

Here is a quicksort algorithm which does not call the method partition. Instead uses the logic inside the code.

```
quicksort(A, start, stop)
```

```
    //Let p be the index of the pivot such that start <= p <= stop
```

```
    swap(A, p, stop)           // swaps A[p] and A[stop]
```

```
    i <- start
```

```
    j <- stop - 1
```

```
    while (i <= j) do
```

```
        while (i < stop & A[i] < A[stop]) i++
```

```
        while (j >= start & A[j] >= A[stop]) j--
```

```
        If (i < j)
```

```
            swap(A, i, j)
```

```
            i++
```

```
            j--
```

```
    swap(A, i, stop)
```

```
    if (start < i - 1) quicksort(A, start, i - 1)    // call only there are more than 1 item
```

```
    if (i + 1 < stop) quicksort(A, i + 1, stop)      // call only there are more than 1 item
```

Most important facts I learned or re-learned?

1. In order to make sure i will never go out of range, keep moving i and j as above. In particular,

```
    while (i < stop & A[i] <= A[stop]) i++
```

```
    while (j >= start & A[j] > A[stop]) j--
```

will create trouble for us.

2. Quicksort will do fine even when there are duplicates.

Best Wishes! As always have fun! That is the best way to learn!!!

Pivots are shown in red. Swapped items are shown in bold.

10 7 1 8 2 **6** 4 3 9 8 **2** (Let p = 5)

10 7 1 8 2 **2** 4 3 9 8 **6**

i j ← j

3 7 1 8 2 2 4 **10** 9 8 **6**

i j

3 **4** 1 8 2 2 **7** 10 9 8 **6**

i → i j

3 4 1 **2** 2 **8** 7 10 9 8 **6**

ji → i

3 4 1 2 2 **6** 7 10 9 8 **8**

3 4 1 2 **2** **6** 7 10 9 8 **8**

i j i → i j

1 4 3 2 **2** **6** 7 **8** 9 8 **10**

j ← ji

1 **2** 3 2 4 **6** 7 **8** 10 8 **9**

J i l j

1 **2** 4 2 **3** **6** 7 **8** 8 10 **9**

i j j i

1 **2** 2 4 **3** **6** 7 **8** 8 **9** **10**

j l i j

1 2 2 **3** 4 6 7 **8**