Quick Sort

Quick Sort

We use Divide-and-Conquer:

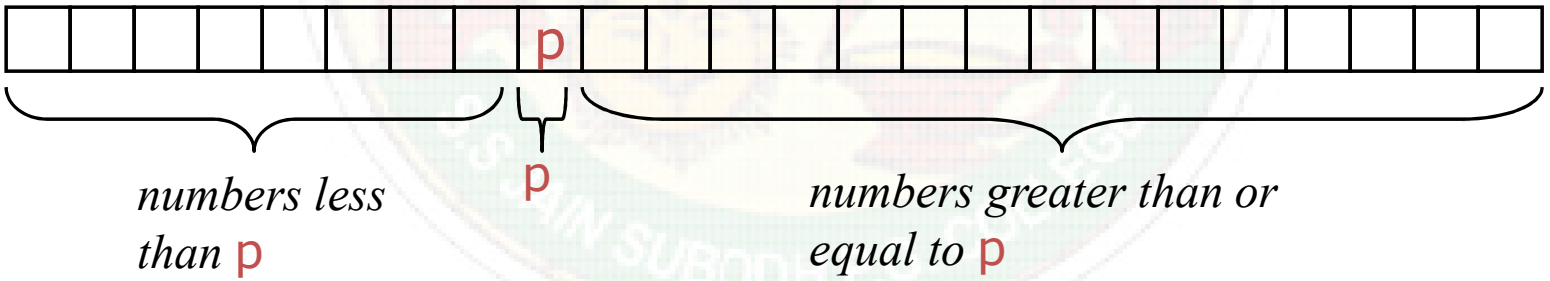
1. Divide: partition $A[p..r]$ into two subarrays $A[p..q-1]$ and $A[q+1..r]$ such that each element of $A[p..q-1]$ is $\leq A[q]$, and each element of $A[q+1..r]$ is $\geq A[q]$. Compute q as part of this partitioning.
2. Conquer: sort the subarrays $A[p..q-1]$ and $A[q+1..r]$ by recursive calls to QUICKSORT.
3. Combine: the partitioning and recursive sorting leave us with a sorted $A[p..r]$ – no work needed here.

An obvious difference between Merge sort and Quick sort is that we do most of the work in the divide stage, with no work at the combine one.

Partitioning in Quicksort

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:

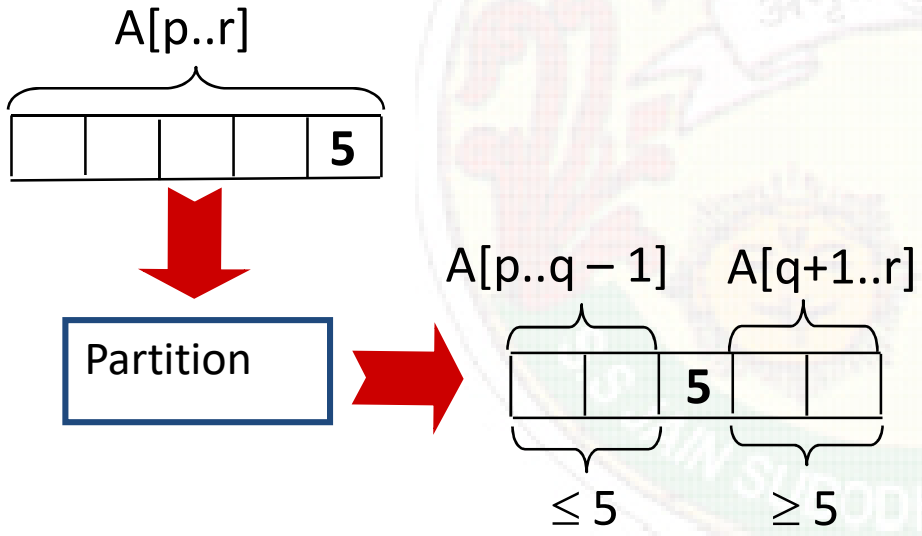


Quick Sort

The Pseudo-Code

```
Quicksort(A, p, r)
    if p < r then
        q := Partition(A, p, r);
        Quicksort(A, p, q – 1);
        Quicksort(A, q + 1, r)
```

```
Partition(A, p, r)
    x := A[r], i:= p – 1;
    for j := p to r – 1 do
        if A[j] ≤ x then
            i := i + 1;
            A[i] ↔ A[j]
    A[i + 1] ↔ A[r];
    return i + 1
```



Example

p r

initially: 2 5 8 3 9 4 1 7 10 **6** **note:** pivot (x) = 6

i j

next iteration: 2 5 8 3 9 4 1 7 10 **6**

i j

next iteration: 2 5 8 3 9 4 1 7 10 **6**

i j

next iteration: 2 5 8 3 9 4 1 7 10 **6**

i j

next iteration: 2 5 3 8 9 4 1 7 10 **6**

i j

```
Partition(A, p, r)
    x, i := A[r], p – 1;
    for j := p to r – 1 do
        if A[j] ≤ x then
            i := i + 1;
            A[i] ↔ A[j]
    A[i + 1] ↔ A[r];
    return i + 1
```

Example (Continued)

next iteration:	2	5	3	8	9	4	1	7	10	**6**
			i		j					
next iteration:	2	5	3	8	9	4	1	7	10	**6**
			i			j				
next iteration:	2	5	3	4	9	8	1	7	10	**6**
				i			j			
next iteration:	2	5	3	4	1	8	9	7	10	**6**
					i			j		
next iteration:	2	5	3	4	1	8	9	7	10	**6**
					i				j	
after final swap:	2	5	3	4	1	**6**	9	7	10	8
					i					j

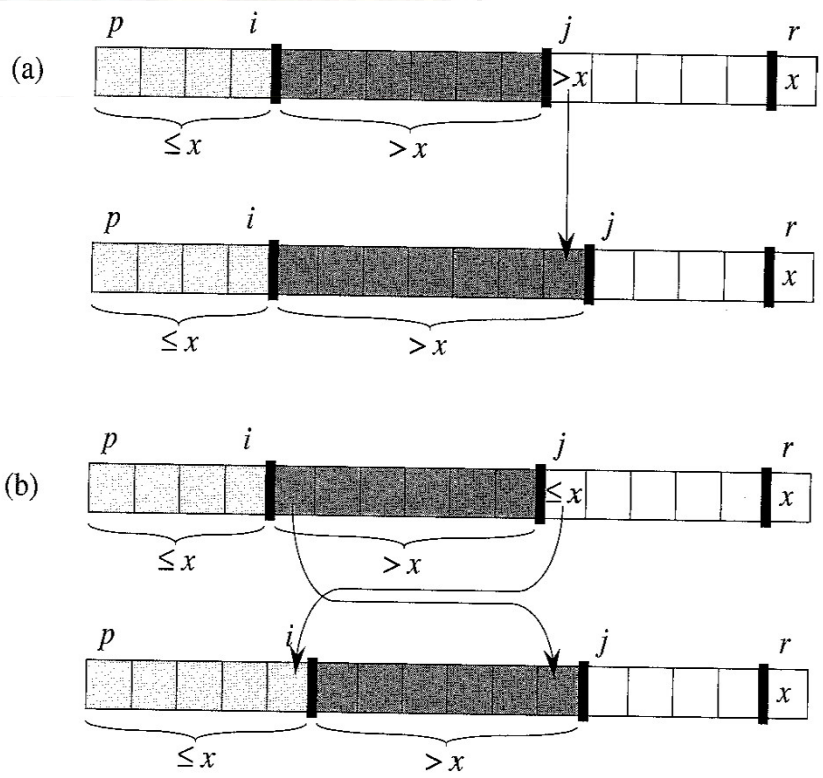
```
Partition(A, p, r)
    x, i := A[r], p – 1;
    for j := p to r – 1 do
        if A[j] ≤ x then
            i := i + 1;
            A[i] ↔ A[j]
    A[i + 1] ↔ A[r];
    return i + 1
```

Partitioning

- Select the last element A[*r*] in the subarray *A*[*p*..*r*] as the *pivot* – the element around which to partition.
- As the procedure executes, the array is partitioned into four (possibly empty) regions.
  1. *A*[*p*..*i*] — All entries in this region are **< *pivot***.
  2. *A*[*i*+1..*j* – 1] — All entries in this region are **> *pivot***.
  3. *A*[*r*] = *pivot*.
  4. *A*[*j*..*r* – 1] — Not known how they compare to *pivot*.
- The above hold before each iteration of the *for* loop, and constitute a *loop invariant.* (4 is not part of the loop.)

Quick Sort

- **Initialization**. Before the first iteration: $i=p-1$, $j=p$. No values between p and i; no values between $i+1$ and $j-1$. The first two conditions are trivially satisfied; the initial assignment satisfies 3.
- **Maintenance**. Two cases
  - Case a: A[j]>x
  - Case b :A[j] ≤ x



Correctness of Partition

- **Termination:**
  - When the loop terminates, $j = r$, so all elements in A are partitioned into one of the three cases:
    - $A[p..i] \leq$ ***pivot***
    - $A[i+1..j-1] >$ ***pivot***
    - $A[r] =$ ***pivot***
- The last two lines swap $A[i+1]$ and $A[r]$.
  - *Pivot* moves from the end of the array to between the two subarrays.
  - Thus, procedure *partition* correctly performs the divide step.

Quicksort Overview

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

Thanks