

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

[Computer Science](#) >

Arrays and Sorting: Insertion Sort (with C Program source code)

Quick Links to Various Sections on the Main Page

[Sign Up](#) per vedere cosa piace ai tuoi amici.

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

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

Insertion sort

Insertion sort uses linear search to find the location of the 1st element in the unsorted list, in the sorted portion of the list. It is an elementary sorting algorithm best used to sort small data sets or insert a new element in the sorted list.

Algorithm:

Insertion sort starts with a sorted list of size 1 and inserts elements one at a time. It continues inserting each successive element into the sorted list so far.

1. Suppose if the array is sorted till index i then we can sort the array till $i+1$ by inserting the $(i+1)$ th element in the correct position from 0 to $i+1$.
2. The position at which $(i+1)$ th element has to be inserted has to be found by iterating from 0 to i .
3. As any array is sorted till 0th position (Single element is always sorted) and we know how to expand, we can sort the whole array.

To check out a Java Applet Visualization of Insertion Sort, click on the image below :

These are Quick Links to
sections of the Main Page

0

Insertion Sort

for i -> 0 to lastIndex - 1

key = array[i] , j = i-1

while(j >= 0 && array[j] > key)

array[j+1] <- array[j]

array[j] <- key ;

next i

9 89 83 2 95

Original Array

9 89 83 2 95

i = 1 And Key = array[1]= And j = i - 1 =0

9 89 83 2 95
|j |i

Displaying positions of Counters i & j :1 and 0

9 89 83 2 95

array[1] <- key (key = 89)

9 89 83 2 95

i = 2 And Key = array[2]= And j = i - 1 =1

9 89 83 2 95
|j |i

Displaying positions of Counters i & j :2 and 1

9 89 83 2 95
|j |i

Displaying positions of Counters i & j :2 and 1

9 89 89 2 95

array[2] <- array[1]

9 83 89 2 95

array[1] <- key (key = 83)

9 83 89 2 95

i = 3 And Key = array[3]= And j = i - 1 =2

9 83 89 2 95
|j |i

Displaying positions of Counters i & j :3 and 2
.....and so on

Related Results

1. [Computer Science program](#)

Top answers for **Computer Science program**

[www.Answered-Questions.com](#)

2. [Computer Science program near you](#)

Get local answers for **Computer Science program**

[www.Answered-Questions.com](#)

3. [Searching for Computer Science program?](#)

Discover 100+ answers for **Computer Science program**

[www.Answered-Questions.com](#)

AdChoices

Chitika | Opt out?

Properties:

1. Best case performance – When the array is already sorted $O(n)$.
Total number of comparisons: $N - 1$ and total number of exchanges:
 $N - 1$.
2. Worst case performance – When the array is sorted in reverse
order $O(n^2)$. $N-1$ iterations of comparison and exchanges.
3. Average case performance – $O(n^2)$
4. It is sensitive to the input as the number of comparison and
exchanges depends on the input.
5. It does not require any extra space for sorting, hence $O(1)$ extra
space.

Tutorial :

Insertion sort

Insertion sort uses linear search to find the location of the 1st element in the unsorted list, in the sorted portion of the list. It is an elementary sorting algorithm best used to sort small data sets or insert a new element in the sorted list.

Algorithm:

Insertion sort starts with a sorted list of size 1 and inserts elements one at a time. It continues inserting each successive element into the sorted list so far.

1. Suppose if the array is sorted till index i then we can sort the array till $i+1$ by inserting $i+1$ th element in the correct position from 0 to $i+1$.
2. The position at which $(i+1)$ th element has to be inserted has to be found by iterating from 0 to i .
3. As any array is sorted till 0th position (Single element is always sorted) and we know how to expand, we can sort the whole array.

Property:

1. Best case performance – When the array is already sorted $O(n)$. Total number of comparisons: $N - 1$ and total number of exchanges: $N - 1$.
2. Worst case performance – When the array is sorted in reverse order $O(n^2)$. $N-1$ iterations of comparison and exchanges.
3. Average case performance – $O(n^2)$
4. It is sensitive to the input as the number of comparison and exchanges depends on the input.
5. It does not require any extra space for sorting, hence $O(1)$ extra space.

Insertion Sort - C Program Source Code

```
#include<stdio.h>
/* Logic : Suppose if the array is sorted till index i then we can sort the array till i+1 by inserting
i+1 th element in the correct position from 0 to i+1. The position at which (i+1)th element has
to be inserted has to be found by iterating from 0 to i. As any array is sorted till 0th position
(single element is always sorted) and we know how to expand, we can sort the whole array
*/
void InsertionSort(int *array , int number_of_elements)
{
    int iter,jter;
    for(iter=1;iter<number_of_elements;iter++)
    {
        int current_element = array[iter];
        jter = iter-1;
        while(jter>=0 && array[jter] > current_element)
        {
            array[jter+1] = array[jter];
            jter--;
        }
        array[jter+1] = current_element;
    }
}
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 */
    InsertionSort(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 List</u>	<u>Double Linked List</u>
<u>Circular Linked List</u>	1.		

<u>Tree Data</u>			
----------------------------------	--	--	--

Structures			
Binary Search Trees	Heaps	Height Balanced Trees	
Graphs and Graph Algorithms			
Depth First Search	Breadth First Search	Minimum Spanning Trees: Kruskal Algorithm	Minimum 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

Sign Up to see what your friends like.

Recommend this on Google

Add a comment...

Facebook social plugin

insertion.png (33k)

Prashant Bhattacharji, Oct v.4

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

<http://www.thelearningpoint.net/computer-science/arrays-and-sorting-insertion-sort--with-c-program-source-code>

Pagina 5 di 5