Selection Sort Algorithm

In this article, we will discuss the Selection sort Algorithm. The working procedure of selection sort is also simple. This article will be very helpful and interesting to students as they might face selection sort as a question in their examinations. So, it is important to discuss the topic.

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. It is also the simplest algorithm. It is an in-place comparison sorting algorithm. In this algorithm, the array is divided into two parts, first is sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and unsorted part is the given array. Sorted part is placed at the left, while the unsorted part is placed at the right.

In selection sort, the first smallest element is selected from the unsorted array and placed at the first position. After that second smallest element is selected and placed in the second position. The process continues until the array is entirely sorted.

The average and worst-case complexity of selection sort is **$O(n^2)$**, where **n** is the number of items. Due to this, it is not suitable for large data sets.

Selection sort is generally used when -

- A small array is to be sorted
- Swapping cost doesn't matter
- It is compulsory to check all elements

Now, let's see the algorithm of selection sort.

Algorithm

1. SELECTION SORT(arr, n)
2.
3. Step 1: Repeat Steps 2 **and** 3 **for** i = 0 to n-1
4. Step 2: CALL SMALLEST(arr, i, n, pos)
5. Step 3: SWAP arr[i] with arr[pos]
6. [END OF LOOP]
7. Step 4: EXIT
8.
9. SMALLEST (arr, i, n, pos)

10. Step 1: [INITIALIZE] SET SMALL = arr[i]
11. Step 2: [INITIALIZE] SET pos = i
12. Step 3: Repeat **for** j = i+1 to n
13. **if** (SMALL > arr[j])
14. SET SMALL = arr[j]
15. SET pos = j
16. [END OF **if**]
17. [END OF LOOP]
18. Step 4: RETURN pos

Working of Selection sort Algorithm

Now, let's see the working of the Selection sort Algorithm.

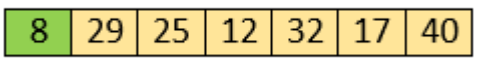
To understand the working of the Selection sort algorithm, let's take an unsorted array. It will be easier to understand the Selection sort via an example.

Let the elements of array are -

Now, for the first position in the sorted array, the entire array is to be scanned sequentially.

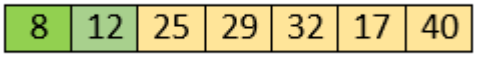
At present, **12** is stored at the first position, after searching the entire array, it is found that **8** is the smallest value.

So, swap 12 with 8. After the first iteration, 8 will appear at the first position in the sorted array.

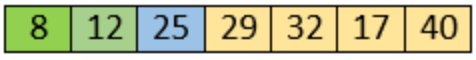


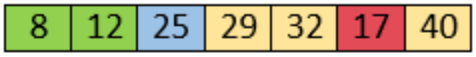
For the second position, where 29 is stored presently, we again sequentially scan the rest of the items of unsorted array. After scanning, we find that 12 is the second lowest element in the array that should be appeared at second position.

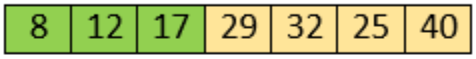
Now, swap 29 with 12. After the second iteration, 12 will appear at the second position in the sorted array. So, after two iterations, the two smallest values are placed at the beginning in a sorted way.

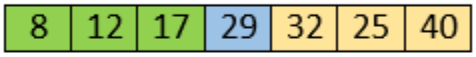


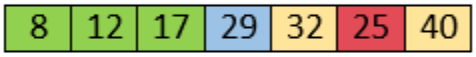
The same process is applied to the rest of the array elements. Now, we are showing a pictorial representation of the entire sorting process.

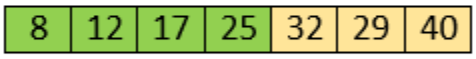












Now, the array is completely sorted.

Selection sort complexity

Now, let's see the time complexity of selection sort in best case, average case, and in worst case. We will also see the space complexity of the selection sort.

1. Time Complexity

Case	Time Complexity
Best Case	$O(n^2)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

- **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of selection sort is **$O(n^2)$**.
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of selection sort is **$O(n^2)$**.
- **Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of selection sort is **$O(n^2)$**.

2. Space Complexity

Space Complexity

Stable

- The space complexity of selection sort is O(1). It is because, in selection sort, an extra variable is required for swapping.

Implementation of selection sort

Now, let's see the programs of selection sort in different programming languages.

Program: Write a program to implement selection sort in C language.

```
1.  #include <stdio.h>
2.
3.  void selection(int arr[], int n)
4.  {
5.      int i, j, small;
6.
7.      for (i = 0; i < n-1; i++)    // One by one move boundary of unsorted subarray
8.      {
9.          small = i; //minimum element in unsorted array
10.
11.         for (j = i+1; j < n; j++)
```

```
12.         if (arr[j] < arr[small])
13.             small = j;
14. // Swap the minimum element with the first element
15.     int temp = arr[small];
16.     arr[small] = arr[i];
17.     arr[i] = temp;
18.     }
19. }
20.
21. void printArr(int a[], int n) /* function to print the array */
22. {
23.     int i;
24.     for (i = 0; i < n; i++)
25.         printf("%d ", a[i]);
26. }
27.
28. int main()
29. {
30.     int a[] = { 12, 31, 25, 8, 32, 17 };
31.     int n = sizeof(a) / sizeof(a[0]);
32.     printf("Before sorting array elements are - \n");
33.     printArr(a, n);
34.     selection(a, n);
35.     printf("\nAfter sorting array elements are - \n");
36.     printArr(a, n);
37.     return 0;
38. }
```

Output:

After the execution of above code, the output will be -

```
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32
```

Program: Write a program to implement selection sort in C++ language.

```cpp
#include <iostream>

using namespace std;

void selection(int arr[], int n)
{
    int i, j, small;

    for (i = 0; i < n-1; i++)    // One by one move boundary of unsorted subarray
    {
        small = i; //minimum element in unsorted array

        for (j = i+1; j < n; j++)
        if (arr[j] < arr[small])
            small = j;
// Swap the minimum element with the first element
    int temp = arr[small];
    arr[small] = arr[i];
    arr[i] = temp;
    }
}

void printArr(int a[], int n) /* function to print the array */
{
    int i;
    for (i = 0; i < n; i++)
        cout<< a[i] <<" ";
}

int main()
{
    int a[] = { 80, 10, 29, 11, 8, 30, 15 };
    int n = sizeof(a) / sizeof(a[0]);
    cout<< "Before sorting array elements are - "<<endl;
    printArr(a, n);
    selection(a, n);
    cout<< "\nAfter sorting array elements are - "<<endl;
```

```
38.     printArr(a, n);
39.
40.     return 0;
41. }
```

Output:

After the execution of above code, the output will be -

```
Before sorting array elements are -
80 10 29 11 8 30 15
After sorting array elements are -
8 10 11 15 29 30 80
```

Program: Write a program to implement selection sort in C# language.

```
1.  using System;
2.  class Selection {
3.  static void selection(int[] arr)
4.  {
5.      int i, j, small;
6.      int n = arr.Length;
7.      for (i = 0; i < n-1; i++)    // One by one move boundary of unsorted subarray
8.      {
9.          small = i; //minimum element in unsorted array
10.
11.         for (j = i+1; j < n; j++)
12.         if (arr[j] < arr[small])
13.             small = j;
14. // Swap the minimum element with the first element
15.     int temp = arr[small];
16.     arr[small] = arr[i];
17.     arr[i] = temp;
18.     }
19. }
20.
21. static void printArr(int[] a) /* function to print the array */
22. {
23.     int i;
```

```
24.     int n = a.Length;
25.     for (i = 0; i < n; i++)
26.     Console.Write(a[i] + " ");
27. }
28.
29.   static void Main() {
30.     int[] a = { 85, 50, 29, 18, 7, 30, 3};
31.     Console.Write("Before sorting array elements are - ");
32.     printArr(a);
33.     selection(a);
34.     Console.Write("\nAfter sorting array elements are - ");
35.     printArr(a);
36.   }
37. }
```

Output:

```
Before sorting array elements are - 85 50 29 18 7 30 3
After sorting array elements are - 3 7 18 29 30 50 85
```

Program: Write a program to implement selection sort in python.

```
1.  def selection(a): # Function to implement selection sort
2.      for i in range(len(a)): # Traverse through all array elements
3.          small = i # minimum element in unsorted array
4.          for j in range(i+1, len(a)):
5.              if a[small] > a[j]:
6.                  small = j
7.      # Swap the found minimum element with
8.      # the first element
9.          a[i], a[small] = a[small], a[i]
10.
11. def printArr(a): # function to print the array
12.
13.     for i in range(len(a)):
14.         print (a[i], end = " ")
15.
16.
```

```
17.
18. a = [69, 14, 1, 50, 59]
19. print("Before sorting array elements are - ")
20. printArr(a)
21. selection(a)
22. print("\nAfter sorting array elements are - ")
23. selection(a)
24. printArr(a)
```

Output:

```
Before sorting array elements are -
69 14 1 50 59
After sorting array elements are -
1 14 50 59 69
```

Program: Write a program to implement selection sort in Java.

```
1.  public class Selection
2.  {
3.      void selection(int a[]) /* function to sort an array with selection sort */
4.  {
5.      int i, j, small;
6.      int n = a.length;
7.      for (i = 0; i < n-1; i++)
8.      {
9.          small = i; //minimum element in unsorted array
10.
11.         for (j = i+1; j < n; j++)
12.         if (a[j] < a[small])
13.             small = j;
14. // Swap the minimum element with the first element
15.     int temp = a[small];
16.     a[small] = a[i];
17.     a[i] = temp;
18.     }
19.
20. }
21. void printArr(int a[]) /* function to print the array */
```

```
{
    int i;
    int n = a.length;
    for (i = 0; i < n; i++)
    System.out.print(a[i] + " ");
}

    public static void main(String[] args) {
    int a[] = { 91, 49, 4, 19, 10, 21 };
    Selection i1 = new Selection();
    System.out.println("\nBefore sorting array elements are - ");
    i1.printArr(a);
    i1.selection(a);
    System.out.println("\nAfter sorting array elements are - ");
    i1.printArr(a);
    System.out.println();
    }
}
```

Output:

```
D:\JTP>javac Selection.java

D:\JTP>java Selection

Before sorting array elements are -
91 49 4 19 10 21
After sorting array elements are -
4 10 19 21 49 91
```

Program: Write a program to implement selection sort in PHP.

```
<?php
    function selection(&$a, $n) /* function to sort an array with selection sort */
{
        for ($i = 0; $i < $n; $i++)
    {
        $small = $i; //minimum element in unsorted array

        for ($j = $i+1; $j < $n; $j++)
        if ($a[$j] < $a[$small])
            $small = $j;
```

```
11. // Swap the minimum element with the first element
12.     $temp = $a[$small];
13.     $a[$small] = $a[$i];
14.     $a[$i] = $temp;
15.     }
16.
17. }
18.     function printArray($a, $n)
19.     {
20.     for($i = 0; $i < $n; $i++)
21.     {
22.         print_r($a[$i]);
23.         echo " ";
24.     }
25.     }
26.     $a = array( 90, 48, 3, 18, 9, 20 );
27.     $n = count($a);
28.     echo "Before sorting array elements are - <br>";
29.     printArray($a, $n);
30.     selection($a, $n);
31.     echo "<br> After sorting array elements are - <br>";
32.     printArray($a, $n);
33. ?>
```

Output:

After the execution of above code, the output will be -