

The Learning Point

Main Page: The Complete Index at a Glance	Mathematics	Computer Science	Physics
Electrical Science and Engineering	An Introduction to Graphics and Solid Modelling		
Test Preparations	Ace the Programming Interviews		
CoursePlex: Online Open Classes from Coursera, Udacity, etc		About	

Mi piace

Place a 3 persone. [Sign Up](#) per vedere cosa piace ai tuoi amici.

To go through the C program / source-code, scroll down to the end of this page

Bubble Sort

It's a sorting algorithm, in which each pair of adjacent items are compared and swapped if they are in wrong order. The comparison is repeated until no swaps are needed, indicating that the list is sorted. The smaller elements 'bubble' to the top of the list, hence, the name Bubble Sort. In this like selection sort, after every pass the largest element moves to the highest index position of the list.

Algorithm:

It starts with the first element of the list. The comparison of the adjacent pair of elements and swapping is done until the list is sorted as follows

1. Compare two adjacent elements and check if they are in correct order (that is second one has to be greater than the first).
2. Swap them if they are not in correct order.

Let iter denotes the number of iterations, then for iter ranging from 1 to n-1 (where n is total number of elements in the list) check

1. If value of the second item at position iter is lesser than the value of the first item i.e. at position iter-1, then swap them.
2. Else, move to the next element at position iter+1.

The effective size of the list is hence reduced by one after every pass and the largest element is moved to its final position in the sorted array. Repeat the two steps until the list is sorted and no swap is required i.e. the effective size is reduced to 1.

To see a Java Applet Visualization of the Bubble Sort Mechanism, click on the image below :

Quick Links to Various Sections on the Main Page

The Learning Point

Computer Science

Algorithms: An introduction to Dynamic Programming

CS - Data Structures and Algorithms

CS - Programming Interviews

CS - Miscellaneous C Programs

CS - Intro to DB Queries

Electrical Science and Engineering

Electrical Sci - DC Circuits

Electrical Sci - Digital Electronics

Mathematics

Mathematics - Linear Algebra

Mathematics - Geometry

Mathematics - Single Variable Calculus

Mathematics - Game Theory

Mathematics - Operations Research

Physics

Physics - Basic Mechanics

Physics - Electrostatics

Sitemap

Recent site activity

These are Quick Links to sections of the Main Page

0



Displaying positions of Counters i & j :0 and 0



Displaying positions of Counters i & j :0 and 1



Need to swap Integers at position 0 and 1 because 53 > 0



State of the array after swapping Integers at position 0 and 1



Displaying positions of Counters i & j :0 and 2



Displaying positions of Counters i & j :0 and 3



Displaying positions of Counters i & j :0 and 4



Displaying positions of Counters i & j :1 and 1



Displaying positions of Counters i & j :1 and 2



Need to swap Integers at position 1 and 2 because 53 > 5



State of the array after swapping Integers at position 1 and 2



Displaying positions of Counters i & j :1 and 3

Related Results**1. [Computer Science bubble](#)**Top answers for **Computer Science bubble**www.Answered-Questions.com**2. [Computer Science bubble near you](#)**Get local answers for **Computer Science bubble**www.Answered-Questions.com**3. [Searching for Computer Science bubble?](#)**Discover 100+ answers for **Computer Science bubble**www.Answered-Questions.com

AdChoices

Chitika | Opt out?

Properties:

1. Best Case performance – When the list is already sorted, we require only one pass to check, hence $O(n)$.
2. Worst Case performance – When the list is in the reverse order, n number of comparisons and swap are required to be done n number of times, hence $O(n^2)$.
3. Average Case performance – $O(n^2)$
4. It does not require any extra space for sorting, hence $O(1)$ extra space.

Complete Tutorial with example :

Bubble Sort

It's a sorting algorithm, in which each pair of adjacent items are compared and swapped if they are in wrong order. The comparison is repeated until no swaps are needed, indicating that the list is sorted. The smaller elements 'bubble' to the top of the list, hence, the name Bubble Sort. In this like selection sort, after every pass the largest element moves to the highest index position of the list.

Algorithm:

It starts with the first element of the list. The comparison of the adjacent pair of elements and swapping is done until the list is sorted as follows

1. Compare two adjacent elements and check if they are in correct order (that is second one has to be greater than the first).
2. Swap them if they are not in correct order.

Let *iter* denotes the number of iterations, then for *iter* ranging from 1 to n-1 (where n is total number of elements in the list) check

1. If value of the second item at position *iter* is lesser than the value of the first item i.e. at position *iter*-1, then swap them.
2. Else, move to the next element at position *iter*+1.

The effective size of the list is hence reduced by one after every pass and the largest element is moved to its final position in the sorted array. Repeat the two steps until the list is sorted and no swap is required i.e. the effective size is reduced to 1.

Bubble Sort - C Program Source Code

```
#include<stdio.h>
/* Logic : Do the following thing until the list is sorted
    (i) Compare two adjacent elements and check if they are in correct order(that is second one has
        to be greater than the first).
    (ii) Swap them if they are not in correct order.
*/
void BubbleSort(int *array,int number_of_elements)
{
    int iter, temp, swapped;
    do
    {
        swapped = 0; /* If no element is swapped array is sorted */
        /* In the following loop compare every pair of adjacent elements and check
            if they are in correct order */
        for(iter = 1; iter < number_of_elements; iter++)
        {
            if(array[iter-1] > array[iter])
            {
                temp = array[iter-1];
                array[iter-1] = array[iter];
                array[iter] = temp;
                swapped = 1;
            }
        }
    }while(swapped);
}
int main()
{
    int number_of_elements;
    scanf("%d",&number_of_elements);
    int array[number_of_elements];
    int iter;
```

```

for(iter = 0;iter < number_of_elements;iter++)
{
    scanf("%d",&array[iter]);
}
/* Calling this functions sorts the array */
BubbleSort(array,number_of_elements);
for(iter = 0;iter < number_of_elements;iter++)
{
    printf("%d ",array[iter]);
}
printf("\n");
return 0;
}

```

Related Tutorials :

<u>Bubble Sort</u>	One of the most elementary sorting algorithms to implement - and also very inefficient. Runs in quadratic time. A good starting point to understand sorting in general, before moving on to more advanced techniques and algorithms. A general idea of how the algorithm works and a the code for a C program.	<u>Insertion Sort</u>	Another quadratic time sorting algorithm - an example of dynamic programming. An explanation and step through of how the algorithm works, as well as the source code for a C program which performs insertion sort.	<u>Selection Sort</u>	Another quadratic time sorting algorithm - an example of a greedy algorithm. An explanation and step through of how the algorithm works, as well as the source code for a C program which performs selection sort.	<u>Shell Sort</u>	An inefficient but interesting algorithm, the complexity of which is not exactly known.	<u>Merge Sort</u>	An example of a Divide and Conquer algorithm. Works in $O(n \log n)$ time. The memory complexity for this is a bit of a disadvantage.	<u>Quick Sort</u>	In the average case, this works in $O(n \log n)$ time. No additional memory overhead - so this is better than merge sort in this regard. A partition element is selected, the array is restructured such that all elements greater or less than the partition are on opposite sides of the partition. These two parts of the array are then sorted recursively.	<u>Heap Sort</u>	Efficient sorting algorithm which runs in $O(n \log n)$ time. Uses the Heap data structure.	<u>Binary Search Algorithm</u>	
------------------------------------	--	---------------------------------------	---	---------------------------------------	--	-----------------------------------	---	-----------------------------------	---	-----------------------------------	---	----------------------------------	---	--	--

Some Important Data Structures and Algorithms, at a glance:

Arrays : Popular Sorting and Searching Algorithms			
<u>Bubble Sort</u>	<u>Insertion Sort</u>	<u>Selection Sort</u>	<u>Shell Sort</u>
<u>Merge Sort</u>	<u>Quick Sort</u>	<u>Heap Sort</u>	<u>Binary Search Algorithm</u>
Basic Data Structures and Operations on them			
<u>Stacks</u>	<u>Queues</u>	<u>Single Linked</u>	<u>Double Linked</u>

		List	List
Circular Linked List	1.		

Tree Data Structures			
Binary Search Trees	Heaps	Height Balanced Trees	
Graphs and Graph Algorithms			
Depth First Search	Breadth First Search	Minimum Spanning Trees: Kruskal Algorithm	Minumum Spanning Trees: Prim's Algorithm
Dijkstra Algorithm for Shortest Paths	Floyd Warshall Algorithm for Shortest Paths	Bellman Ford Algorithm	
Popular Algorithms in Dynamic Programming			
Dynamic Programming	Integer Knapsack problem	Matrix Chain Multiplication	Longest Common Subsequence
Greedy Algorithms			
Elementary cases : Fractional Knapsack Problem, Task Scheduling	Data Compression using Huffman Trees		

Consigli

Registrazione

Crea un account o [accedi](#) per vedere cosa consigliano i tuoi amici.



Algorithms: Graph Traversal : Depth First Search (with C Program source code) – The Learning Point
3 people recommended this.



The Learning Point
5 people recommended this.

Plug-in sociale di Facebook

[Like](#)
[Send](#)
3 people like this. [Sign Up](#) to see what your friends li



Recommend this on Google



Add a comment...



Ramaram Choudhary · Computer Teacher at Apex sr.sec.school.ji
 too good
[Reply](#) · [Like](#) · September 6 at 8:52am

Facebook social plugin



**bubblesort.png** (34k)

Prashant Bhattacharji, Oct

v.1

[Accedi](#) | [Attività recente del sito](#) | [Segnala abuso](#) | [Stampa pagina](#) | [Rimuovi accesso](#) | Powered by [Google Sites](#)