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

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

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

Selection Sort

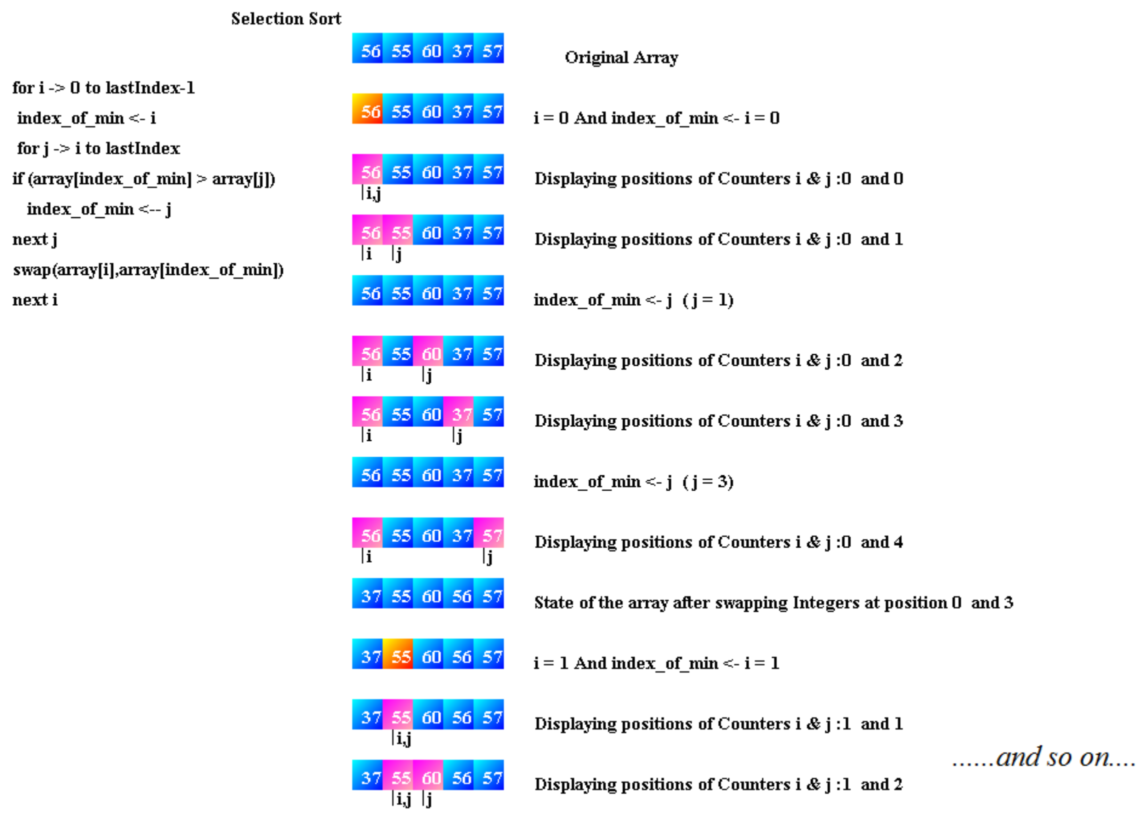
The idea of the selection sort is to find the smallest element in the list and exchange it with the element in the first position. Then, find the second smallest element and exchange it with the element in the second position, and so on until the entire array is sorted.

Algorithm:

For every index from 0 to number_of_elements-1, we find the element which is appropriate for that index and we swap the element which is already there with the element which has to be there. Finding the element which is appropriate for an index is simple. We just have to find the minimum value which is there from that index till number_of_elements-1.

1. minIndex denotes the index which has the minimum value which for now assumed to be
the value at current index and we update it in a for loop.
2. For elements from minIndex+1 to the last element of the list check if some index has got
element smaller than minimum then update minindex to be that index.
3. Then swap the two elements i.e. the element at minidex in 1 and the element at the
updated minindex.
4. Follow the first three steps for every index from 0 to number_of_elements-1.

To Check out a Java Applet Visualization of Selection Sort, click on the image below :



Properties:

Best case performance – When the list is already sorted O(n2).
Worst case performance - When the list is sorted in reverse order O(n2).
Average case performance – O(n2).
It does not require any extra space for sorting, hence O(1) extra space.
It is not stable.
O(n2) time complexity makes it difficult for long lists.

Tutorial :

1 / 2

Selection Sort

The idea of the selection sort is to find the smallest element in the list and exchange it with the element in the first position. Then, find the second smallest element and exchange it with the element in the second position, and so on until the entire array is sorted.

Algorithm:

For every index from 0 to number_of_elements-1, we find the element which is appropriate for that index and we swap the element which is already there with the element which has to be there. Finding the element which is appropriate for an index is simple. We just have to find the minimum value which is there from that index till number_of_elements-1.

1. minIndex denotes the index which has the minimum value which for now assumed to be the value at current index and we update it in a for loop.
2. For elements from minIndex+1 to the last element of the list check if some index has got element smaller than minimum then update minindex to be that index.
3. Then swap the two elements i.e. the element at minidex in 1 and the element at the updated minindex.
4. Follow the first three steps for every index from 0 to number_of_elements-1.

Property:

1. Best case performance – When the list is already sorted $O(n^2)$.
2. Worst case performance - When the list is sorted in reverse order $O(n^2)$.
3. Average case performance – $O(n^2)$.
4. It does not require any extra space for sorting, hence O(1) extra space.
5. It is not stable.

Selection Sort - C Program Source Code

```
#include<stdio.h>
/* Logic : For every index from 0 to number_of_elements-1, we find the element which is appropriate
           for that index and we swap the element which is already there with the element which has to
           be there. Finding the element which is appropriate for an index is simple. We just have to
           find the minimum value which is there from that index till number_of_elements-1.
 */
void SelectionSort(int *array,int number_of_elements)
{
        int iter,jter,minIndex,temp;
        for(iter = 0;iter<number_of_elements;iter++)
        {
                /*minIndex denotes the index which has the minimum value which for now assumed to be
                 the vlaue at current index and we update it in the for loop given below
                 */
                minIndex = iter;
                for(jter = iter+1; jter<number_of_elements;jter++)
                {
                        if(array[jter] < array[minIndex])
                        {
                                /* If some index has got element smaller than minimum then update
                                   minindex  to be that index*/
                                minIndex = jter;
                        }
                }
                temp = array[iter];
                array[iter] = array[minIndex];
                array[minIndex] = temp;
        }
}
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 */
        SelectionSort(array,number_of_elements);
        for(iter = 0;iter < number_of_elements;iter++)
```

```
        {
                printf("%d ",array[iter]);
        }
        printf("\n");
        return 0;

}
```

Related Tutorials :

Bubble Sort	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.	Insertion Sort	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.	Selection Sort	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.	Shell Sort	An inefficient but interesting algorithm, the complexity of which is not exactly known.	Merge Sort	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.	Quick Sort	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.	Heap Sort	Efficient sorting algorithm which runs in O(n log n) time. Uses the Heap data structure.	Binary Search Algorithm	Commonly used algorithm used to find the position of an element in a sorted array. Runs in O(log n) time.

Some Important Data Structures and Algorithms, at a glance:

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

selection.png (41k) Prashant Bhattacharji, Oct v.1