Selection Sort Algorithm

Selection sort is conceptually the most simplest sorting algorithm. This algorithm will first find the **smallest** element in the array and swap it with the element in the **first** position, then it will find the **second smallest** element and swap it with the element in the **second** position, and it will keep on doing this until the entire array is sorted.

It is called selection sort because it repeatedly **selects** the next-smallest element and swaps it into the right place.

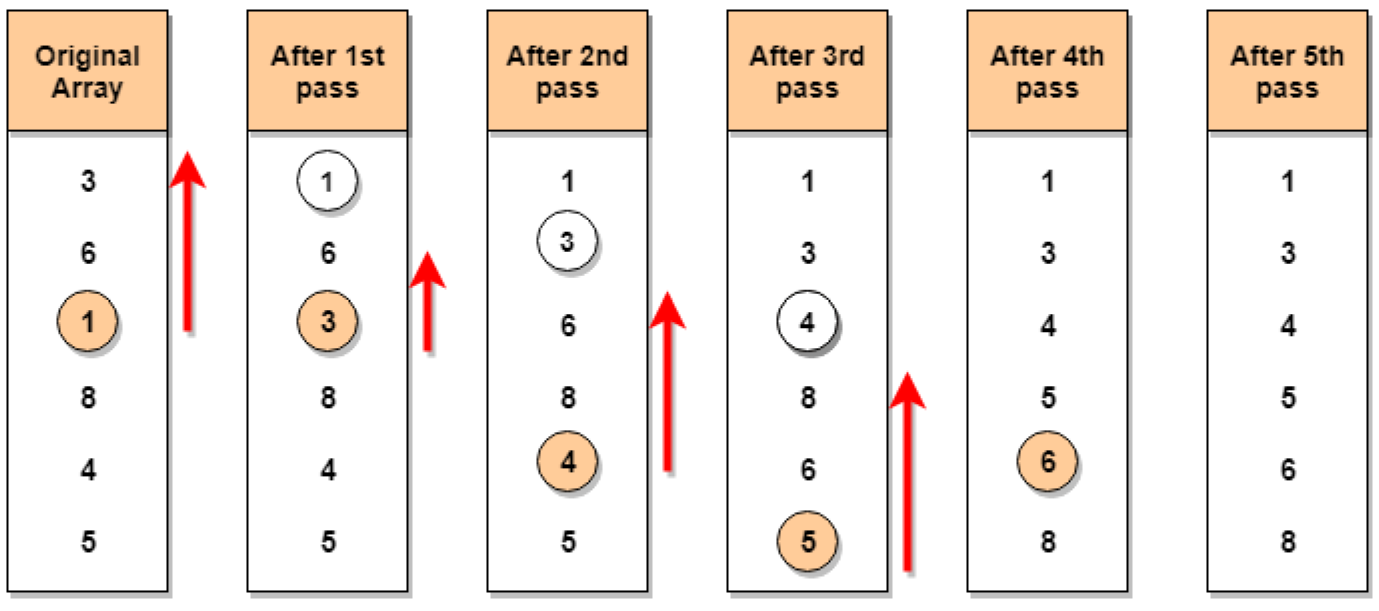
How Selection Sort Works?

Following are the steps involved in selection sort(for sorting a given array in ascending order):

1. Starting from the first element, we search the smallest element in the array, and replace it with the element in the first position.
2. We then move on to the second position, and look for smallest element present in the subarray, starting from index 1, till the last index.
3. We replace the element at the **second** position in the original array, or we can say at the first position in the subarray, with the second smallest element.
4. This is repeated, until the array is completely sorted.

Let's consider an array with values {3, 6, 1, 8, 4, 5}

Below, we have a pictorial representation of how selection sort will sort the given array.



In the **first** pass, the smallest element will be 1, so it will be placed at the first position.

Then leaving the first element, **next smallest** element will be searched, from the remaining elements. We will get 3 as the smallest, so it will be then placed at the second position.

Then leaving 1 and 3(because they are at the correct position), we will search for the next smallest element from the rest of the elements and put it at third position and keep doing this until array is sorted.

Finding Smallest Element in a subarray

In selection sort, in the first step, we look for the smallest element in the array and replace it with the element at the first position. This seems doable, isn't it?

Consider that you have an array with following values {3, 6, 1, 8, 4, 5}. Now as per selection sort, we will start from the first element and look for the smallest number in the array, which is 1 and we will find it at the **index** 2. Once the smallest number is found, it is swapped with the element at the first position.

Well, in the next iteration, we will have to look for the second smallest number in the array. How can we find the second smallest number? This one is tricky?

If you look closely, we already have the smallest number/element at the first position, which is the right position for it and we do not have to move it anywhere now. So we can say, that the first element is sorted, but the elements to the right, starting from index 1 are not.

So, we will now look for the smallest element in the subarray, starting from index 1, to the last index.

Confused? Give it time to sink in.

After we have found the second smallest element and replaced it with element on index 1(which is the second position in the array), we will have the first two positions of the array sorted.

Then we will work on the subarray, starting from index 2 now, and again looking for the smallest element in this subarray.

Implementing Selection Sort Algorithm

In the C program below, we have tried to divide the program into small functions, so that it's easier fo you to understand which part is doing what.

There are many different ways to implement selection sort algorithm, here is the one that we like:

```
// C program implementing Selection Sort

# include <stdio.h>



// function to swap elements at the given index values

void swap(int arr[], int firstIndex, int secondIndex)

{

    int temp;

    temp = arr[firstIndex];

    arr[firstIndex] = arr[secondIndex];

    arr[secondIndex] = temp;

}
```

```c
// function to look for smallest element in the given subarray
int indexOfMinimum(int arr[], int startIndex, int n)
{
    int minValue = arr[startIndex];
    int minIndex = startIndex;

    for(int i = minIndex + 1; i < n; i++) {
        if(arr[i] < minValue)
        {
            minIndex = i;
            minValue = arr[i];
        }
    }
    return minIndex;
}

void selectionSort(int arr[], int n)
{
    for(int i = 0; i < n; i++)
    {
        int index = indexOfMinimum(arr, i, n);
        swap(arr, i, index);
    }

}

void printArray(int arr[], int size)
{
    int i;
    for(i = 0; i < size; i++)

    {

        printf("%d ", arr[i]);

    }

    printf("\n");

}


int main()

{

    int arr[] = {46, 52, 21, 22, 11};

    int n = sizeof(arr)/sizeof(arr[0]);

    selectionSort(arr, n);

    printf("Sorted array: \n");

    printArray(arr, n);

    return 0;

}
```

Copy

Note: Selection sort is an **unstable sort** i.e it might change the occurrence of two similar elements in the list while sorting. But it can also work as a stable sort when it is implemented using linked list.

Complexity Analysis of Selection Sort

Selection Sort requires two nested `for` loops to complete itself, one `for` loop is in the function `selectionSort`, and inside the first loop we are making a call to another function `indexOfMinimum`, which has the second(inner) `for` loop.

Hence for a given input size of `n`, following will be the time and space complexity for selection sort algorithm:

Worst Case Time Complexity [Big-O]: **O(n^2)**

Best Case Time Complexity [Big-omega]: **O(n^2)**

Average Time Complexity [Big-theta]: **O(n^2)**

Space Complexity: **O(1)**