Quick Sort

Quick Sort: a divide and Conquer Sorting Algorithm

- Pick an element, called a *pivot*, from the array.
- ***Partitioning***: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the *partition* operation.
- **Recursively** apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

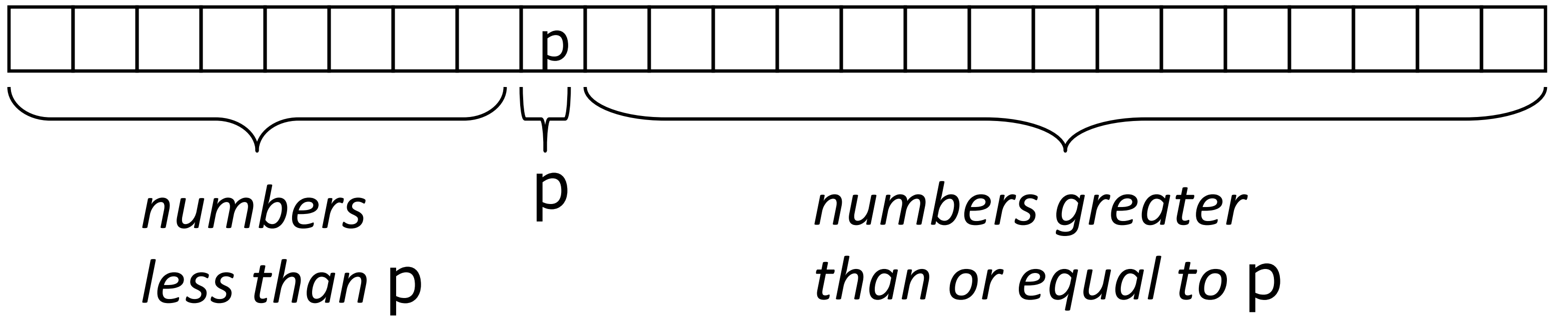
Quick Sort

- To sort a[left...right]:

```
1. if left < right:
    1.1. Partition a[left...right] such that:
            all a[left...p-1] are less than a[p], and
            all a[p+1...right] are >= a[p]
    1.2. Quicksort a[left...p-1]
    1.3. Quicksort a[p+1...right]
2. Terminate
```

Partitioning

- A key step in the Quicksort algorithm is partitioning the array
    - We choose some (any) number p in the array to use as a pivot
    - We partition the array into three parts:



Partitioning

- Choose an array value (say, the first or the last) to use as the pivot
- Starting from the left end, find the first element that is greater than or equal to the pivot
- Searching backward from the right end, find the first element that is less than the pivot
- Interchange (swap) these two elements
- Repeat, searching from where we left off, until done

Partitioning

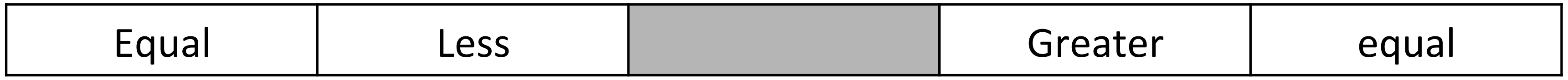
- To partition a[left...right]:

```
1. Set p = a[left], l = left + 1, r = right;
2. while l < r, do
    2.1. while l < right & a[l] < =p, set l = l + 1
    2.2. while r > left & a[r] >= p, set r = r - 1
    2.3. if l < r, swap a[l] and a[r]
3. Set a[left] = a[r], a[r] = p
4. Terminate
```

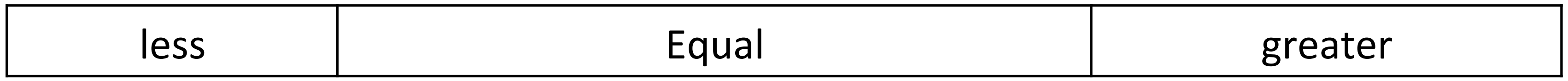
How to handle keys equal to partitioning elements?

  - Move from left to find an element that is not less
  - Move from right to find an element that is not greater
  - Stop if pointers has crossed
  - Exchange
  - If left element equal exchange to left end
  - If right element equal exchange to right end
- Swap equals to center after partition

- Partitioning invariant :



- After Partition



Modify Partitioning

- To partition a[left...right]:

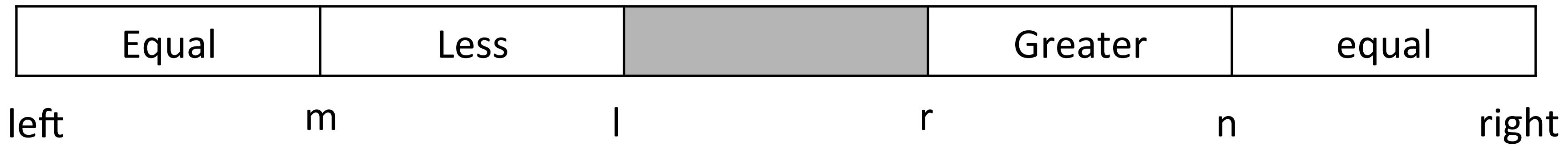
```
1. Set p = a[left], l = left + 1, r = right;
2. while l < r, do
    2.1. while l < right & a[l] < p, set l = l + 1
    2.2. while r > left & a[r] >= p, set r = r - 1
    2.3. if l < r, swap a[l] and a[r]
3. Set a[left] = a[r], a[r] = p
4. Terminate
```

Pseudo code from : https://www.cs.princeton.edu/~rs/talks/QuicksortIsOptimal.pdf

Modify Partitioning

- To partition a[left…right]:

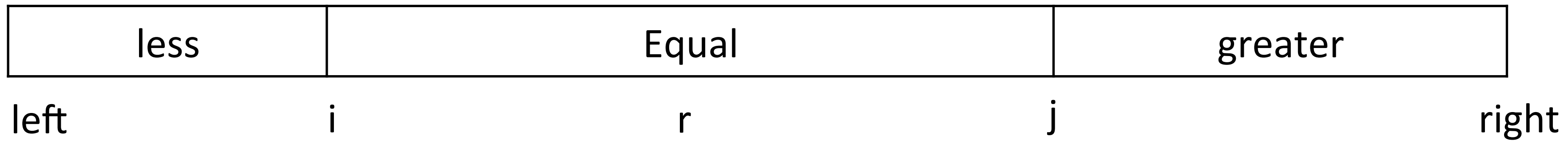
```
1. Set p = a[left], l = left + 1, r = right, m= left-1, n=right;
2. while l < r, do
    2.1. while l < right & a[l] < p, set l = l + 1
    2.2. while r > left & a[r] > p, set r = r - 1
    2.3. if l < r, swap (a[l] and a[r])
    2.4. if (a[l]==p) {m++; swap(a[m], a[l])}
    2.5 if (a[r]==p){n--; swap(a[n],a[r]) }
3. Set a[left] = a[r], a[r] = p
```



Modify Partitioning

- To partition a[left...right]:

```
1. Set p = a[left], l = left + 1, r = right, m= left, n=right+1;
2. while l < r, do
    2.1. while l < right & a[l] < p, set l = l + 1
    2.2. while r > left & a[r] > p, set r = r - 1
    2.3. if l < r, swap (a[l] and a[r])
    2.4. if (a[l]==p) {m++; swap(a[m], a[l])}
    2.5 if (a[r]==p){n--; swap(a[n],a[r]) }
3. Set a[left] = a[r], a[r] = p, i=r-1, j=r+1;
4.for (k=left+1; k<=m;k++, i--) swap(a[k], a[i])
5.for (k=right; k>=n; k--, j++) swap (a[k], a[j])
```


less
Equal
greater
left
i
r
j
right