4) Naive, Lomuto & Hoare Partition

1) Naive Partition.

⇒ I/p : arr[] = { 3, 8, 6, 12, 10, 7 } $\boxed{P = 5}$
↳ pivot = arr[5] = $\boxed{7}$

O/p : { 3, 6, $\boxed{7}$, 12, 10 }
(OR) Return value = 2.
{ 6, 3, $\boxed{7}$, 12, 10 }

Above is example of partition, i.e pivot is fixed at it correct position.

→ Types
- → Stable → Naive.
- → Unstable → Lomuto, Hoare.

2) Algo for Naive Partition

O(n) → Space & time.

≤ pivot	pivot	> pivot

↳ index of last occurence of pivot.

```
int partition ( int arr[], int low , int high, int p)
{ int temp[high-low+1], index = 0;  ]Greater temp array
  for ( int i=l; i<=h ; i++)          → smaller
  { if (arr[i] < arr[p]) { p[index] = arr[i]; index++; }}

  for (int i=l; i<=h; i++)           → Equal.
  { if (arr[i] == arr[p]) { temp[index]= arr[i]; index++; }}

  int res = l+ index - 1;
```
→ P.T.O.

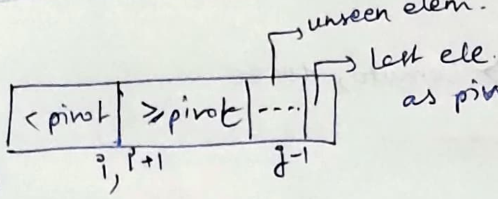
```
for (int i = l; i<=h; i++)
{ if(arr[i] > arr[p]) { temp[index] = arr[i]; index++; } }      ] Last occurrence of pivot

for (int i=l; i<=h; i++)
{ arr[i] = temp[i-l]; }          ] for copying to original array.

return res;

}
```

Lomuto Partition



☆ At last we just swap Last ele and arr[i+1].

O(n) time
O(1) space.

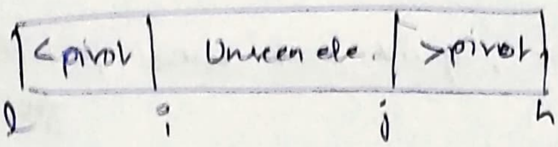
```
int LPartition (int arr[], int l, int h)
{
    int pivot = arr[h];
    int i = l-1;
    for (in j=l; j<=h-1; j++)
    {  if(arr[i] < pivot)
        { l++;
          swap (arr[i], arr[j]);
        }
    }
    swap (arr[i+1], arr[h]); ☆
    return (i+1);
}
```

Hoare Partition

=> Better then Lomuto Partition. ~at least 2x faster~

O(1) → space
O(n) → time.

=> First element taken as pivot

→ 

doesn't put pivot in correct place, just divides left & right array.

=> int partition (int arr[] ; int l , int h)

```
{ int pivot = arr[l];
  int i = l-1, j = h+1;
  while( true)
  { do{ i++ }
    while(arr[i]<pivot)
    do{ j--; }
    while (arr[j]< pivot)
    if( i>=j ) return j;
    swap(arr[i], arr[j]);
  }
}
```