

# Stable Vs Unstable Sorts

Let's consider an array in which 5 appears twice:



...it is unstable.

If we rearrange it as a stable sort:



...notice the 5 is ahead of the 5 prime, because if the 5 prime is ahead of the regular 5:



...it becomes unstable because the original order has changed.

In Java, if we strictly use the **greater than** symbol, the order of elements will be stable:

```
public void sort(int[] data) {  
    for (int i = 0; i < data.length; i++) {  
        int current = data[i];  
        int j = i - 1;  
        while (j >= 0 && data[j] > current) {  
            data[j + 1] = data[j];  
            j--;  
        }  
        data[j + 1] = current;  
    }  
}
```

...but if we use **greater than or equal**, the order becomes unstable:

```
public void sort(int[] data) {  
    for (int i = 0; i < data.length; i++) {  
        int current = data[i];  
        int j = i - 1;  
        while (j >= 0 && data[j] >= current) {  
            data[j + 1] = data[j];  
            j--;  
        }  
        data[j + 1] = current;  
    }  
}
```

...even if 5 and 5' appear equal on the outer surface, they might not be sorted in order.


Example:

Let's say we have an unsorted set of data:

Name	Age
John Doe	25
Nancy Cooper	24
Amit Kumar	21
Nancy Cooper	28

...we sort it by age:

Name	Age
John Doe	25
Nancy Cooper	24
Amit Kumar	21
Nancy Cooper	28



Name	Age
Amit Kumar	21
Nancy Cooper	24
John Doe	25
Nancy Cooper	28

If the user clicks on Name to sort by name...

Sorted By Name

Name	Age
Amit Kumar	21
John Doe	25
Nancy Cooper	24
Nancy Cooper	28

Stable Sort

Name	Age
Amit Kumar	21
John Doe	25
Nancy Cooper	28
Nancy Cooper	24

Unstable Sort

...it may be either stable or unstable.

This is an example of when we prefer to have a stable sort in the real world.