Selection Sort Algorithm

Difficulty Level : Easy • Last Updated : 30 Mar, 2023

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion. This process is repeated for the remaining unsorted portion of the list until the entire list is sorted. One variation of selection sort is called "Bidirectional selection sort" which goes through the list of elements by alternating between the smallest and largest element, this way the algorithm can be faster in some cases.

The algorithm maintains two subarrays in a given array.

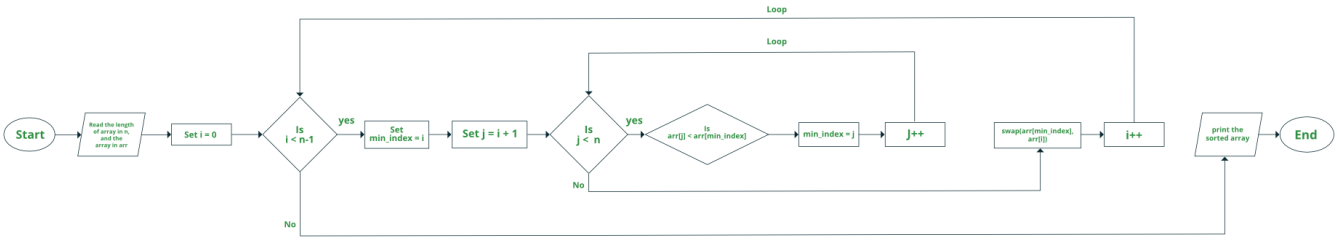
- The subarray which already sorted.
- The remaining subarray was unsorted.

In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the beginning of the sorted subarray.

After every iteration sorted subarray size increase by one and the unsorted subarray size decrease by one.

After the N (size of the array) iteration, we will get a sorted array.

Flowchart of the Selection Sort:



Flowchart for Selection Sort

How does selection sort work?

Lets consider the following array as an example: ***arr[] = {64, 25, 12, 22, 11}***

First pass:

- *For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where* ***64*** *is stored presently, after traversing whole array it is clear that* ***11*** *is the lowest value.*

64	*25*	*12*	*22*	*11*

- *Thus, replace 64 with 11. After one iteration* ***11****, which happens to be the least value in the array, tends to appear in the first position of the sorted list.*

11	*25*	*12*	*22*	*64*

Second Pass:

- *For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.*

11	***25***	*12*	*22*	*64*

- *After traversing, we found that* ***12*** *is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.*

Third Pass:

- *Now, for third place, where **25** is present again traverse the rest of the array and find the third least value present in the array.*

11	*12*	***25***	*22*	*64*

- *While traversing, **22** came out to be the third least value and it should appear at the third place in the array, thus swap **22** with element present at third position.*

11	*12*	***22***	*25*	*64*

Fourth pass:

- *Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array*
- *As **25** is the 4th lowest value hence, it will place at the fourth position.*

11	*12*	*22*	***25***	*64*

Fifth Pass:

- *At last the largest value present in the array automatically get placed at the last position in the array*
- *The resulted array is the sorted array.*

11	***12***	***22***	***25***	***64***

- Initialize minimum value(**min_idx**) to location 0.
- While traversing if any element smaller than **min_idx** is found then swap both values.
- Then, increment **min_idx** to point to the next element.
- Repeat until the array is sorted.

Below is the implementation of the above approach:

C++

```
// C++ program for implementation of
// selection sort
#include <bits/stdc++.h>
using namespace std;

//Swap function
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;
    // One by one move boundary of
    // unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in
        // unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        {
          if (arr[j] < arr[min_idx])
              min_idx = j;
        }
        // Swap the found minimum element
        // with the first element
        if (min_idx!=i)
            swap(&arr[min_idx], &arr[i]);
    }
}
```

```
    int i;
    for (i=0; i < size; i++)
    {
      cout << arr[i] << " ";
      cout << endl;
    }
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
// This is code is contributed by rathbhupendra
```

C

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

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
           if(min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
```

```c
/* 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, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Python3

```python
# Python program for implementation of Selection
# Sort
import sys
A = [64, 25, 12, 22, 11]

# Traverse through all array elements
for i in range(len(A)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j

    # Swap the found minimum element with
    # the first element
    A[i], A[min_idx] = A[min_idx], A[i]

# Driver code to test above
print ("Sorted array")
for i in range(len(A)):
    print("%d" %A[i],end=" , ")
```

Java

```java
import java.io.*;
public class SelectionSort
{
    void sort(int arr[])
    {
        int n = arr.length;

        // One by one move boundary of unsorted subarray
        for (int i = 0; i < n-1; i++)
        {
            // Find the minimum element in unsorted array
            int min_idx = i;
            for (int j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;

            // Swap the found minimum element with the first
            // element
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
    }

    // Prints the array
    void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }

    // Driver code to test above
    public static void main(String args[])
    {
        SelectionSort ob = new SelectionSort();
        int arr[] = {64,25,12,22,11};
        ob.sort(arr);
        System.out.println("Sorted array");
        ob.printArray(arr);
    }
}
/* This code is contributed by Rajat Mishra*/
```

C#

```
// C# program for implementation
// of Selection Sort
```

```
class GFG
{
    static void sort(int []arr)
    {
        int n = arr.Length;

        // One by one move boundary of unsorted subarray
        for (int i = 0; i < n - 1; i++)
        {
            // Find the minimum element in unsorted array
            int min_idx = i;
            for (int j = i + 1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;

            // Swap the found minimum element with the first
            // element
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
    }

    // Prints the array
    static void printArray(int []arr)
    {
        int n = arr.Length;
        for (int i=0; i<n; ++i)
            Console.Write(arr[i]+" ");
        Console.WriteLine();
    }

    // Driver code
    public static void Main()
    {
        int []arr = {64,25,12,22,11};
        sort(arr);
        Console.WriteLine("Sorted array");
        printArray(arr);
    }

}
// This code is contributed by Sam007
```

PHP

```
<?php
// PHP program for implementation
// of selection sort
```

```
    for($i = 0; $i < $n ; $i++)
    {
        $low = $i;
        for($j = $i + 1; $j < $n ; $j++)
        {
            if ($arr[$j] < $arr[$low])
            {
                $low = $j;
            }
        }

        // swap the minimum value to $ith node
        if ($arr[$i] > $arr[$low])
        {
            $tmp = $arr[$i];
            $arr[$i] = $arr[$low];
            $arr[$low] = $tmp;
        }
    }
}

// Driver Code
$arr = array(64, 25, 12, 22, 11);
$len = count($arr);
selection_sort($arr, $len);
echo "Sorted array : \n";

for ($i = 0; $i < $len; $i++)
    echo $arr[$i] . " ";

// This code is contributed
// by Deepika Gupta.
?>
```

Javascript

```
// Javascript program for implementation of selection sort
function swap(arr,xp, yp)
{
    var temp = arr[xp];
    arr[xp] = arr[yp];
    arr[yp] = temp;
}

function selectionSort(arr,  n)
{
    var i, j, min_idx;

    // One by one move boundary of unsorted subarray
```

```
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i + 1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(arr,min_idx, i);
    }
}

function printArray( arr,  size)
{
    var i;
    for (i = 0; i < size; i++)
        document.write(arr[i] + " ");
    document.write(" <br>");
}

var arr = [64, 25, 12, 22, 11];
    var n = 5;
    selectionSort(arr, n);
    document.write("Sorted array: <br>");
    printArray(arr, n);

// This code is contributed by akshitsaxenaa09.
```

Output

```
Sorted array:
11 12 22 25 64
```

Complexity Analysis of Selection Sort:

Time Complexity: The time complexity of Selection Sort is $O(N^2)$ as there are two nested loops:

- One loop to select an element of Array one by one = $O(N)$
- Another loop to compare that element with every other Array element = $O(N)$

Therefore overall complexity = $O(N) * O(N) = O(N*N) = O(N^2)$

Auxiliary Space: $O(1)$ as the only extra memory used is for temporary variables while swapping two values in Array. The selection sort never makes more than $O(N)$ swaps and can be useful when the memory write is a costly operation.

Is Selection Sort Algorithm stable?

Stability: The default implementation is **not stable**. However, it can be made stable. Please see the stable selection sort for details.

Is Selection Sort Algorithm in place?

Yes, it does not require extra space.

Snapshots: Quiz on Selection Sort

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz: Coding practice for sorting

Advantages of Selection Sort Algorithm:

- Simple and easy to understand.
- Preserves the relative order of items with equal keys which means it is stable.
- Works well with small datasets.
- It is adaptable to various types of data types.
- Selection sort is an in-place sorting algorithm, which means it does not require any additional memory to sort the list.
- It has a best-case and average-case time complexity of O(n^2), making it efficient for small data sets.
- It is easy to modify to sort in ascending or descending order.
- It can be easily implemented in hardware, making it suitable for real-time applications.
- It can also be used as a subroutine in more efficient sorting algorithms.
- It does not require any special memory or auxiliary data structures, making it a lightweight solution.
- The algorithm can be easily paralleled, allowing for efficient sorting on multi-core processors.
- It can be used in limited memory environments as it requires minimum extra memory.
- It is easy to understand, making it a popular choice for teaching purposes.
- It is suitable for sorting data with few unique keys, as it performs well in such scenarios.

Disadvantages of the Selection Sort Algorithm:

- Selection sort has a time complexity of O(n^2) in the worst and average case.
- Does not work well on large datasets.
- The selection sort algorithm needs to iterate over the list multiple times, thus it can lead to an unbalanced branch.
- Selection sort has poor cache performance and hence it is not cache friendly.

- Not a good choice for large data sets with slow random access memory (RAM)
- It's not a comparison sort and doesn't have any performance guarantees like merge sort or quick sort.
- It has poor cache performance
- It can cause poor branch prediction due to its high branch misprediction rate
- It has many write operations, leading to poor performance on systems with slow storage.
- It is not a parallelizable algorithm, meaning that it cannot be easily split up to be run on multiple processors or cores.
- It does not handle data with many duplicates well, as it makes many unnecessary swaps.
- It can be outperformed by other algorithms such as quicksort and heapsort in most cases.

Summary:

- Selection sort is a simple and easy-to-understand sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.
- This process is repeated for the remaining unsorted portion of the list until the entire list is sorted.
- It has a time complexity of O(n^2) in the worst and average case which makes it less efficient for large data sets.
- Selection sort is a stable sorting algorithm.
- It can be used to sort different types of data.
- It has specific applications where it is useful such as small data sets and memory-constrained systems.

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

Related Articles

1. Comparison among Bubble Sort, Selection Sort and Insertion Sort
2. A sorting algorithm that slightly improves on selection sort
3. Selection Sort VS Bubble Sort