Bubble Sort Algorithm

Bubble Sort is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares all the element one by one and sort them based on their values.

If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will **swap** both the elements, and then move on to compare the second and the third element, and so on.

If we have total n elements, then we need to repeat this process for n-1 times.

It is known as **bubble sort**, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises up to the water surface.

Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required.

NOTE: If you are not familiar with Sorting in data structure, you should first learn what is sorting to know about the basics of sorting.

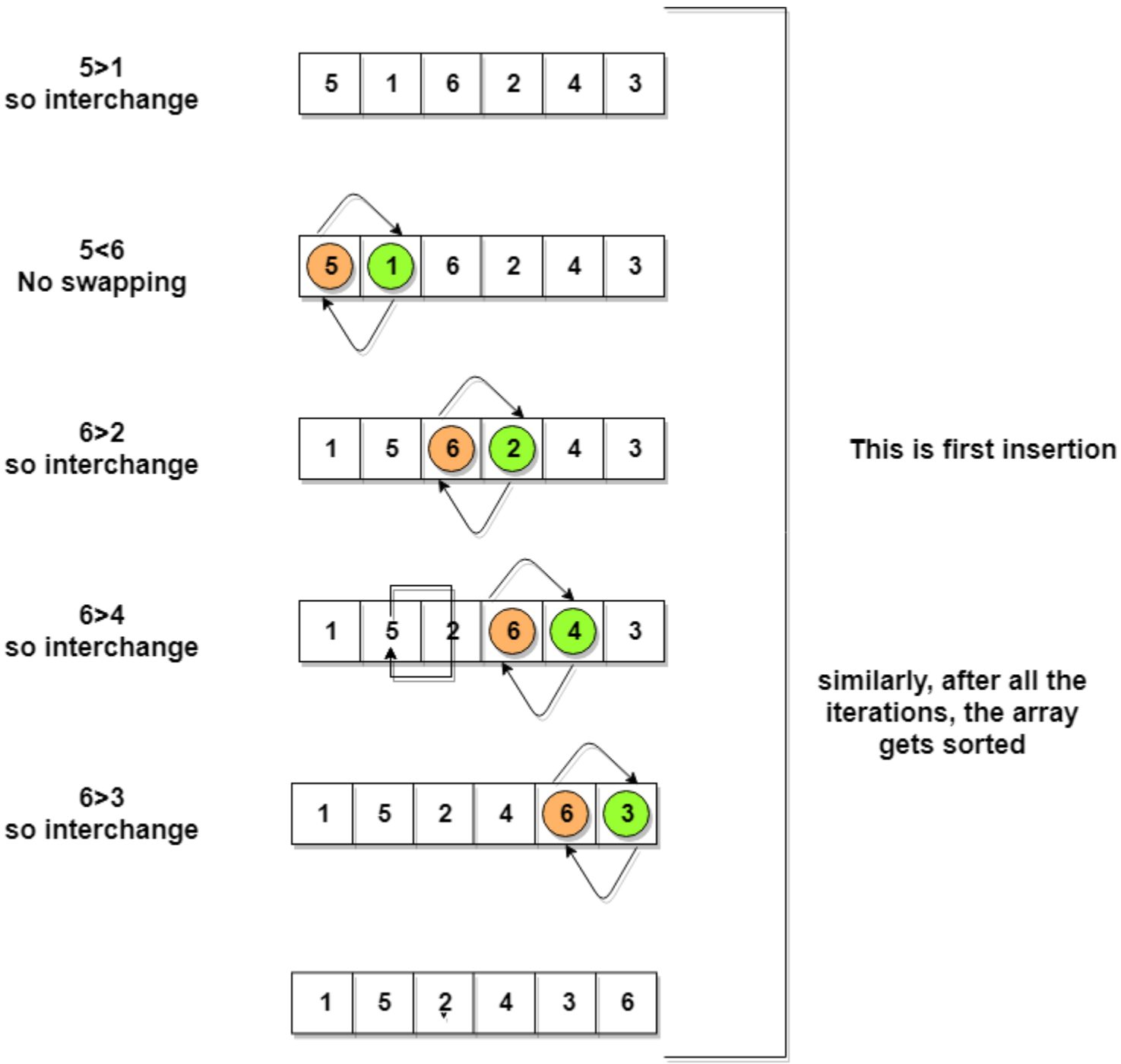
Implementing Bubble Sort Algorithm

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

1. Starting with the first element(index = 0), compare the current element with the next element of the array.
2. If the current element is greater than the next element of the array, swap them.
3. If the current element is less than the next element, move to the next element. **Repeat Step 1**.

Let's consider an array with values {5, 1, 6, 2, 4, 3}

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



So as we can see in the representation above, after the first iteration, 6 is placed at the last index, which is the correct position for it.

Similarly after the second iteration, 5 will be at the second last index, and so on.

Time to write the code for bubble sort:

```c
// below we have a simple C program for bubble sort

#include <stdio.h>



void bubbleSort(int arr[], int n)

{

    int i, j, temp;

    for(i = 0; i < n; i++)

    {

        for(j = 0; j < n-i-1; j++)

    {

        if( arr[j] > arr[j+1])

        {

            // swap the elements

            temp = arr[j];

            arr[j] = arr[j+1];

            arr[j+1] = temp;

        }

    }

}



// print the sorted array

    printf("Sorted Array: ");

    for(i = 0; i < n; i++)

    {

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

    }

}



int main()

{

    int arr[100], i, n, step, temp;

    // ask user for number of elements to be sorted
```

```
    printf("Enter the number of elements to be sorted: ");
    scanf("%d", &n);
    // input elements if the array
    for(i = 0; i < n; i++)
    {
        printf("Enter element no. %d: ", i+1);
        scanf("%d", &arr[i]);
    }
    // call the function bubbleSort
    bubbleSort(arr, n);

    return 0;
}
```

Copy

Although the above logic will sort an unsorted array, still the above algorithm is not efficient because as per the above logic, the outer for loop will keep on executing for **6** iterations even if the array gets sorted after the second iteration.

So, we can clearly optimize our algorithm.

Optimized Bubble Sort Algorithm

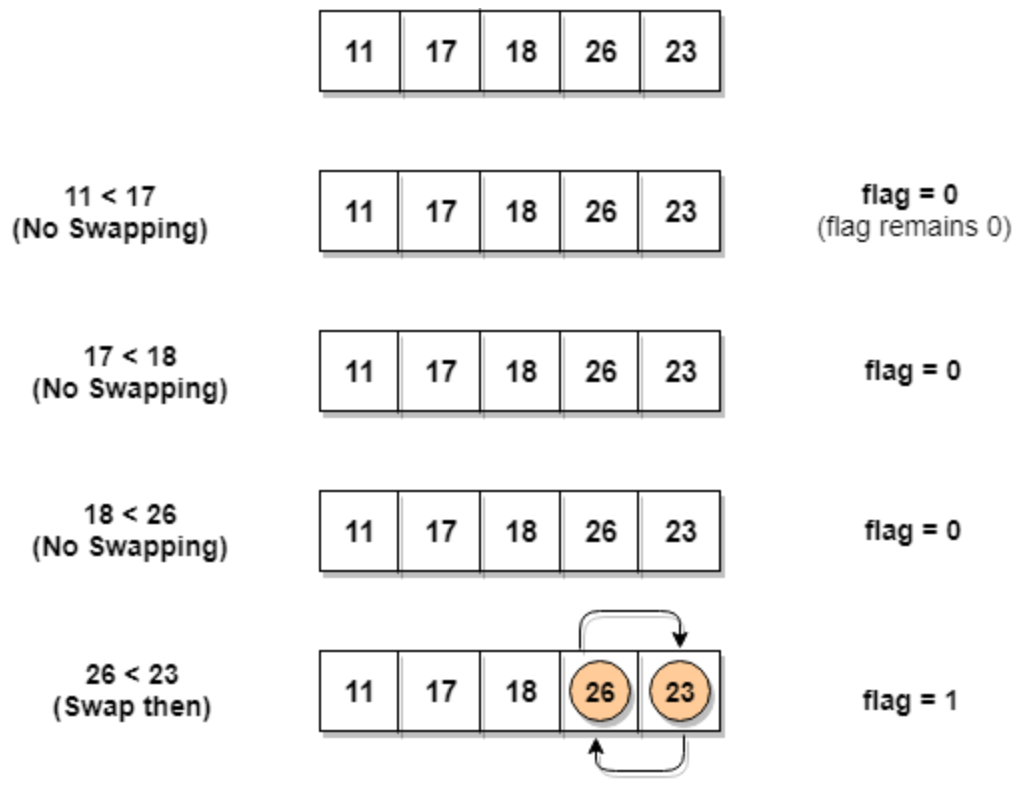
To optimize our bubble sort algorithm, we can introduce a flag to monitor whether elements are getting swapped inside the inner for loop.

Hence, in the inner for loop, we check whether swapping of elements is taking place or not, everytime.

If for a particular iteration, no swapping took place, it means the array has been sorted and we can jump out of the for loop, instead of executing all the iterations.

Let's consider an array with values {11, 17, 18, 26, 23}

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



As we can see, in the first iteration, swapping took place, hence we updated our flag value to 1, as a result, the execution enters the for loop again. But in the second iteration, no swapping will occur, hence the value of flag will remain 0, and execution will break out of loop.

```c
// below we have a simple C program for bubble sort

#include <stdio.h>



void bubbleSort(int arr[], int n)

{

    int i, j, temp, flag=0;

    for(i = 0; i < n; i++)

    {

        for(j = 0; j < n-i-1; j++)

        {

            // introducing a flag to monitor swapping

            if( arr[j] > arr[j+1])

            {

                // swap the elements

                temp = arr[j];

                arr[j] = arr[j+1];

                arr[j+1] = temp;

                // if swapping happens update flag to 1

                flag = 1;

            }

        }

        // if value of flag is zero after all the iterations of inner loop

        // then break out

        if(flag==0)

        {

            break;

        }

    }



    // print the sorted array

    printf("Sorted Array: ");

    for(i = 0; i < n; i++)
```

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


int main()
{
    int arr[100], i, n, step, temp;
    // ask user for number of elements to be sorted
    printf("Enter the number of elements to be sorted: ");
    scanf("%d", &n);
    // input elements if the array
    for(i = 0; i < n; i++)
    {
        printf("Enter element no. %d: ", i+1);
        scanf("%d", &arr[i]);
    }
    // call the function bubbleSort
    bubbleSort(arr, n);


    return 0;
}
```

Copy

```
Enter the number of elements to be sorted: 5
Enter element no. 1: 2
Enter element no. 2: 3
Enter element no. 3: 34
Enter element no. 4: 22
Enter element no. 5: 11
Sorted Array: 2  3  11  22  34
```

In the above code, in the function bubbleSort, if for a single complete cycle of j iteration(inner for loop), no swapping takes place, then flag will remain 0 and then we will break out of the for loops, because the array has already been sorted.

Complexity Analysis of Bubble Sort

In Bubble Sort, n-1 comparisons will be done in the 1st pass, n-2 in 2nd pass, n-3 in 3rd pass and so on. So the total number of comparisons will be,

```
(n-1) + (n-2) + (n-3) + ..... + 3 + 2 + 1
```

```
Sum = n(n-1)/2

i.e O(n²)
```

Hence the **time complexity** of Bubble Sort is **O(n²)**.

The main advantage of Bubble Sort is the simplicity of the algorithm.

The **space complexity** for Bubble Sort is **O(1)**, because only a single additional memory space is required i.e. for temp variable.

Also, the **best case time complexity** will be **O(n)**, it is when the list is already sorted.

Following are the Time and Space complexity for the Bubble Sort algorithm.

- Worst Case Time Complexity [Big-O]: **O(n²)**
- Best Case Time Complexity [Big-omega]: **O(n)**
- Average Time Complexity [Big-theta]: **O(n²)**
- Space Complexity: **O(1)**

Now that we have learned Bubble sort algorithm, you can check out these sorting algorithms and their applications as well:

- Insertion Sort
- Selection Sort
- Quick Sort
- merge Sort
- Heap Sort
- Counting Sort
