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

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

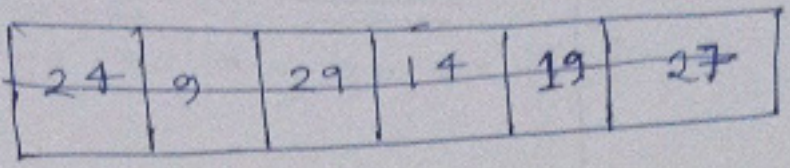
To sort an entire array A, the initial call is Quicksort(A, 1, A.length).

Partitioning the array —

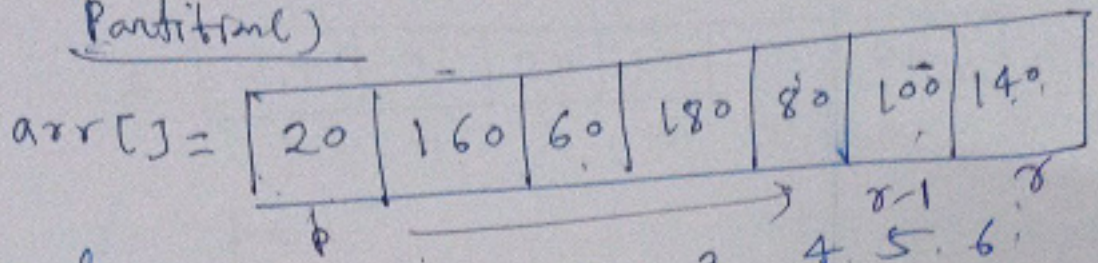
The key to the algorithm is the Partition procedure, which rearranges the subarray A[p.....r] in place.

```
Partition(A, p, r)
1  x = A[r]
2  i = p-1
3  for j = p to r-1
4      if A[j] ≤ x
5          i = i+1
6          exchange A[i] with A[j]
7  exchange A[i+1] with A[r]
8  return i+1.
```

Quick sort



Partition()



low = p = 0, high = r = 6, 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++

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

160 ≤ 140 ×

No chge 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}

0 1 2 3 4 5 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, 100, 160, 180, 140}

0 1 2 3 4 5 6

Come out of loop as 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, 140, 180, 160}.

finally we place pivot at correct position.