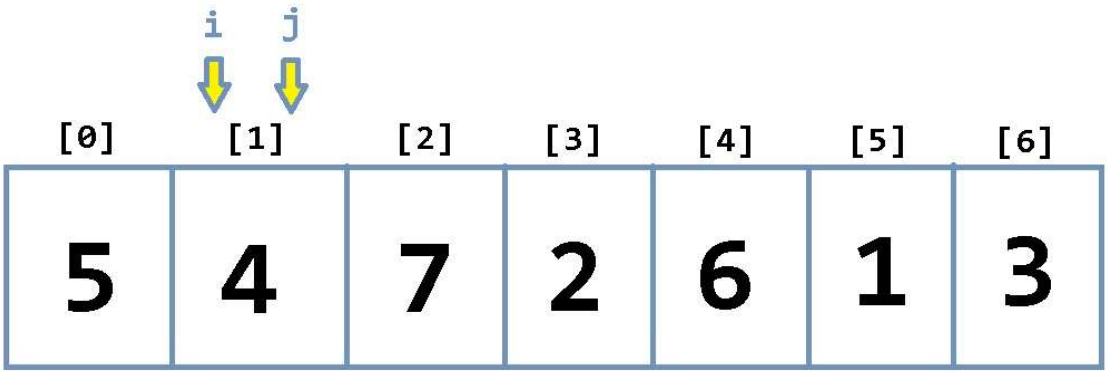
Insertion sort in C++, source code and explanation

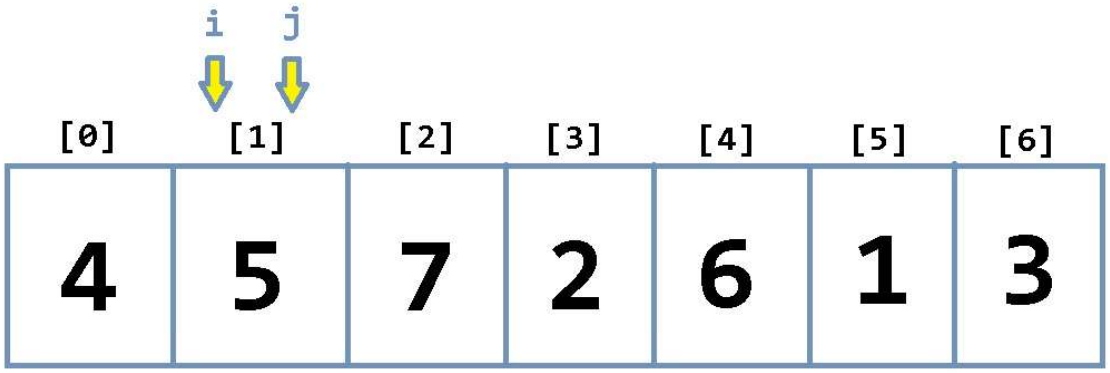
Insertion Sort in C++

Insertion sort is a faster and more improved sorting algorithm than selection sort. In selection sort the algorithm iterates through all of the data through every pass whether it is already sorted or not. However, insertion sort works differently, instead of iterating through all of the data after every pass the algorithm only traverses the data it needs to until the segment that is being sorted is sorted. Again there are two loops that are required by insertion sort and therefore two main variables, which in this case are named 'i' and 'j'. Variables 'i' and 'j' begin on the same index after every pass of the first loop, the second loop only executes if variable 'j' is greater then index 0 AND arr[j] < arr[j - 1]. In other words, if 'j' hasn't reached the end of the data AND the value of the index where 'j' is at is smaller than the value of the index to the left of 'j', finally 'j' is decremented. As long as these two conditions are met in the second loop it will keep executing, this is what sets insertion sort apart from selection sort. Only the data that needs to be sorted is sorted. As always a visual representation helps.

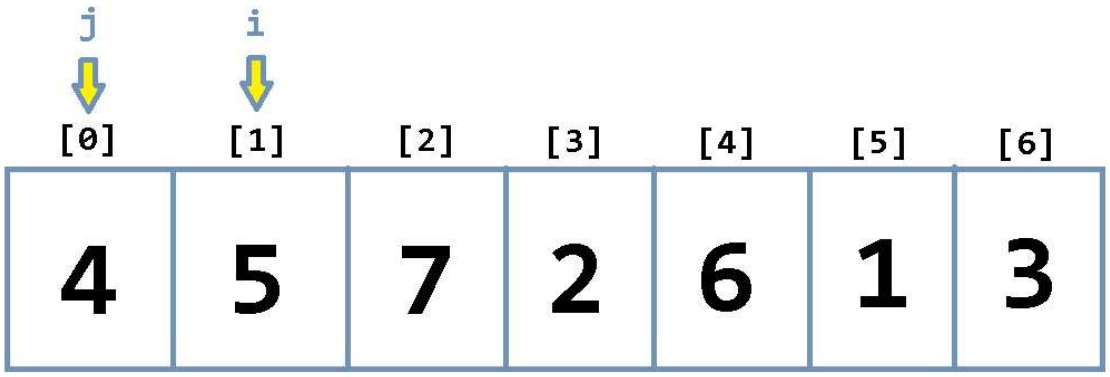
Algorithm for insertion sort starts here variable i begins at index 1 and j = i



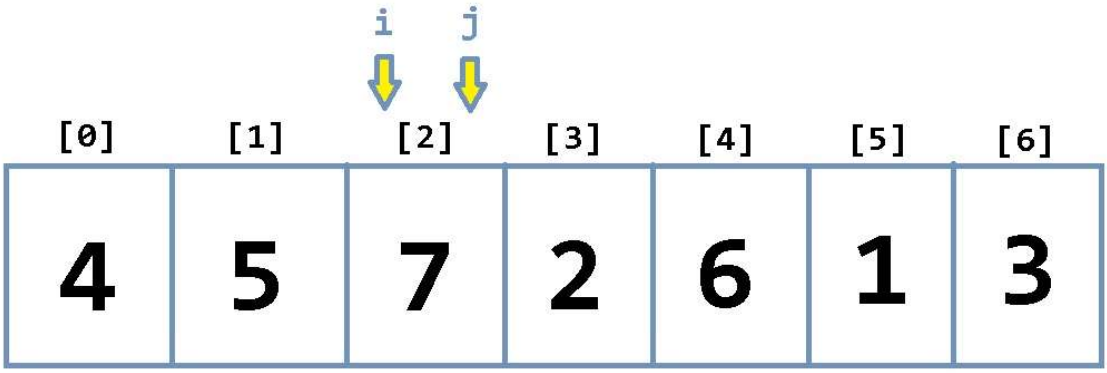
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



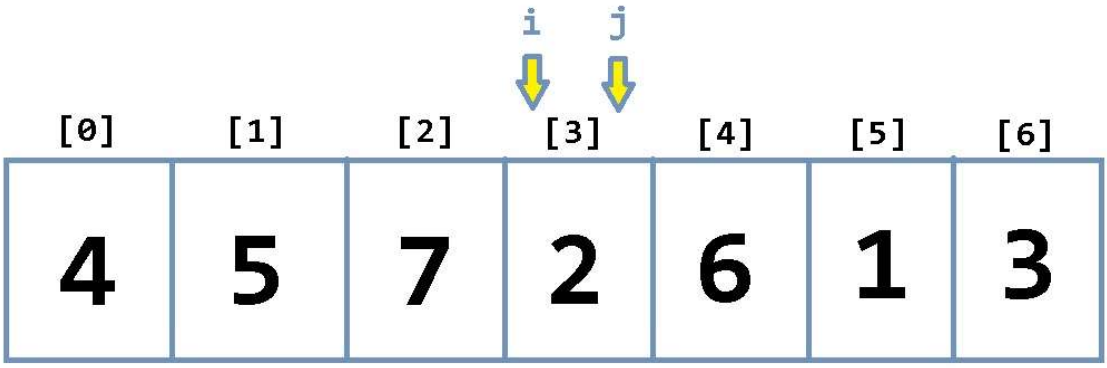
j is decremented, condition for the second loop is no longer met therefore the loop breaks back into the first loop



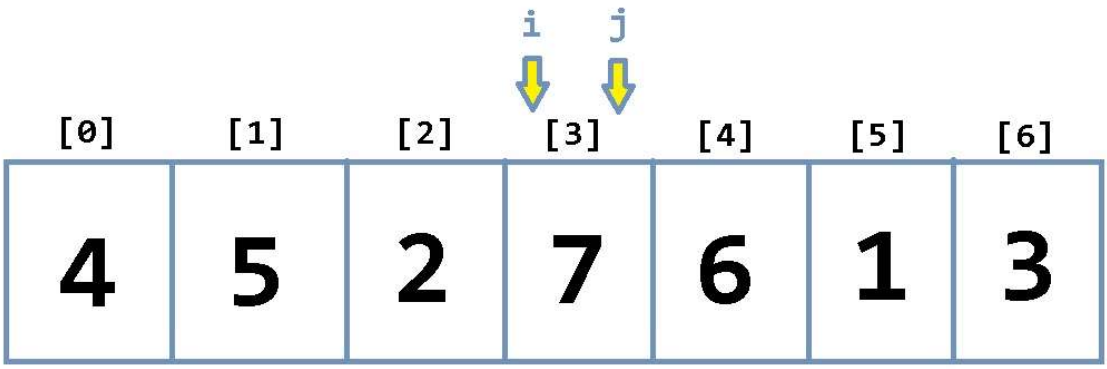
i is incremented, j = i



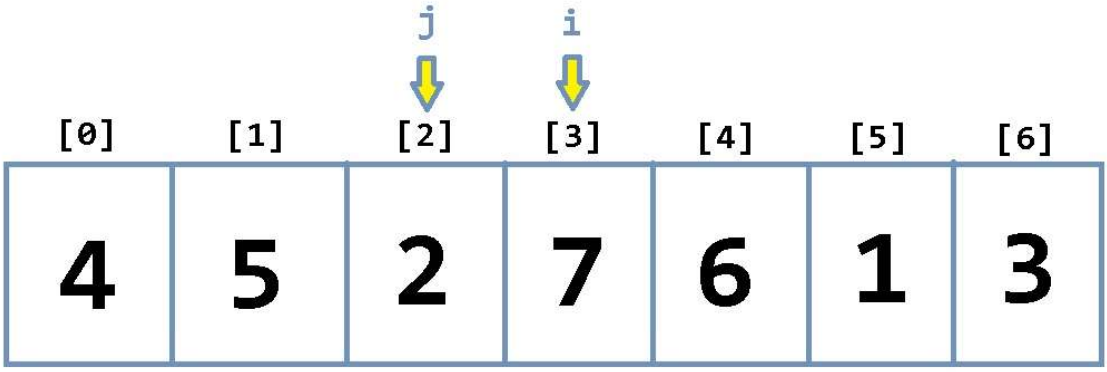
Conditions for the second loop are not met so no swap and j is not decremented, i is incremented and j = i



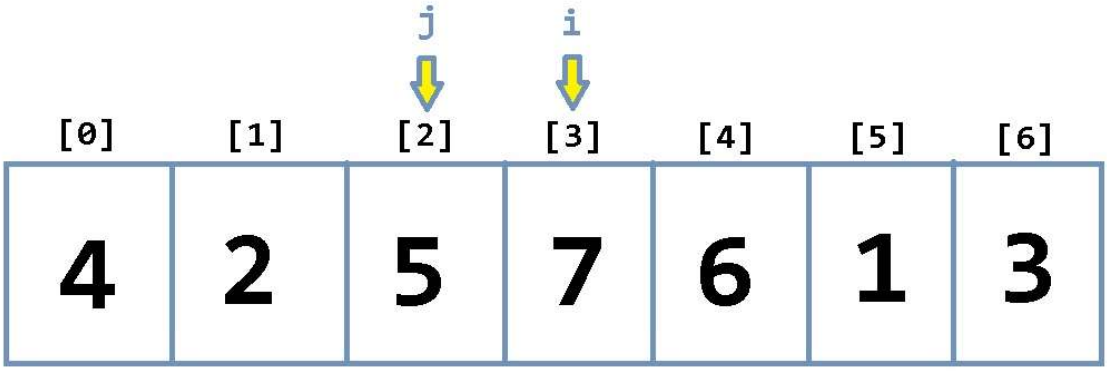
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



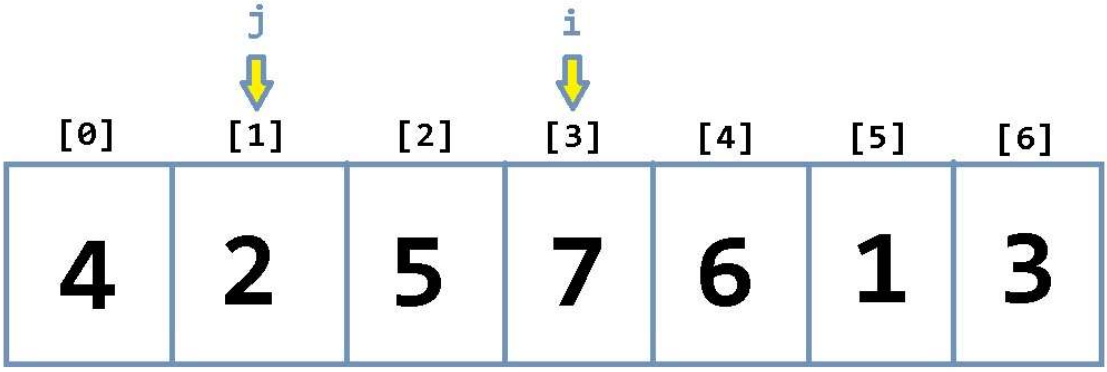
j is decremented



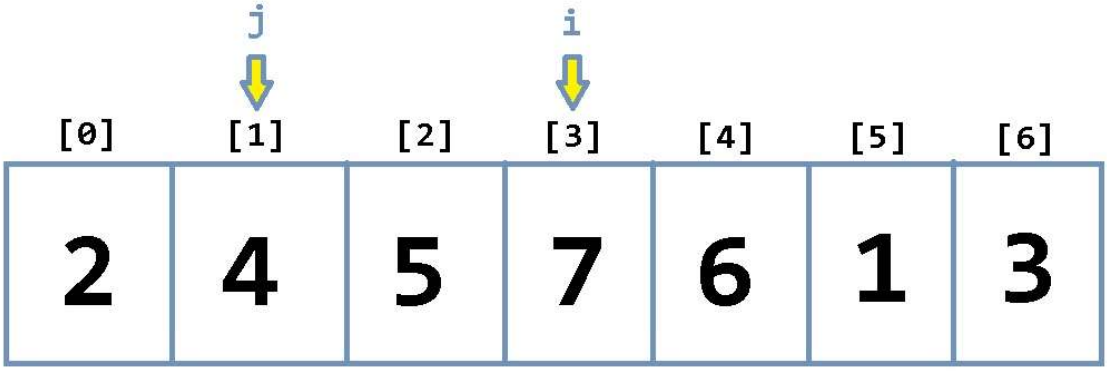
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



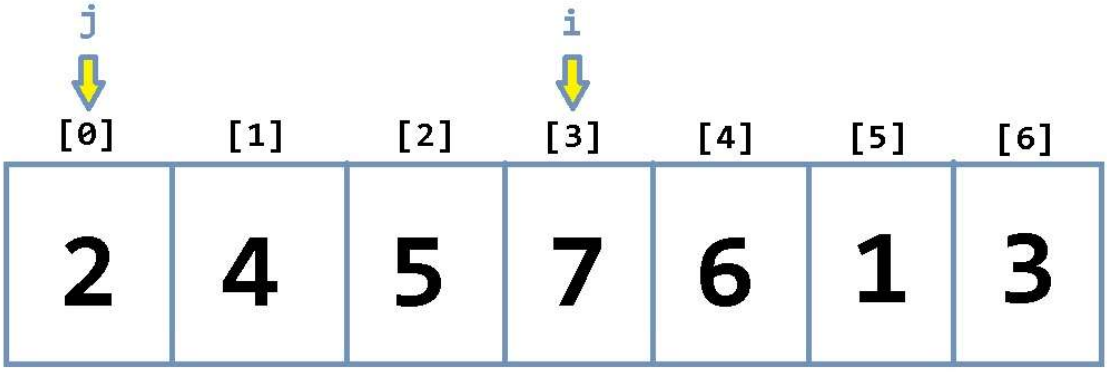
j is decremented



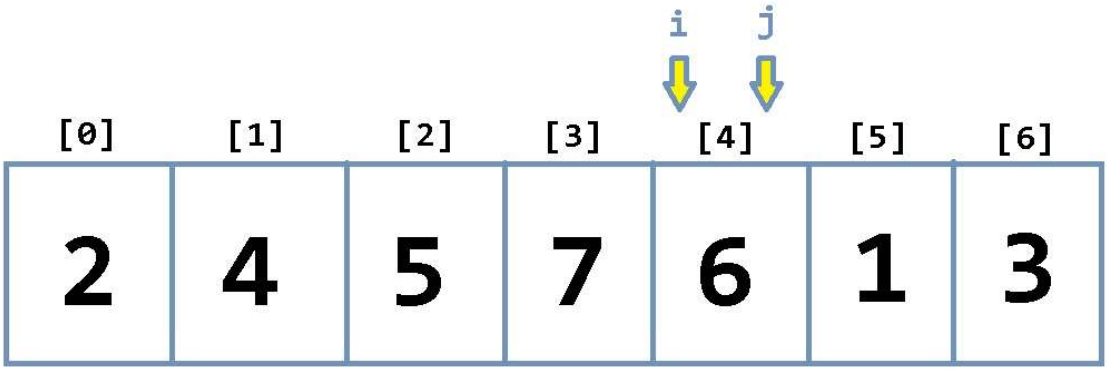
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



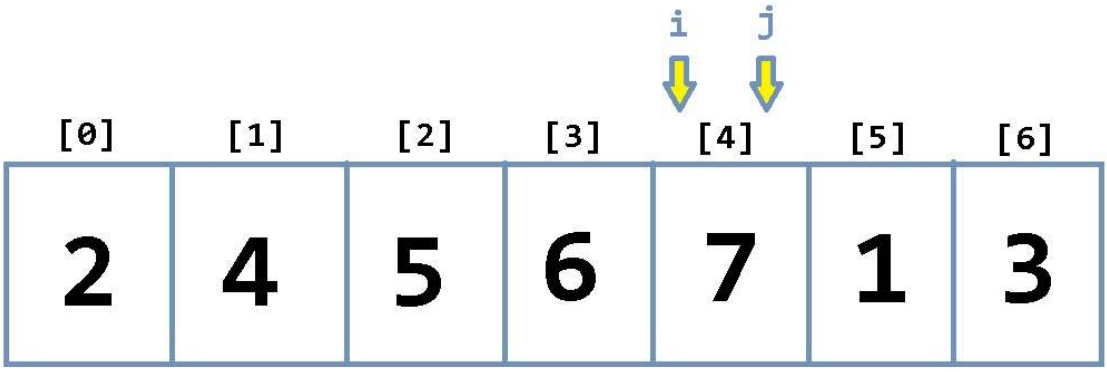
j is decremented, condition for the second loop is no longer met therefore the loop breaks back into the first loop



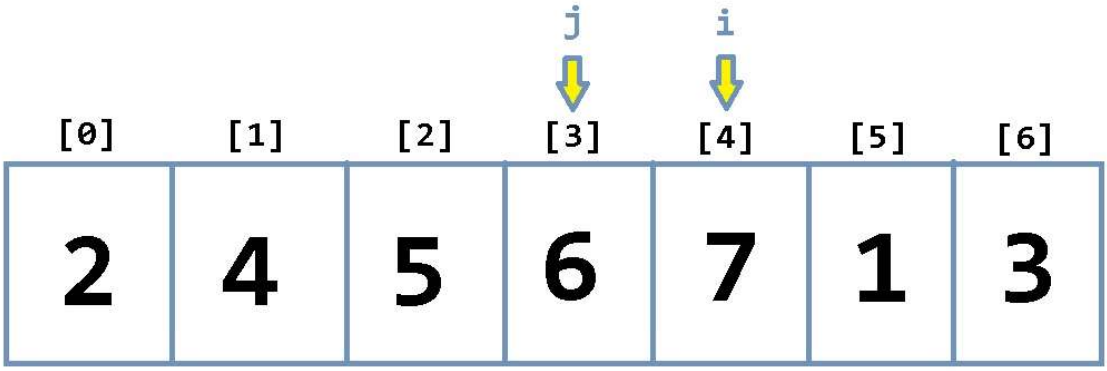
i is incremented, j = i



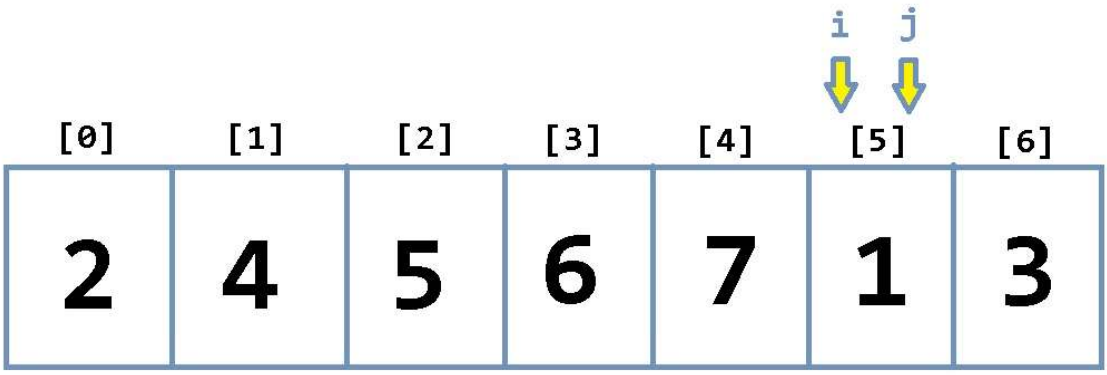
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



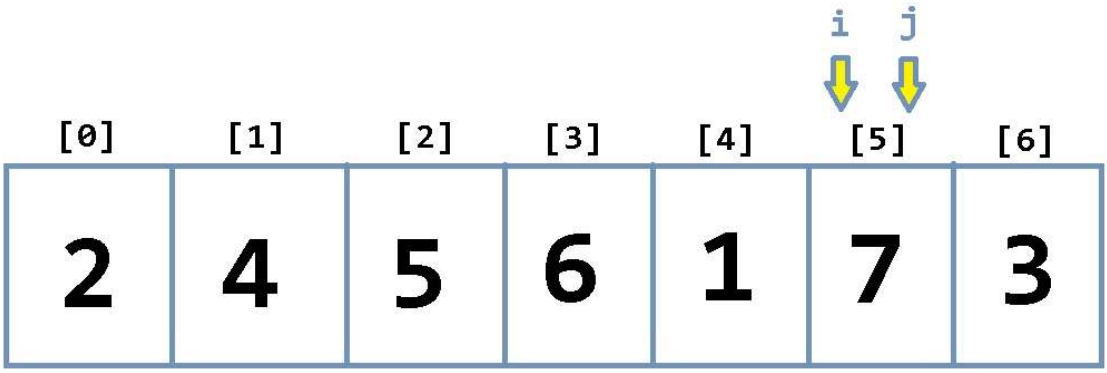
j is decremented, condition for the second loop is no longer met therefore the loop breaks back into the first loop



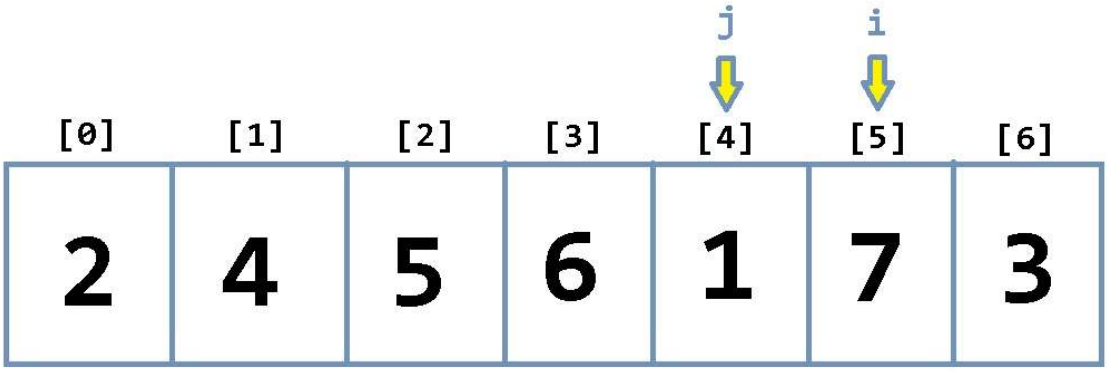
i is incremented, j = i



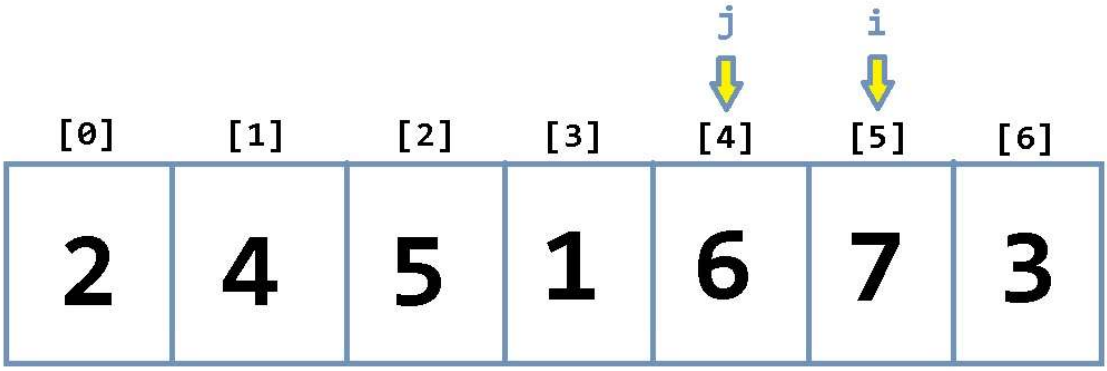
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



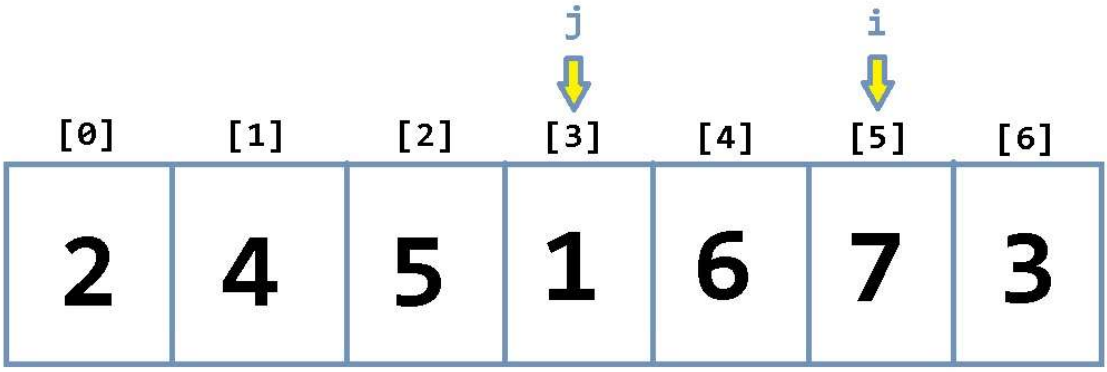
j is decremented



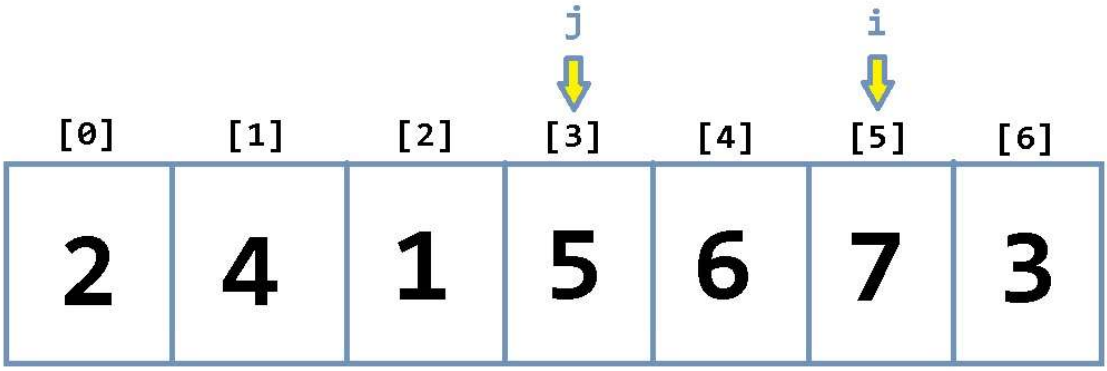
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



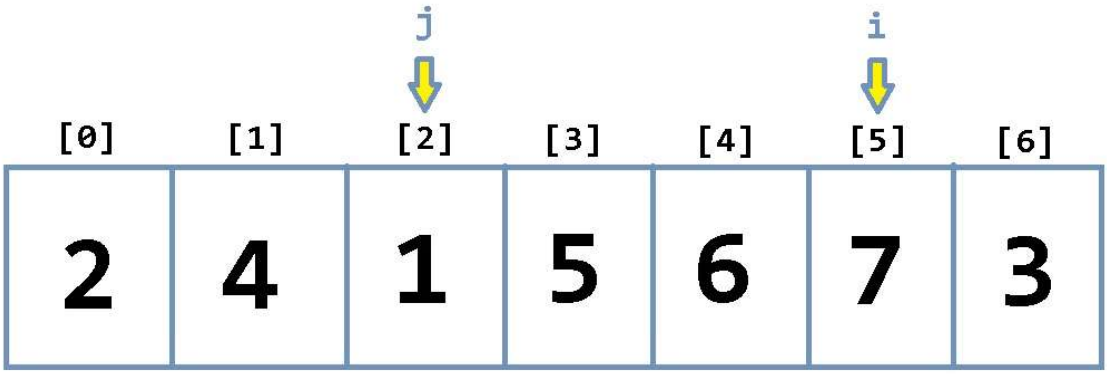
j is decremented



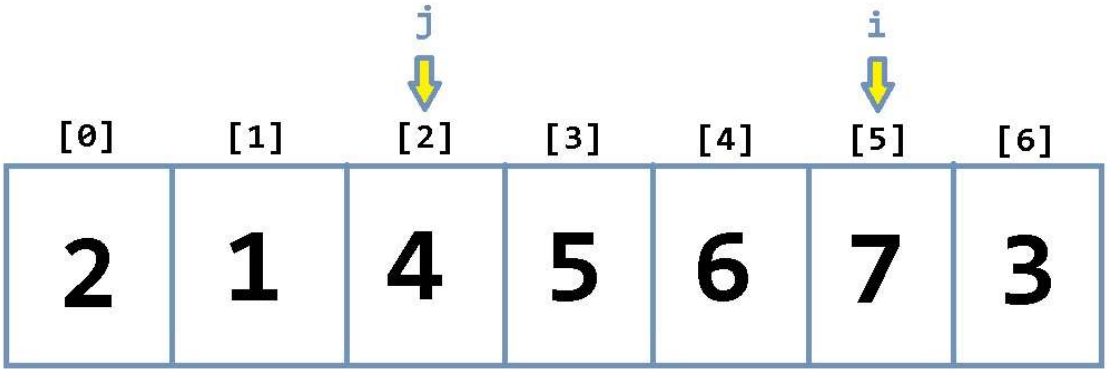
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



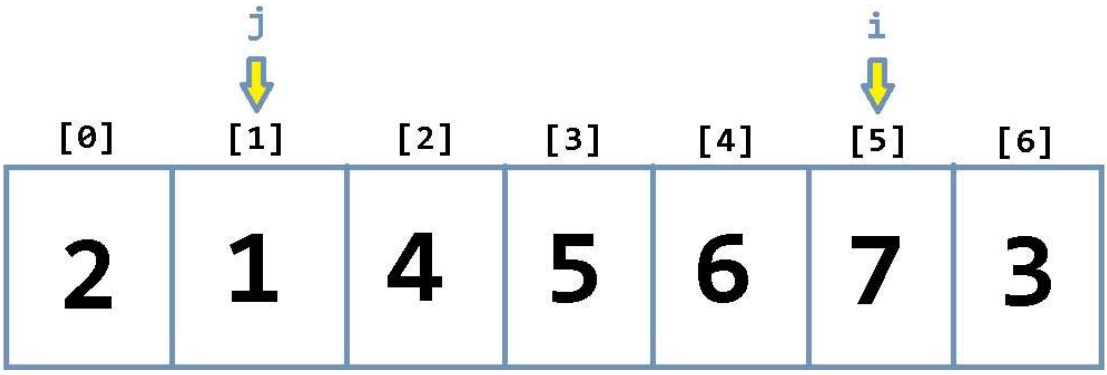
j is decremented



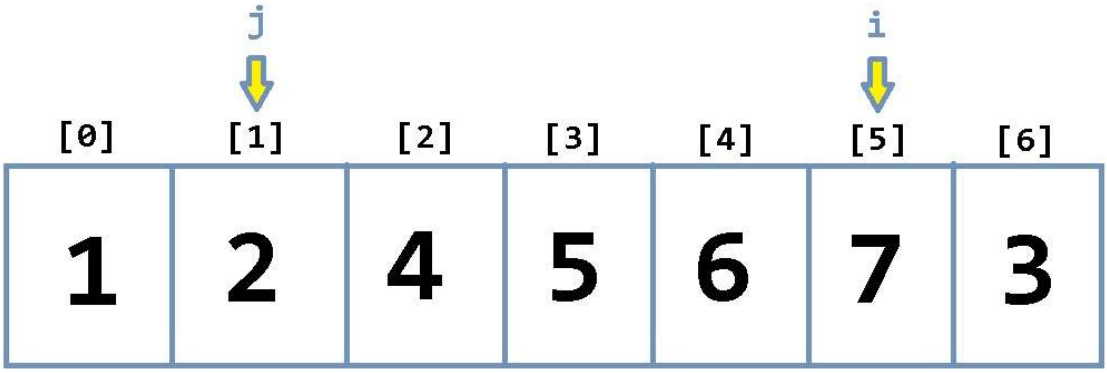
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



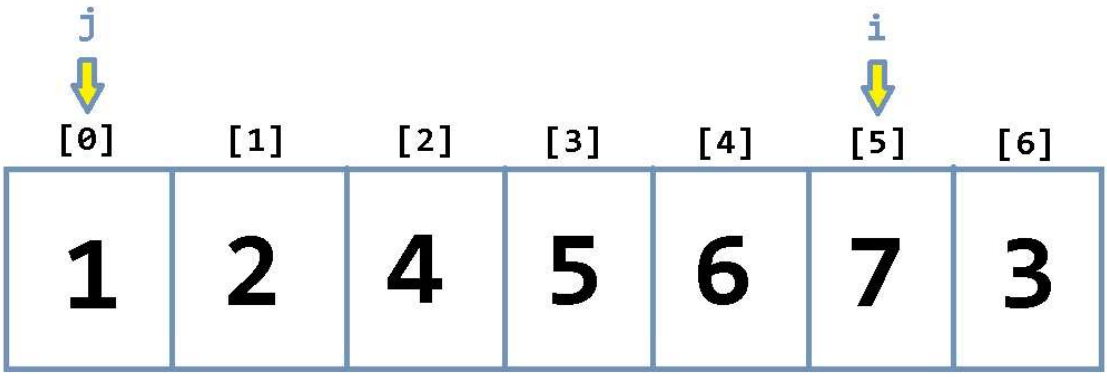
j is decremented



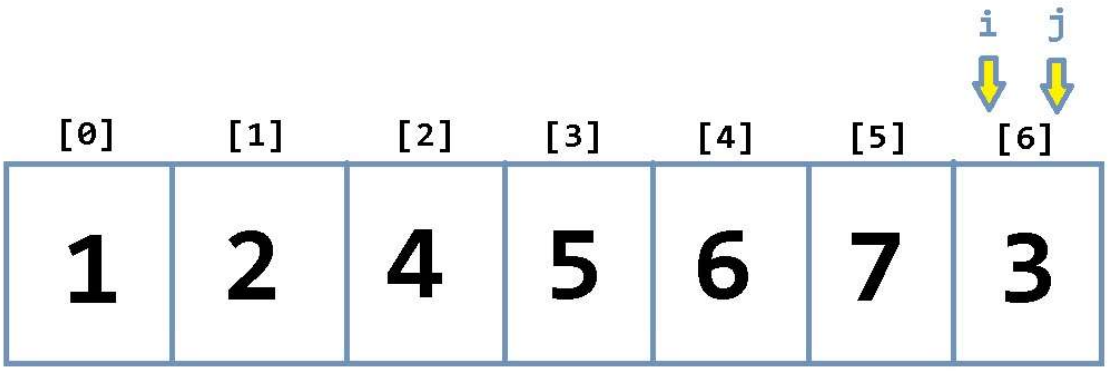
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



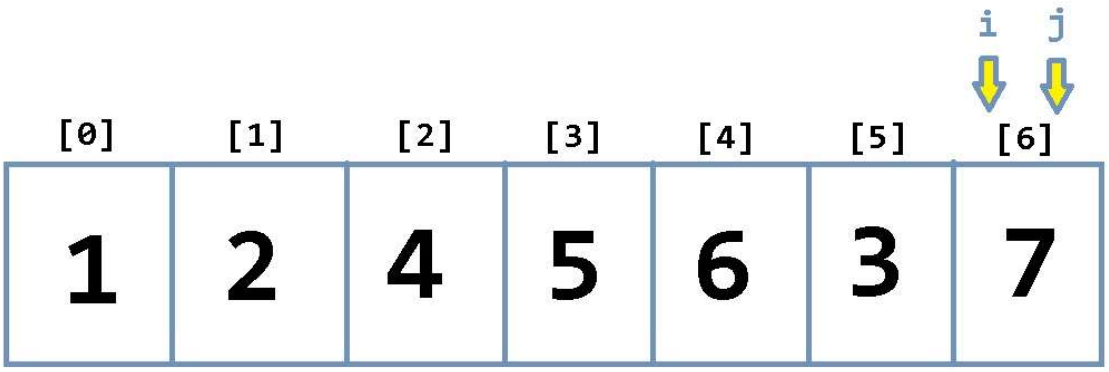
j is decremented, condition for the second loop is no longer met therefore the loop breaks back into the first loop



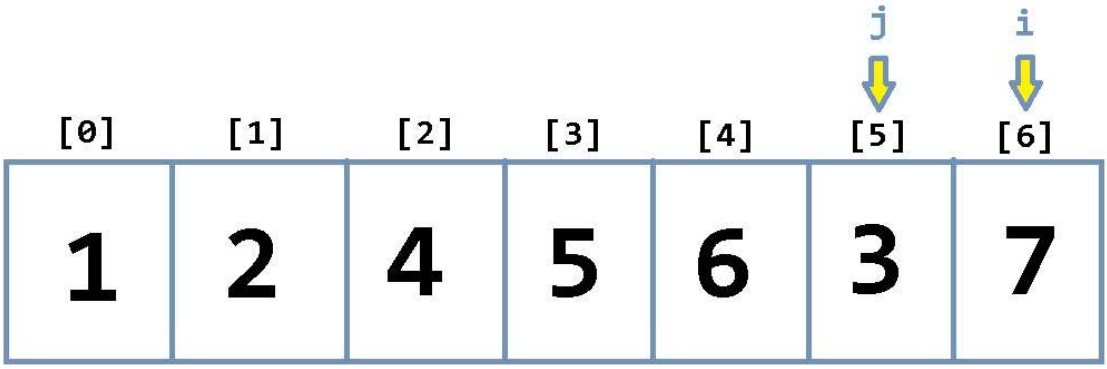
i is incremented, j = i



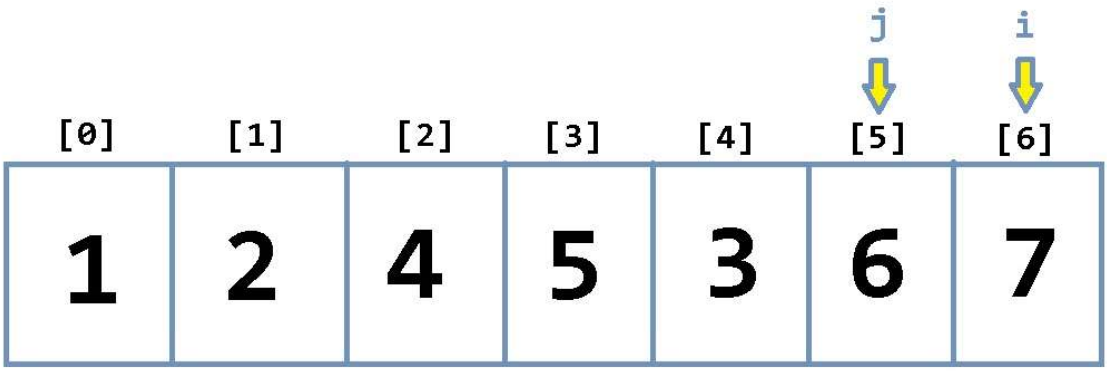
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



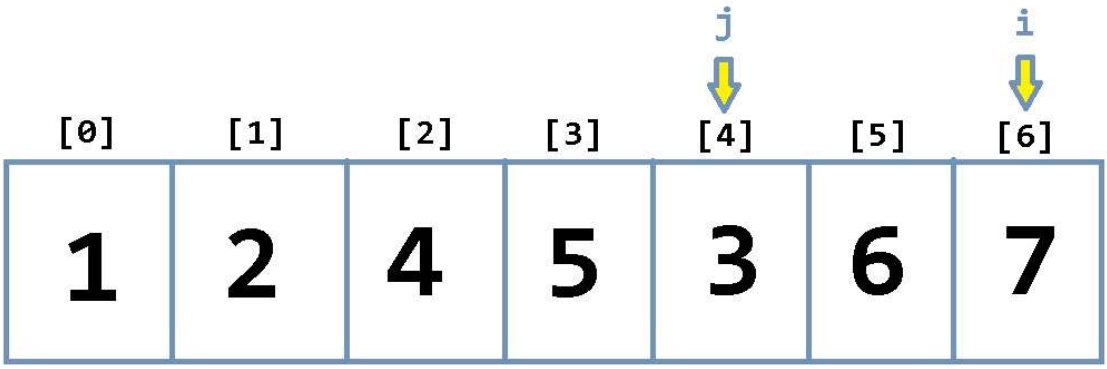
j is decremented



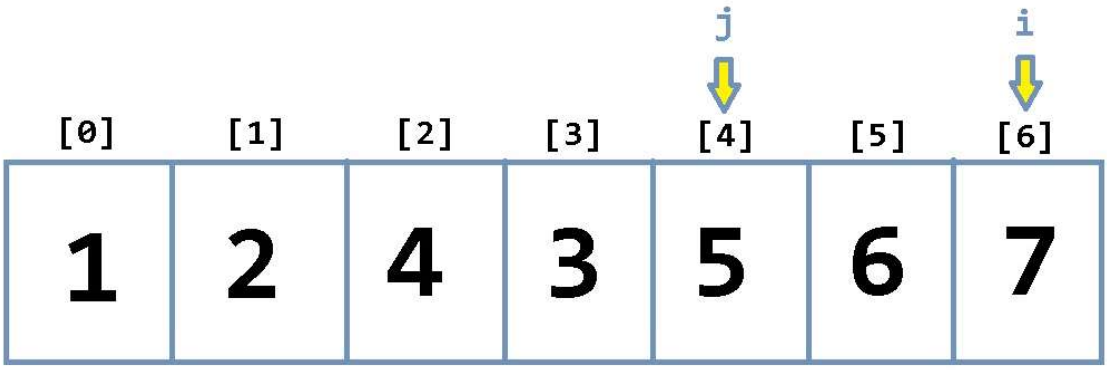
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



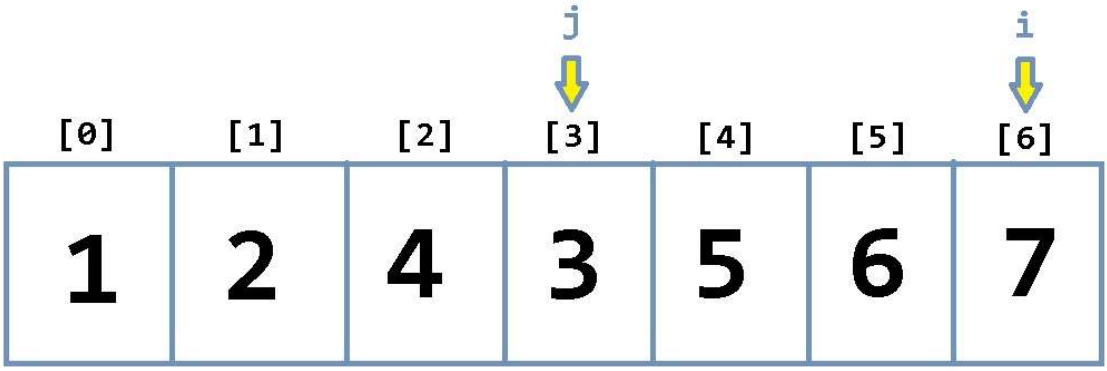
j is decremented



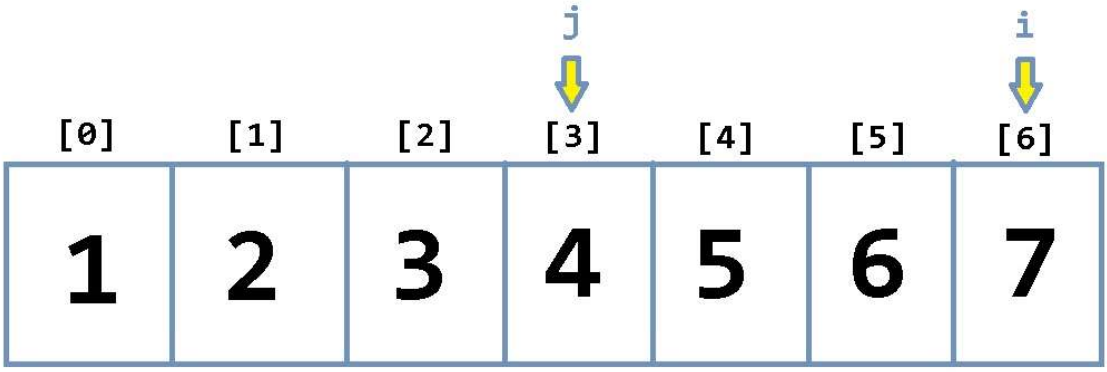
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



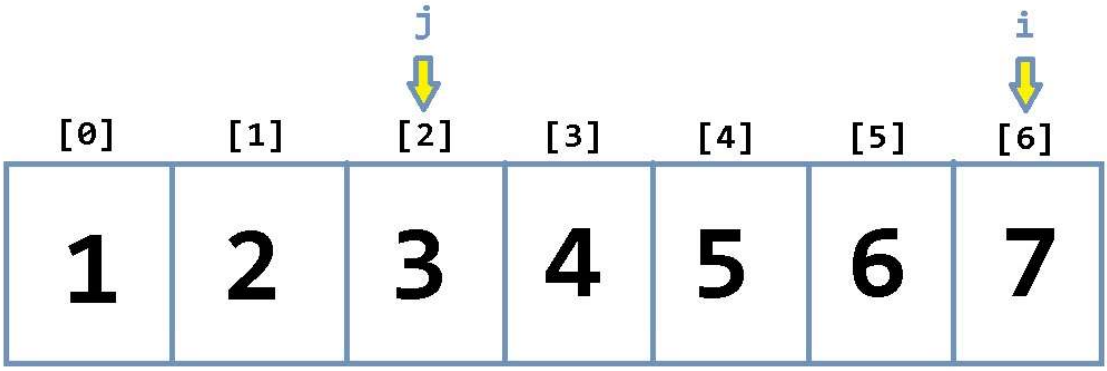
j is decremented



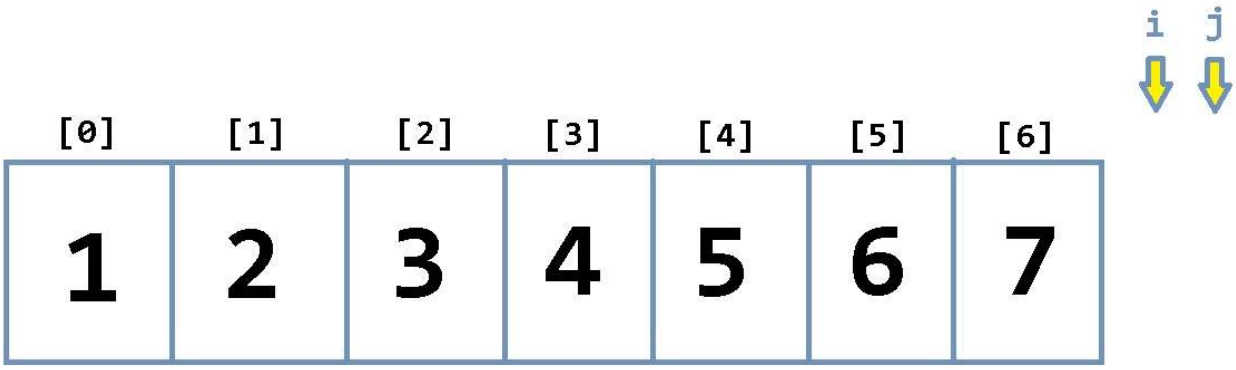
j > 0 AND arr[j] < arr[j-1] are both met so the algorithm jumps into the second loop which performs the swap



j is decremented, condition for the second loop is no longer met therefore the loop breaks back into the first loop



i is incremented, j = i, the first loops condition is no longer met so the loop breaks and the insertion sort algorithm finishes




```
void insertion_sort (int arr[], int length){
                int j, temp;

        for (int i = 0; i < length; i++){
                j = i;

                while (j > 0 && arr[j] < arr[j-1]){
                        temp = arr[j];
                        arr[j] = arr[j-1];
```

```
                        arr[j-1] = temp;
                        j--;
                        }
                }
}
```