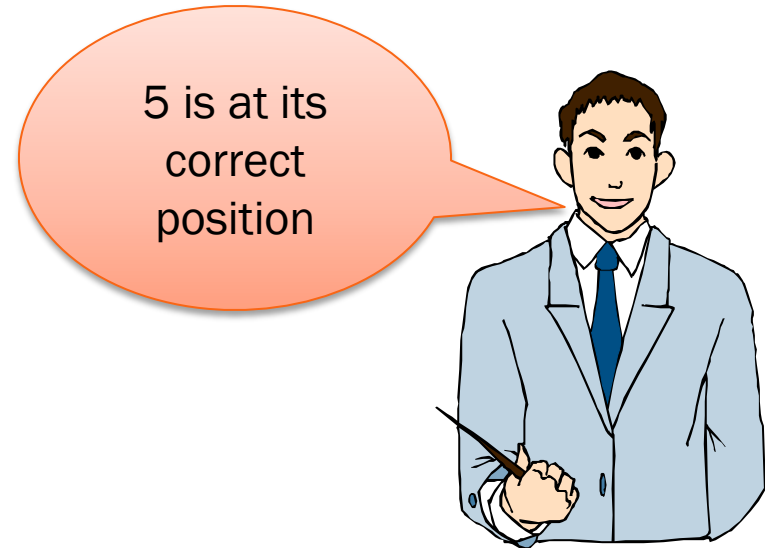
Now 6 is at its correct position



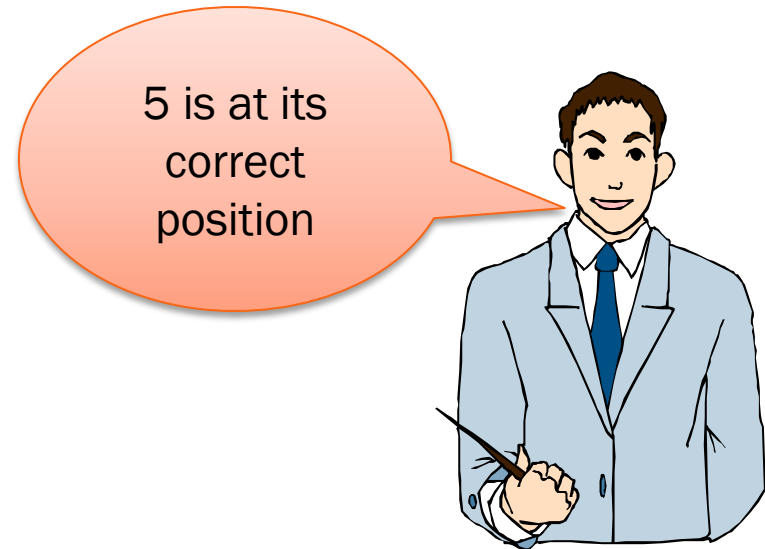
Index	value
0	1
1	4
2	5
3	6
4	7
5	8
6	10
7	12
8	17
9	21



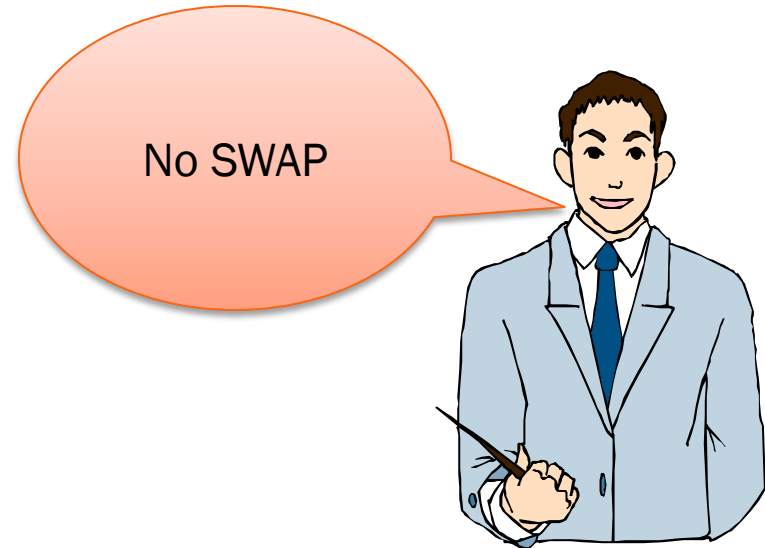
Index	value
0	1
1	4
2	5
3	6
4	7
5	8
6	10
7	12
8	17
9	21



Index	value
0	1
1	4
2	5
3	6
4	7
5	8
6	10
7	12
8	17
9	21

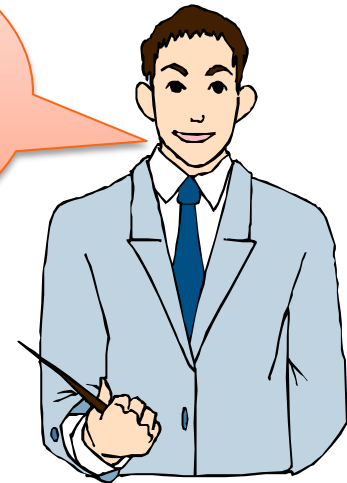


Index	value
0	1
1	4
2	5
3	6
4	7
5	8
6	10
7	12
8	17
9	21



Index	value
0	1
1	4
2	5
3	6
4	7
5	8
6	10
7	12
8	17
9	21

Now all the elements are in sorted order



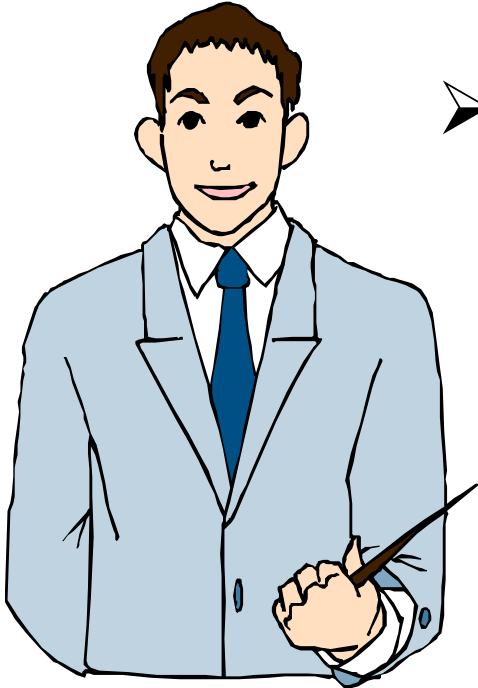
Hope you must have
understood how
Bubble Sorting
works !!



How to write Algorithm & Python function for Bubble Sorting ?



- In Bubble Sorting Nested Loop is used
- Outer loop is used for PASS i.e. **0 to $<N-1$**
- Inner loop is used for Comparisons i.e. from **0 to $<N-1-i$**




ALGORITHM

```
for i = L to U
  for j = L to (U-1)-i
    if A[j]>A[j+1]
      temp=A[j]
      A[j]=A[j+1]
      A[j+1]=temp
```

Bubble Sort (Ascending order)

```
aList = [21,8,15,27,2,80,1]
print("Original list is ", aList)
n = len(aList)
for i in range(n):
    print("Pass ", i+1)
    for j in range(n-1-i):
        if aList[j]>aList[j+1]:
            aList[j],aList[j+1] = aList[j+1],aList[j]
    print("After pass ",i+1," List is :",aList)
print("### SORTED LIST ###")
print("Sorted List is : ",aList)
```

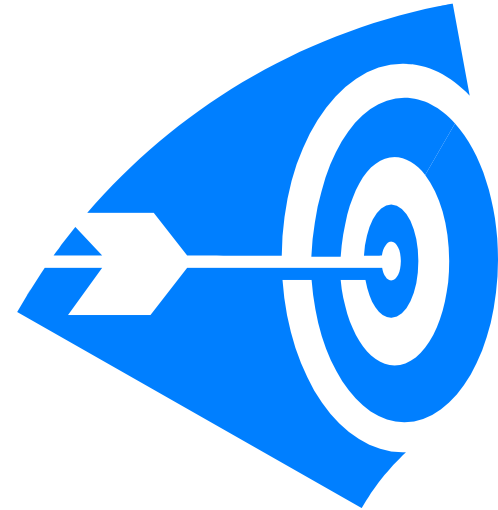


Output

```
Original list is [21, 8, 15, 27, 2, 80, 1]
Pass 1
After pass 1 List is : [8, 15, 21, 2, 27, 1, 80]
Pass 2
After pass 2 List is : [8, 15, 2, 21, 1, 27, 80]
Pass 3
After pass 3 List is : [8, 2, 15, 1, 21, 27, 80]
Pass 4
After pass 4 List is : [2, 8, 1, 15, 21, 27, 80]
Pass 5
After pass 5 List is : [2, 1, 8, 15, 21, 27, 80]
Pass 6
After pass 6 List is : [1, 2, 8, 15, 21, 27, 80]
Pass 7
After pass 7 List is : [1, 2, 8, 15, 21, 27, 80]
Sorted List is [1, 2, 8, 15, 21, 27, 80]
```


Feel free to contact :

Mr. Vinod Kumar Verma
P.G.T(Computer Science)
Kendriya Vidyalaya OEF Kanpur
Email-vinodexclusively@gmail.com



Thank You