arr[ ] = {20, 60, 80, 100, $\underset{p}{\underline{140}}$, 180, 1

```
Quicksort(A, p, r)
    if p < r
        q = Partition(A, p, r).
        Quicksort(A, p, q-1)
        Quicksort(A, q+1, r)
```

To sort an entire array A, the initial call is Quicksort(A, 1, A. length).

Partitioning the array –

The key to the algorithm is the Partition ~~procedu~~ procedure, which rearranges the subarray A[p.....r] in place.

```
Partition(A, p, r)
    x = A[r]
    i = p-1
    for j = p to r-1
        if A[j] ≤ x
            i = i+1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i+1.
```

Quick sort

~~24~~	~~9~~	~~29~~	~~14~~	~~19~~	~~27~~

Partition()

arr[] =

20	160	60	180	80	100	140
p					r-1	r

Indexes: 0 1 2 3 4 5 6

low = 0 (p), high = 6 (r), pivot (x) = arr[h] = 140

initialize i = 0-1 = -1

Traverse elements from j = low to high-1

j = 0 : ~~arr~~ since arr[j] <= pivot, do i++
i=0 and swap(arr[i], arr[j])

i = -1+1 = 0
arr[] = {20, 160, 60, 180, 80, 100, 140} // no change as i and j are same

---

j = 1   arr[1] ≤ pivot ~~= 140~~

$160 \le 140$ ×

No change in i and arr[]

---

j = 2 : Since arr[2] ≤ pivot yes

i = 0+1 = 1   swap(arr[i], arr[j])

arr[] = {20, 60, 160, 180, 80, 100, 140}

0 1 2 3 4 5 6

j = 3. arr[3] ≤ pivot

180 ≤ 140 → No

∴ No change

---

j = 4 arr[4] ≤ pivot

80 ≤ 140 yes

i = 1+1 = 2.

swap (arr[i], arr[j])

swap (arr[2], arr[4])

$arr[\,] = \{20, 60, 80, 180, 160, 100, 140\}$

$\quad\quad\quad\;\; 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$

---

j = 5 : arr[5] ≤ 140

100 ≤ 140 yes

i = 2+1 = 3.

swap (arr[i] with arr[j])

(arr[3] with arr[5])

$arr[\,] = \{20, 60, 80, \underline{100}, 160, \underline{180}, 140\}$

$\quad\quad\quad\;\; 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$

Come out of loop as $\underline{j = high - 1}$.

swap A[i+1] with A[r]

A[3+1] with A[6]

A[4] with A[6]

$arr[\,] = \{20, 60, 80, 100, \boxed{140}, \underline{180, 160}\}$.

finally we place pivot at correct position.