SCJ2013 Data Structure & Algorithms

# Bubble Sort

Nor Bahiah Hj Ahmad & Dayang
Norhayati A. Jawawi

# Bubble Sort

Sorting activities for Bubble:

- Go through multiple passes over the array.
- In every pass:
  - Compare adjacent elements in the list.
  - Exchange the elements if they are out of order.
  - Each pass moves the largest (or smallest) elements to the end of the array
- Repeating this process in several passes eventually sorts the array into ascending (or descending) order.
- Bubble sort complexity is is $O(n^2)$ and only suitable to sort array with small size of data.

# Bubble Sort Implementation

```
// Sorts items in an array into ascending order.
void BubbleSort(dataType data[], int listSize)
{   int pass, tempValue;
   for ( pass =1;pass < listSize; pass++ )
   {
    // moves the largest element to the
    // end of the array
       for (int x = 0; x < listSize - pass; x++)
            //compare adjacent elements
           if ( data[x]>data[x+1] )
            {   // swap elements
                tempValue = data[x];
               data[x] = data[x+1];
               data[x+1] = tempValue;
            }
     }
} // end Bubble Sort
```

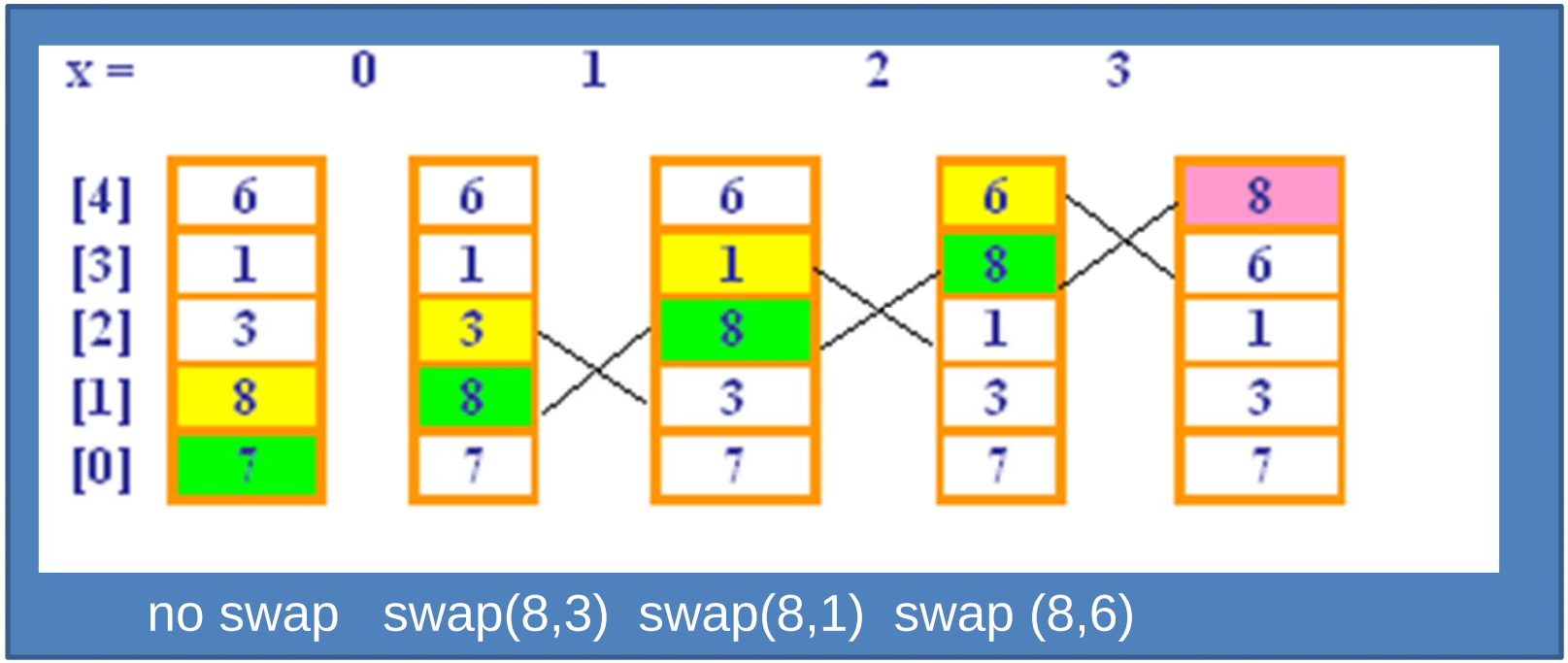
**External for loop is used to control the number of passes needed.**

**Internal for loop** is used to compare adjacent elements and swap elements if they are not in order. After the internal loop has finished execution, the largest element in the array will be moved at the top.

**if statement** is used to compare the adjacent elements.

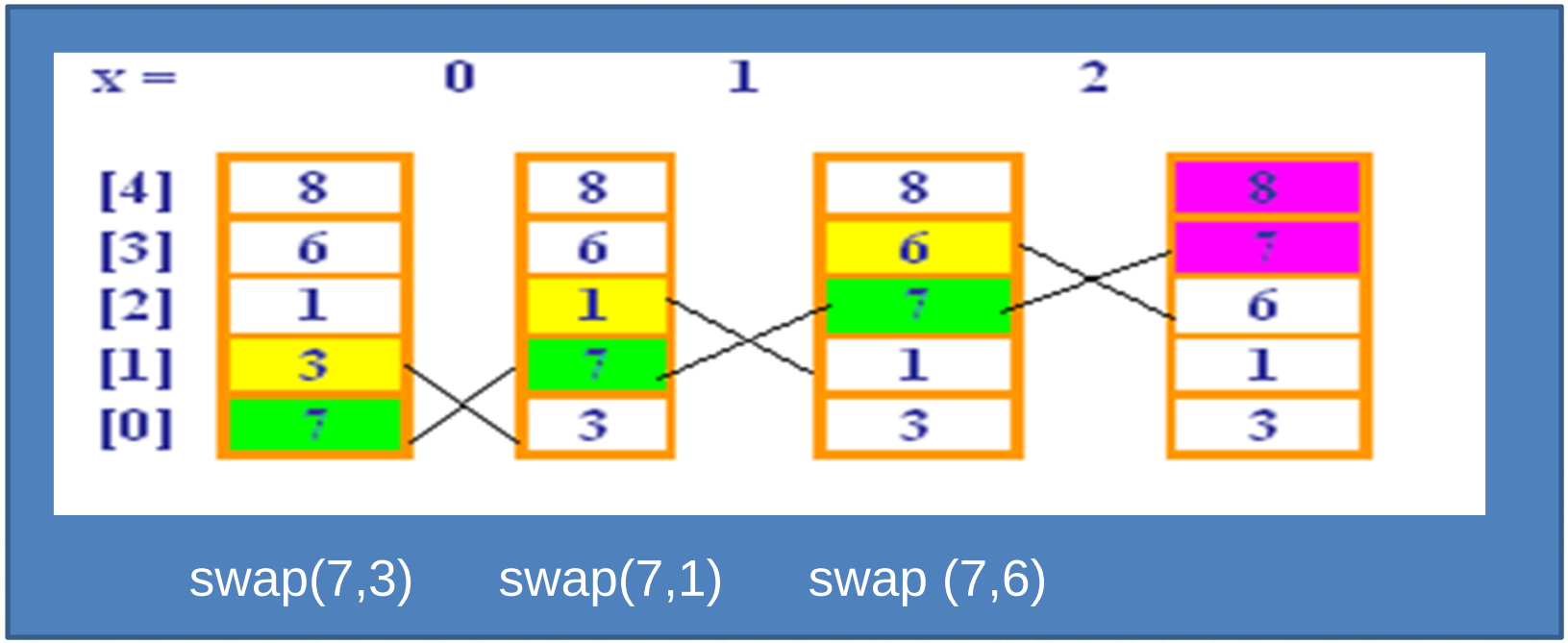
# Sort [7 8 3 1 6] into Ascending Order

pass = 1 List Size = 5



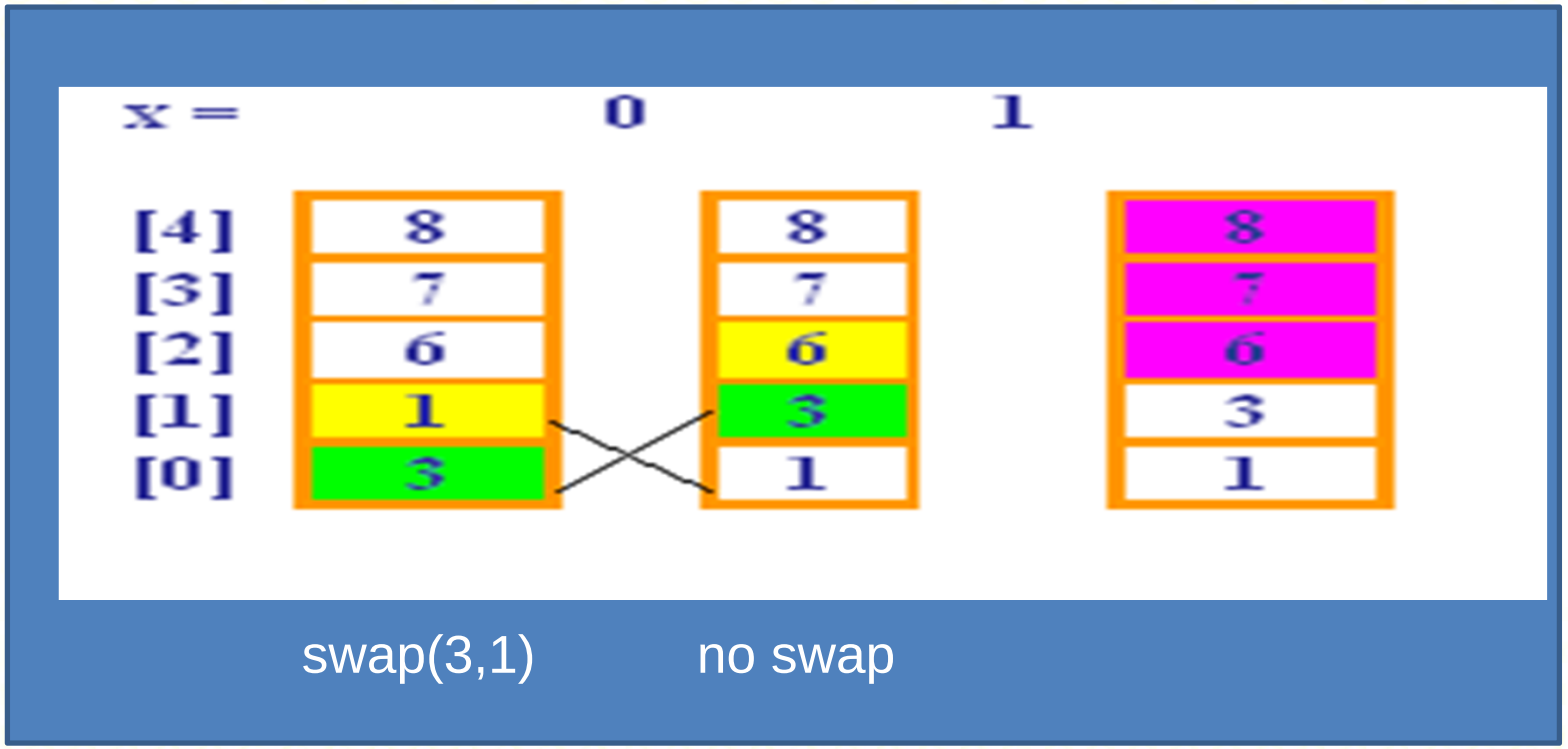
# Sort [7 8 3 1 6] into Ascending Order

pass = 2 listSize = 5



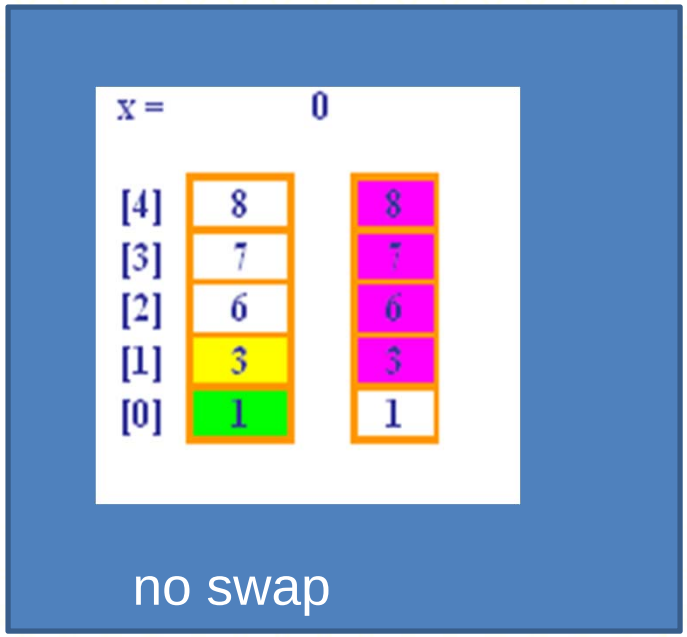
# Sort [7 8 3 1 6] into Ascending Order

pass= 3



# Sort [7 8 3 1 6] into Ascending Order

pass = 4



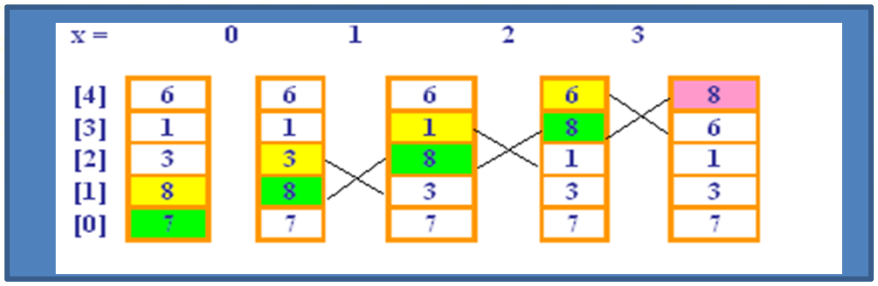
# Bubble Sort Analysis

- The number of comparison between elements and the number of exchange between elements determine the efficiency of Bubble Sort algorithm.

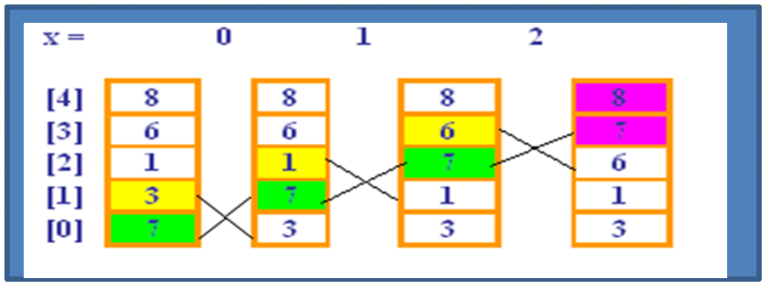
- Generally, the number of comparisons between elements in Bubble Sort can be stated as follows:

$$(n-1)+(n-2)+\ldots\ldots+2+1 = n(n-1)/2 = O(n^2)$$

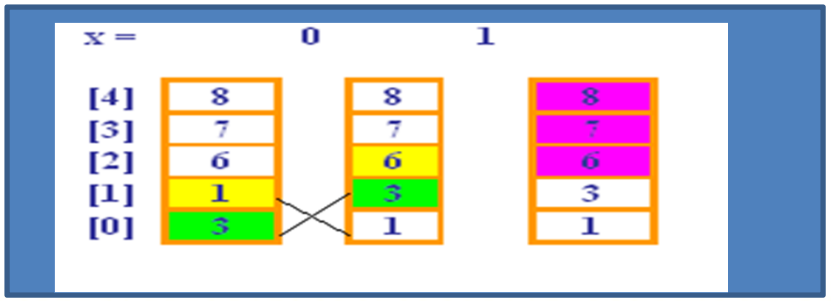
# Bubble Sort Analysis [7 8 3 1 6]



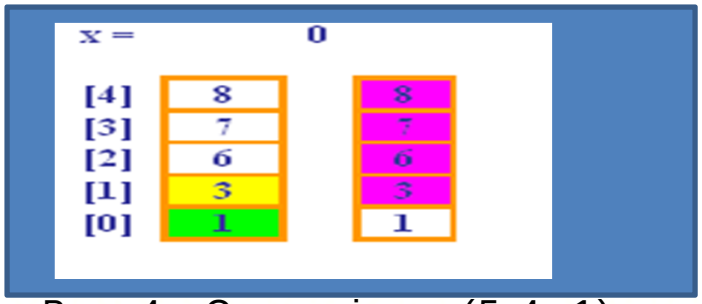

Pass 1 : Comparison (listSize-pass=4)




Pass 2 : Comparisons (listSize-pass:(5-2=3))




Pass 3 : Comparison (5-3= 2)



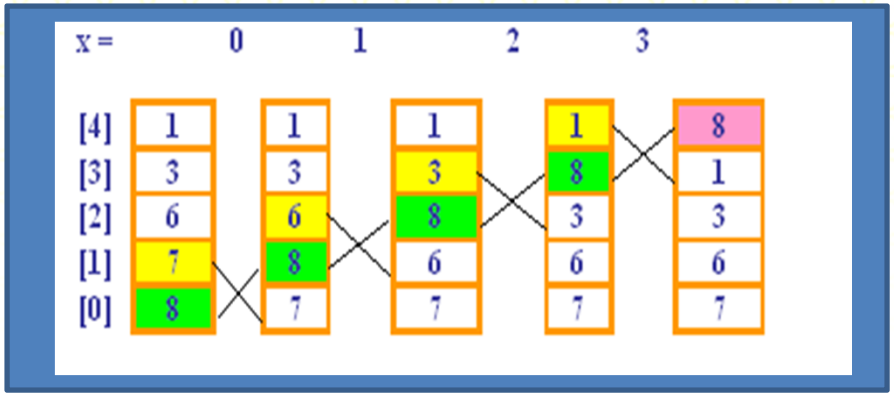

Pass 4 : Comparisons (5-4=1)

$(n-1)+(n-2)+\ldots\ldots\ldots\ldots+2+1 = n(n-1)/2 = O(n^2)$

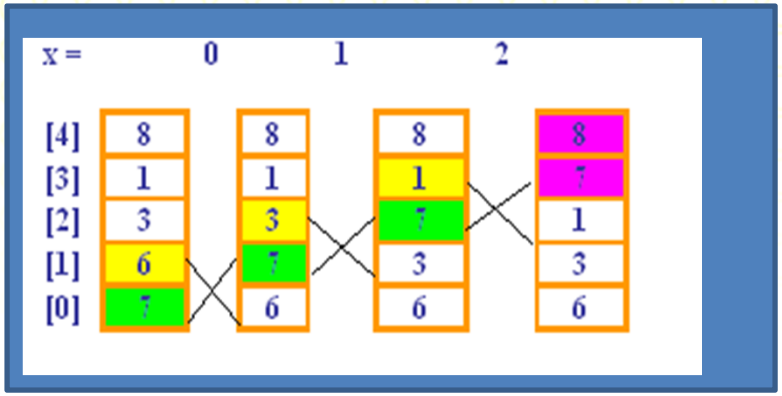
**The number of comparisons:**

**(5-1) + (5-2) + (5-3) + (5-4) = 4 + 3 + 2 + 1 = 10.**

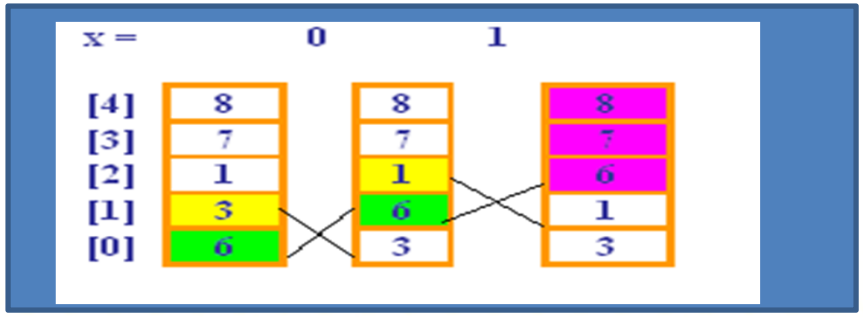
# Worse Case Analysis [8 7 6 3 1]



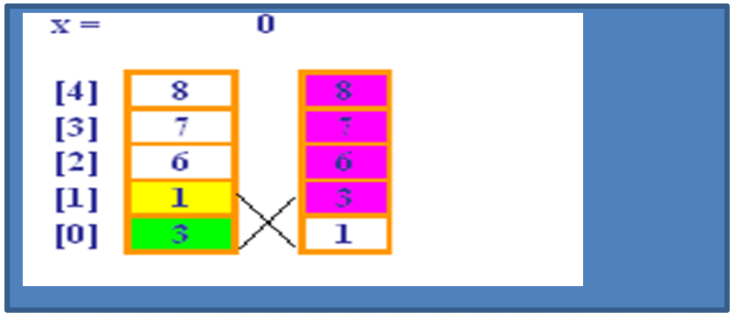

Pass 1 : Comparison (5-1=4)




Pass 2 : Comparisons (5-2=3)



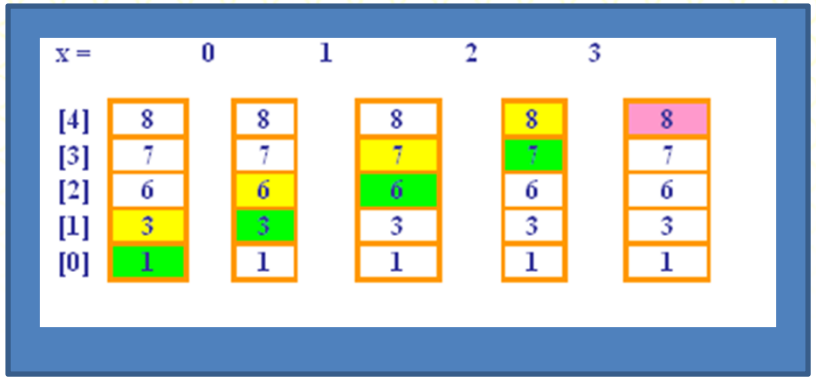

Pass 3 : Comparison (5-3= 2)



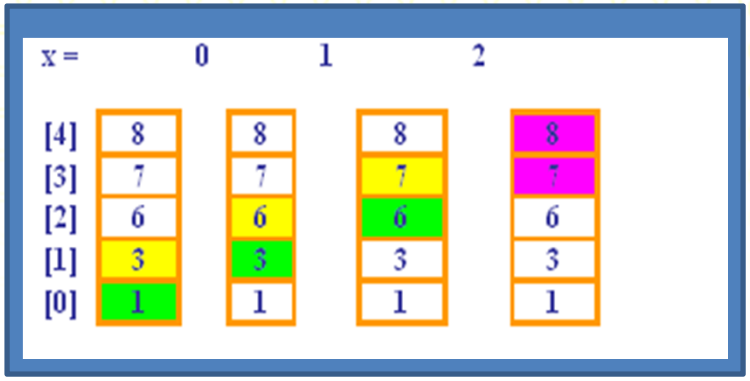

Pass 4 : Comparisons (5-4=1)

**The number of comparisons to sort data in this list: (5-1) + (5-2) + (5-3) + (5-4) = 4 + 3 + 2 + 1 = 10.**

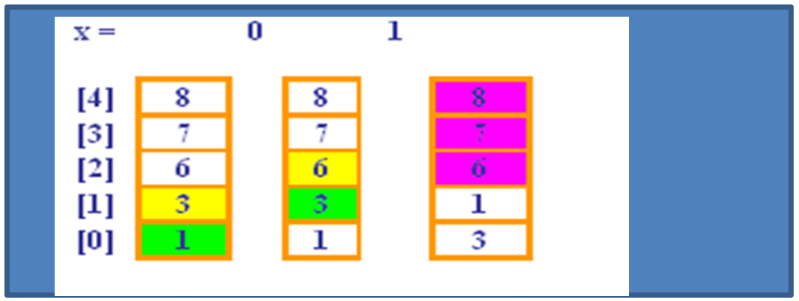
# Best Case Analysis [1 3 6 7 8]



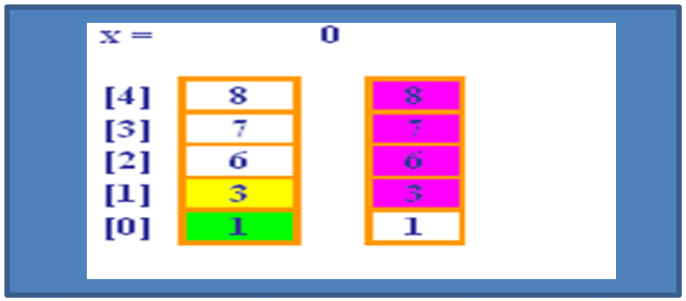

Pass 1 : Comparison (5-1=4)




Pass 2 : Comparisons (5-2=3)




Pass 3 : Comparison (5-3= 2)




Pass 4 : Comparisons (5-4=1)

**The number of comparisons to sort data in this list:**
**(5-1) + (5-2) + (5-3) + (5-4) = 4 + 3 + 2 + 1 = 10.**

# Bubble Sort Analysis

In any cases, (worse case, best case or average case) to sort the list in ascending order the number of comparisons between elements is the same.

- Worse Case [8 7 6 3 1]
- Average Case [7 8 3 1 6]
- Best Case [1 3 6 7 8]

All lists with 5 elements need 10 comparisons to sort all the data.

# Bubble Sort Analysis

- In the example given, it can be seen that the number of comparison for worse case and best case is the same - with 10 comparisons.
- The difference can be seen in the number of swapping elements. Worse case has maximum number of swapping: 10, while best case has no swapping since all data is already in the right position.
- For best case, starting with pass one, there is no exchange of data occur.
- From the example, it can be concluded that in any pass, if there is no exchange of data occur, the list is already sorted. The next pass shouldn't be continued and the sorting process should stop.

# Improved Bubble Sort

```
// Improved Bubble Sort
// Sorts items in an array into ascending order.

void bubbleSort(DataType data[], int n)
{  int temp;
   bool sorted = false;  // false when swaps occur
   for (int pass = 1; (pass < n) && !sorted; ++pass)
   {  sorted = true;  // assume sorted
      for (int x = 0; x < n-pass; ++x)
      {  if (data[x] > data[x+1])
         {  // exchange items
               temp = data[x];
               data[x] = data[x+1];
               data[x+1] = temp;
               sorted = false;  // signal exchange
         } // end if
      } // end for
   } // end for
} // end bubbleSort
```

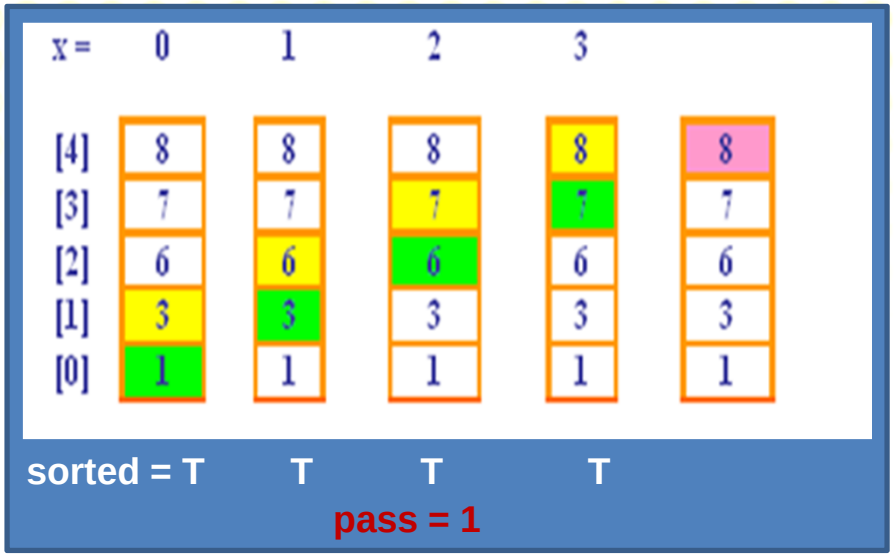
To improve the efficiency of Bubble Sort, a condition that check whether the list is sorted should be add at the external loop

A Boolean variable, **sorted** is added in the algorithm to signal whether there is any exchange of elements occur in certain pass.

In external loop, **sorted** is set to **true**. If there is exchange of data inside the inner loop, sorted is set to **false**.

Another pass will continue, if sorted is false and will stop if sorted is true.

# Improved Bubble Sort : Best Case[1 3 6 7 8]

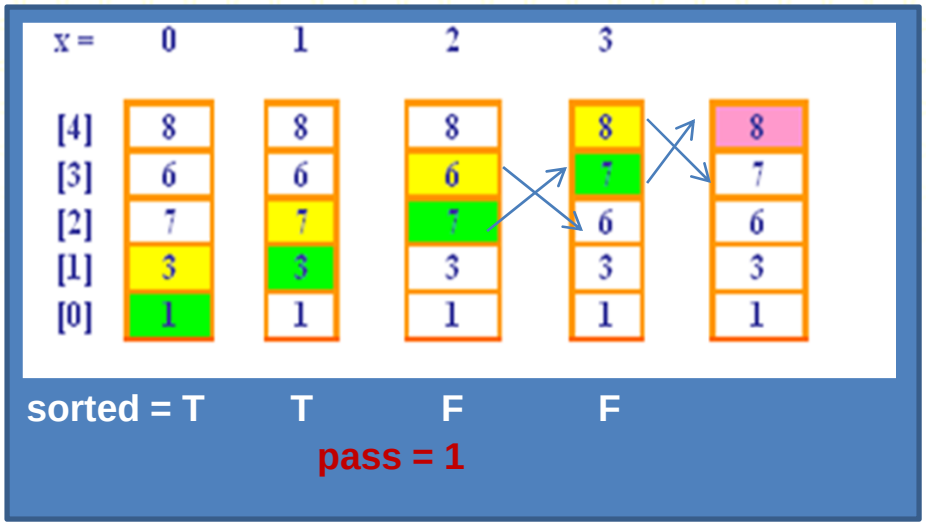


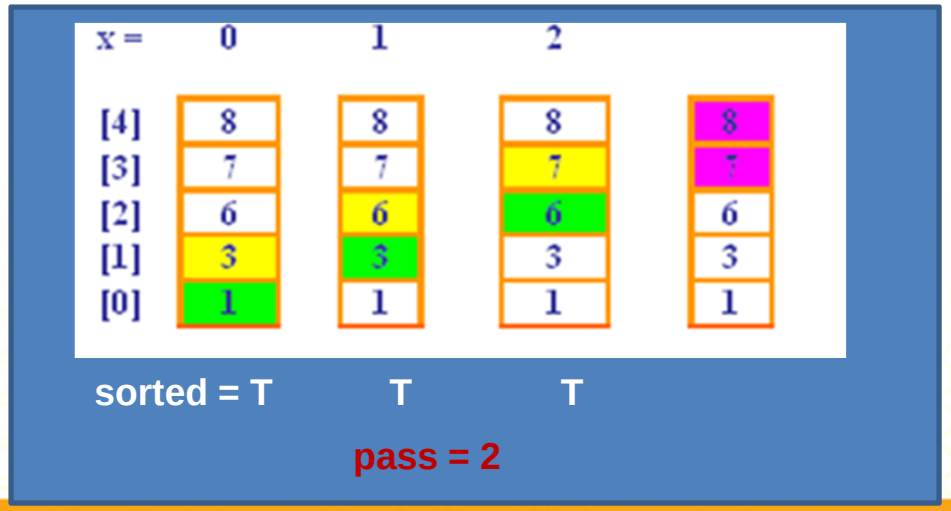

In pass 1, there is no exchange of data occur and variable sorted is always True.

Therefore, condition statement in external loop will become false and the loop will stop execution. In this example, pass 2 will not be continued.

Analysis - For best case, the number of comparison between elements is 4, (n-1) which is O(n).

# Improved Bubble Sort : Average Case [1 3 7 6 8]




For Average Case:

- [1 3 7 6 8] we have to go through 2 passes only. The subsequent passes is not continued since the array is already sorted.

- Conclusion – For improved Bubble Sort, the sorting time and the number of comparisons between data in average case and best case can be minimized.

# Bubble Sort – Algorithm Complexity

- Complexity is measured based on time consuming operations to compare and swap elements.
- Number of comparisons
  - a *for* loop embedded inside a *while* loop
  - Worst Case (n-1)+(n-2)+(n-3) …+1 , or $O(n^2)$
  - Best Case – (n-1) or O(n)
- Number of Swaps
  - inside a conditional -> #swaps data dependent !!
  - Best Case 0, or O(1)
  - Worst Case (n-1)+(n-2)+(n-3) …+1 , or $O(n^2)$

# Summary and Conclusion

Bubble Sort takes several passes to sort elements in an array. Every pass need to do comparisons between elements and exchange the data if the elements are not in the right order. However the complexity of Bubble sort is the same for best case and worse case.

| Bubble Sort | Comparisons | Swaps |
|---|---|---|
| Best Case | $O(n^2)$ | 0 |
| Average Case | $O(n^2)$ | $O(n^2)$ |
| Worst Case | $O(n^2)$ | $O(n^2)$ |