Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:
First Pass:
(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4
(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2
(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.
Second Pass:
(**1 4** 2 5 8) –> (**1 4** 2 5 8)
(1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2
(1 2 **4 5** 8) –> (1 2 **4 5** 8)
(1 2 4 **5 8**) –> (1 2 4 **5 8**)
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.
Third Pass:
(**1 2** 4 5 8) –> (**1 2** 4 5 8)
(1 **2 4** 5 8) –> (1 **2 4** 5 8)
(1 2 **4 5** 8) –> (1 2 **4 5** 8)
(1 2 4 **5 8**) –> (1 2 4 **5 8**)

```
// C++ program for implementation of Bubble sort

#include <bits/stdc++.h>

using namespace std;

  void swap(int *xp, int *yp)

{

    int temp = *xp;
```

```
    *xp = *yp;
    *yp = temp;
}
// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)
      // Last i elements are already in place
    for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
}
  /* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
```

```
}

  // Driver code

int main()

{

    int arr[] = {64, 34, 25, 12, 22, 11, 90};

    int n = sizeof(arr)/sizeof(arr[0]);

    bubbleSort(arr, n);

    cout<<"Sorted array: \n";

    printArray(arr, n);

    return 0;

}

  //
```

Output:

Sorted array:

11 12 22 25 34 64 90

<!—-Illustration :

i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i = 1	0	3	1	5	8	2	4	7	**9**
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	**8**	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	**7**		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	**5**			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	**4**				
	1	1	2	3					
i = 6	0	1	2	**3**					
		1	**2**						

—>

Optimized Implementation:

The above function always runs O(n^2) time even if the array is sorted. It can be optimized by stopping the algorithm if inner loop didn't cause any swap.

```
// Optimized implementation of Bubble sort

#include <stdio.h>

  void swap(int *xp, int *yp)

{

    int temp = *xp;

    *xp = *yp;
```

```
    *yp = temp;
}
  // An optimized version of Bubble Sort
void bubbleSort(int arr[], int n)
{
   int i, j;
   bool swapped;
   for (i = 0; i < n-1; i++)
   {
     swapped = false;
     for (j = 0; j < n-i-1; j++)
     {
        if (arr[j] > arr[j+1])
        {
           swap(&arr[j], &arr[j+1]);
           swapped = true;
        }
     }
       // IF no two elements were swapped by inner loop, then break
     if (swapped == false)
```

```
        break;
    }
}
  /* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("n");
}
  // Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
```

```
}
```

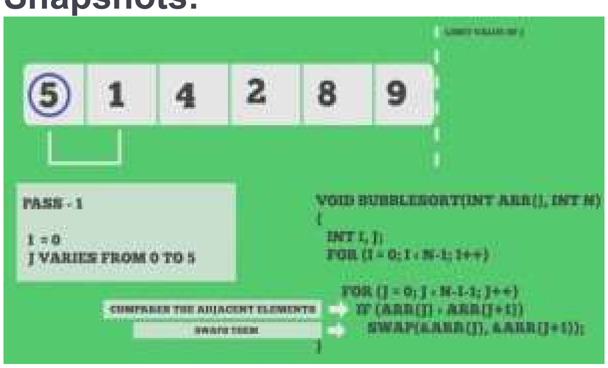
Output:
Sorted array:

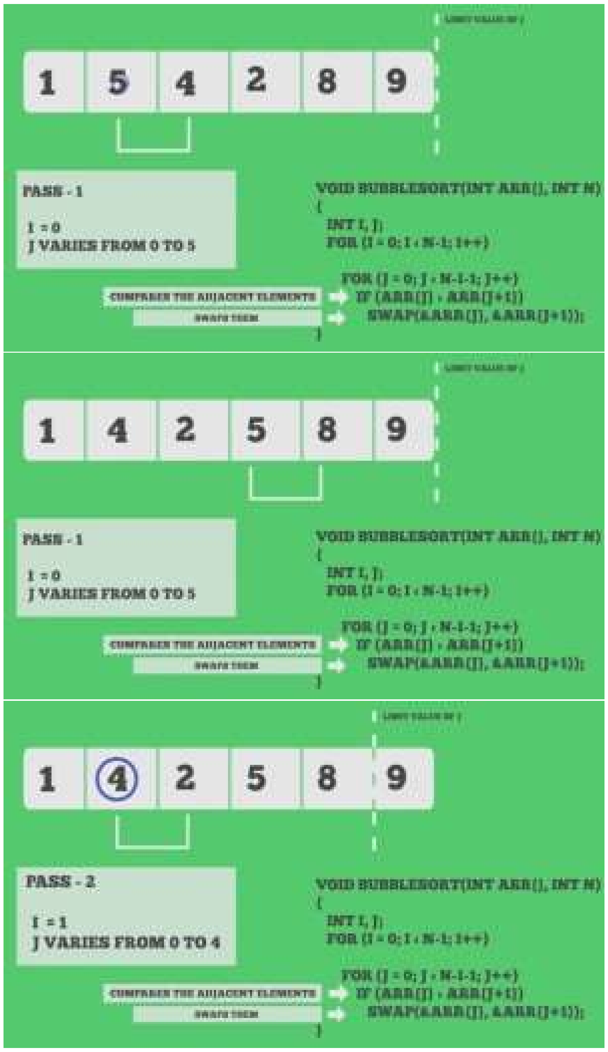
11 12 22 25 34 64 90

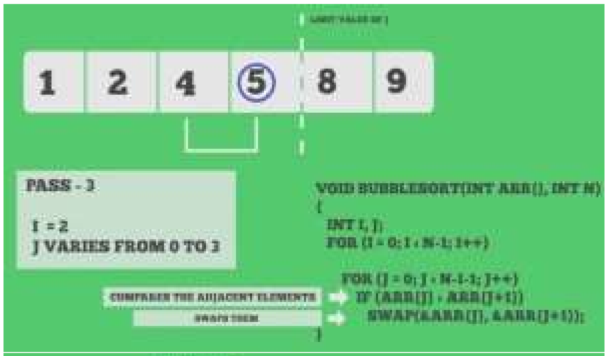
Worst and Average Case Time Complexity: O(n*n).
Worst case occurs when array is reverse sorted.
Best Case Time Complexity: O(n). Best case occurs when array is already sorted.
Auxiliary Space: O(1)
Boundary Cases: Bubble sort takes minimum time (Order of n) when elements are already sorted.
Sorting In Place: Yes
Stable: Yes
Due to its simplicity, bubble sort is often used to introduce the concept of a sorting algorithm.

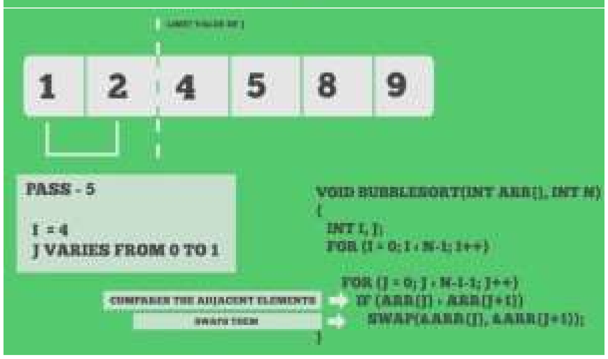
- In computer graphics it is popular for its capability to detect a very small error (like swap of just two elements) in almost-sorted arrays and fix it with just linear complexity (2n).
- For example, it is used in a polygon filling algorithm, where bounding lines are sorted by their x coordinate at a specific scan line (a line parallel to x axis) and with incrementing y their order changes (two elements are swapped) only at intersections of two lines.

Snapshots:




1 5 4 2 8 9
PASS - 1
I = 0
J VARIES FROM 0 TO 5
VOID BUBBLESORT(INT ARR[], INT N)
{
INT I, J;
FOR (I = 0; I < N-1; I++)
FOR (J = 0; J < N-I-1; J++)
IF (ARR[J] > ARR[J+1])
SWAP(&ARR[J], &ARR[J+1]);
}
1 4 2 5 8 9
PASS - 1
I = 0
J VARIES FROM 0 TO 5
VOID BUBBLESORT(INT ARR[], INT N)
{
INT I, J;
FOR (I = 0; I < N-1; I++)
FOR (J = 0; J < N-I-1; J++)
IF (ARR[J] > ARR[J+1])
SWAP(&ARR[J], &ARR[J+1]);
}
1 4 2 5 8 9
PASS - 2
I = 1
J VARIES FROM 0 TO 4
VOID BUBBLESORT(INT ARR[], INT N)
{
INT I, J;
FOR (I = 0; I < N-1; I++)
FOR (J = 0; J < N-I-1; J++)
IF (ARR[J] > ARR[J+1])
SWAP(&ARR[J], &ARR[J+1]);
}


1
2
4
5
8
9
PASS - 3
I = 2
J VARIES FROM 0 TO 3
VOID BUBBLESORT(INT ARR[], INT N)
{
INT I, J;
FOR (I = 0; I < N-1; I++)
FOR (J = 0; J < N-I-1; J++)
IF (ARR[J] > ARR[J+1])
SWAP(&ARR[J], &ARR[J+1]);
}
COMPARES THE ADJACENT ELEMENTS
SWAPS THEM


1
2
4
5
8
9
PASS - 5
I = 4
J VARIES FROM 0 TO 1
VOID BUBBLESORT(INT ARR[], INT N)
{
INT I, J;
FOR (I = 0; I < N-1; I++)
FOR (J = 0; J < N-I-1; J++)
IF (ARR[J] > ARR[J+1])
SWAP(&ARR[J], &ARR[J+1]);
}
COMPARES THE ADJACENT ELEMENTS
SWAPS THEM