# GeeksQuiz

Computer science mock tests for geeks

# 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** )

Following is C implementation of Bubble Sort.

```
// C program for implementation of Bubble sort
#include <stdio.h>

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; i++)
       for (j = 0; j < n-i-1; j++) //Last i elements are already in place
           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++)
        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
```

**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; 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
```

**Time Complexity:** O(n*n)

**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 (Source: Wikipedia)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Category: Searching and Sorting



**Krishna Kanta Singh** • 6 months ago

```
for (c = 0 ; c < ( n - 1 ); c++)
{
for (d = 0 ; d < n - c - 1; d++)
{
if (array[d] > array[d+1]) /* For decreasing order use < */
{
swap = array[d];
array[d] = array[d+1];
array[d+1] = swap;
}
}
}
```

n-1 and not n

8 • Reply • Share ›

**Zsw-seu** ↗ Krishna Kanta Singh • 2 months ago

I agree your answer,it should be n-1

1 • Reply • Share ›

**Vikas Malviya** ➚ Krishna Kanta Singh • a month ago

Yes, we are effectively moving n-1 elements to their correct position.So the remaining one element moves to its correct location automatically.

• Reply • Share ›

**Rajat Ghai** • a year ago

In bubble sort we compare 1st element with 2nd element. Then if 1st element is greater that 2nd element we swap them. After that we compare 2nd element with 3rd element. And if 2nd element is greater than third then we swap them.
You can learn from step by step tutorial Bubble Sort in C.

3 • Reply • Share ›

**pavithra** • 2 months ago

can anyone help me by answering this question please
where do we use bubble sort in real life?? Any examples??

• Reply • Share ›

**Guest** • 3 months ago

best case time complexity should hv been O(n^2)

• Reply • Share ›

**Vijay Prajapati** • 4 months ago

Time Complexity: O(n*n) ?????

• Reply • Share ›

**Yogesh** • 4 months ago

Hi frnds, I tried to sort an array using the above optimized implementation, and its not working due to variable swap. Please can any one explain what i m doing wrong. I tried with { 5,1,4,2,8 }.

• Reply • Share ›

**Krishna Kanta Singh** • 6 months ago

should be less i

• Reply • Share ›

**ram.suriya** • 10 months ago

```
void bubble(int *arr, int n){
bool swapped = true;
while( swapped == true ){
swapped = false;
for( int i=0; i<n-1; i++){="" if(="" arr[i]="">
swap(&arr[i],&arr[i+1]);
```

```
swapped = true;
}
}
}
}
```

• Reply • Share ›

**Aman** • a year ago

Hi,
I have came up with a different approach which is more of recursive using only one for loop:

```
void swap(int* i,int* j)
{
int temp = *i;
*i = *j;
*j = temp;
}

void Bubble_Sort(int arr[],int l)
{
int count = 0;
for(int i=0;i<l-1;i++) {="" if(arr[i]="">> arr[i+1])

{
swap(&arr[i],&arr[i+1]);
count++;
}
}
if(count == 0)
return;
Bubble_Sort(arr, l);
}
```

• Reply • Share ›

**Manish** ➚ Aman • a year ago

Ya its a nice attempt of thinking from different perspective. But question is what we want to achieve with the new approach. Recursive approach still runs in O(n^2) complexity, apart from consuming stack for recursive calls. Rather iterative solution can be better choice if we want to reduce memory consumption but that too Still be O(n^2). e.g. We can keep a boolean variable, and in the for loop we can, after each iteration of the loop we can check value of boolean variable and i,

```
if(i==l-2 && isRepeatRequired==true) { i = 0; isRepeatRequired= false; }
```

2 • Reply • Share ›

**Aman** Manish • a year ago

Correctly pointed out. By using recursion over iteration we are actually doing trade off between time and memory. But here there is no trade off and instead we have more memory/stack consumption.

• Reply • Share ›