Sorting
QuickSort

Debswapna Bhattacharya, PhD
Assistant Professor
Computer Science and Software Engineering
Auburn University

QuickSort is a Recursive Divide & Conquer Algorithm

It is the fastest general purpose sorting algorithm for large inputs

```
QUICKSORT(A,p,r)
if p < r
then q = PARTITION(A,p,r)
        QUICKSORT(A,p,q-1)
        QUICKSORT(A,q+1,r)
```

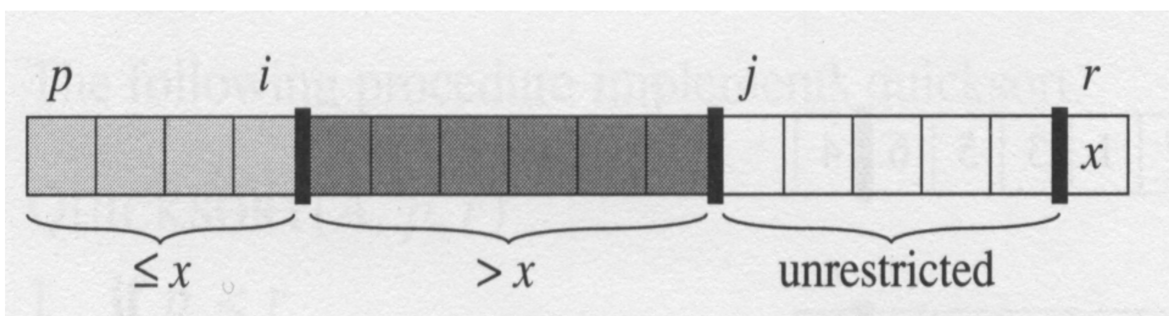
Partition

- What does Partition do?
- Why?
- Is it correct?
- How efficient is it?

Partition

with index i=p-1, and index j between p and r-1, for any array index k,

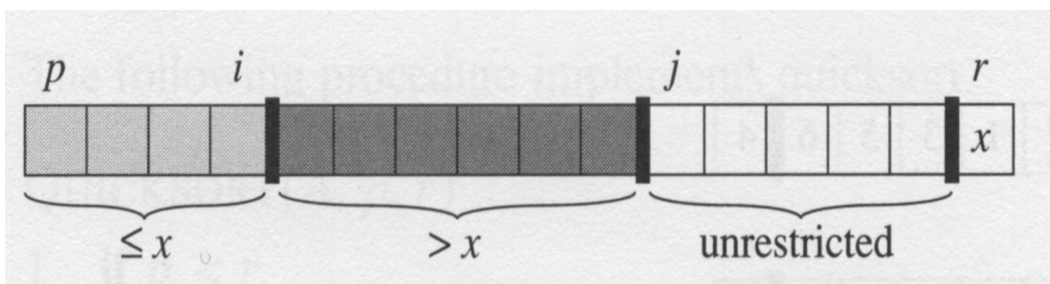
1. if $p \leq k \leq i$, then $A[k] \leq x$.
2. if $i + 1 \leq k \leq j - 1$, then $A[k] > x$.
3. if $k = r$, then $A[k] = x$.

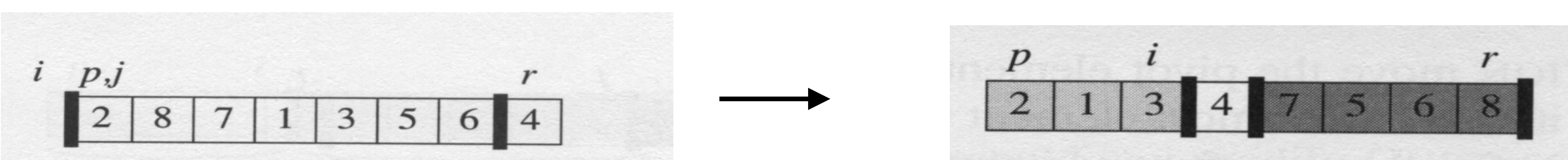


Partition

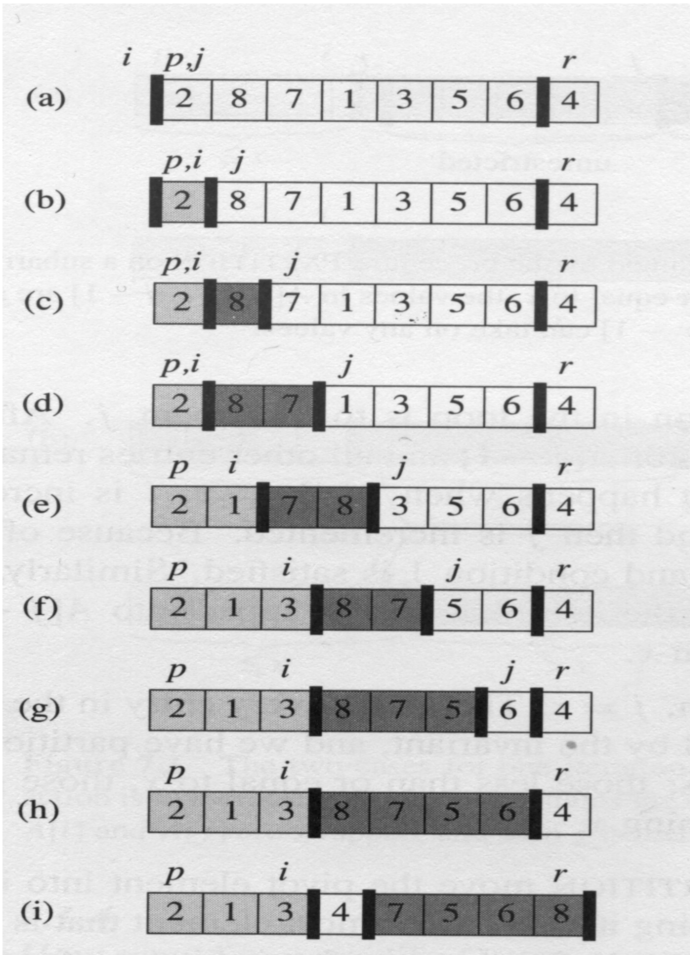
with index i=p-1, and index j between p and r-1, for any array index k,

1. if p ≤ k ≤ i, then A[k] ≤ x.
2. if i + 1 ≤ k ≤ j -1, then A[k] > x.
3. if k = r, then A[k] = x.





The operation of *Partition* on a sample array




```
x = A[r]
i  = p – 1
for j = p to r -1
      if A[j] ≤ x
            then i = i + 1
            swap A[i] and A[j]
swap A[i +1] and A[r]
return  i +1
```

Partition(A, p, r)

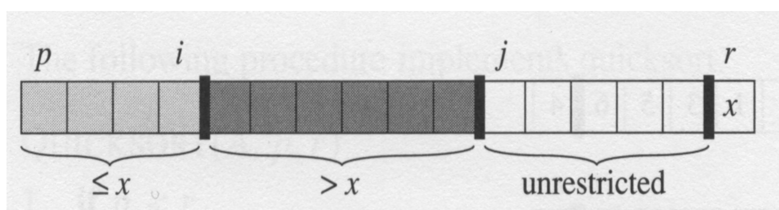
```
x = A[r]
i  = p – 1
for j = p to r -1
    if A[j] ≤ x
        then i = i + 1
        swap A[i] and A[j]
swap A[i +1] and A[r]
return  i +1
```

Complexity:
Partition on A[p…r] is $\Theta(n)$
where $n = r - p + 1$

Correctness of Partition: Loop Invariant

At the beginning of any iteration of the loop of lines 3-6 with a j value between p and r-1, for any array index k,

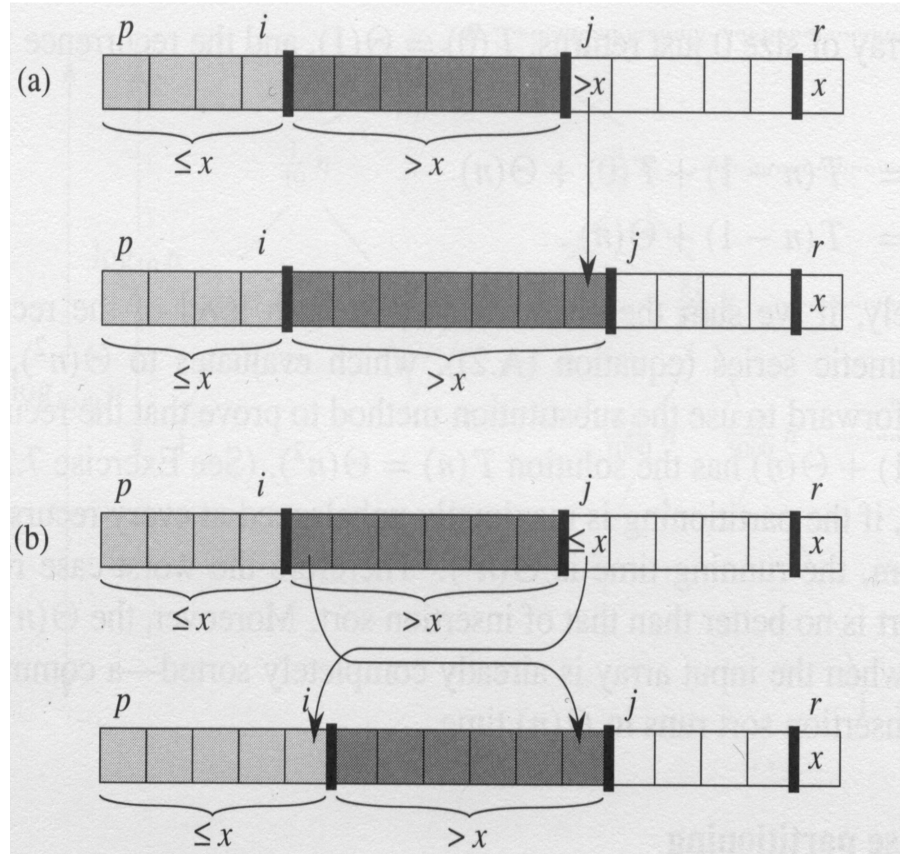
1. if $p \le k \le i$, then $A[k] \le x$.
2. if $i + 1 \le k \le j - 1$, then $A[k] > x$.
3. if $k = r$, then $A[k] = x$.



Thinking Assignment: Satisfy yourself that LI is true at initialization

Why the Loop Invariant is Maintained

Only two cases possible for what happens to A[j] in any one iteration of procedure *Partition*



Thinking Assignment: What is the state of the array at termination?

Thinking Assignment: Write the complete Loop Invariant proof of correctness of Partition yourself before reading p.173

Thinking Assignments

1. What is the "strategy" employed by Partition?
2. This strategy is a highly efficient one to employ for rearranging data in an array.
3. Modify the Partition algorithm to move all 0's in an array to its left or right end.
4. Can you modify the Partition algorithm to move all –ve numbers in an array to the left, all +ve numbers to the right and all 0's to the middle?

How efficient is QuickSort?

- Recursive algorithm
- So to answer this question we must determine the recurrences of the algorithm

QuickSort Recurrences

- When $n \geq 2$, T(n) =
  T(size of left partition) + T(size of right partition) + 24n + 3
  = T(size of left partition) + T(size of right partition) + 24n
  {3 can be ignored since as n increases, c'n>>c'}
- When $n<2$, T(n) = 3
- Simplify the two recurrences by using larger of the two constants in both recurrences:
  T(n)= T(size of left partition) + T(size of right partition) + 24n, $n \geq 2$
  T(n)=3, $n<2$
- What are the possibilities for the partition sizes?

QuickSort Recurrences

- Left partition (or right partition) could be empty if A[r] happens to be the smallest (or largest) number in A.
- So $T(n) = T(0) + T(n-1) + cn$; $T(1) = c$
- You can easily show by backward/forward substitution method (**thinking assignment**: do this as an exercise to improve your analytic skills) that these recurrences have the solution $T(n) = \Theta(n^2)$
- This is the worst case partitioning!
- **Thinking assignment**: Can you think of an input that will produce this kind of partition in <u>every</u> recursive call?

QuickSort Recurrences

- Partitioning can also divide the array equally: one partition of size floor(n/2) and the other of size ceiling(n/2)-1
- $T(n) = 2T(n/2) + cn; T(1) = c$
- You can easily show by applying the master method (**thinking assignment**: do this as an exercise to improve your analytic skills) that if $T(n) = 2T(n/2) + cn$ then $T(n) = \Theta(n\lg n)$.
- **Thinking assignment**: Can you think of an input that will produce this kind of partition in <u>every</u> recursive call?
- This is the best case partitioning. In fact, the split doesn't have to be 50-50. This complexity holds whenever the split is of constant proportionality.

Average Case Performance

- Good and bad splits tend to balance out in practice (see p. 176 of text)

- So the average performance of quicksort is O(nlgn) (see p.177-178 of text)

- To get this balance, in practice we don't pick A[r] as the pivot; instead a median-of-three approach is used to pick the pivot in practice.

Median-of-Three Pivot Picking

```
Median-of-Three-Partition (A,p,r)
first=A[p]
m=floor((p+r)/2)
middle=A[m]
last=A[r]
Median-of-Three=median(first,middle,last)
if Median-of-Three≠last then
        if Median-of-Three=first then index=p else index=m
        swap A[r] and A[index]
return Partition(A,p,r)
```

Now modify the QuickSort algorithm to call Median-of-Three-Partition (A,p,r) in step 2 instead.

Random Sampling

- Another way to make sure of random distribution of good and bad splits is to choose randomly so that any of the r-p+1 elements in the array has an equal chance of being picked.

Randomized Quicksort

Randomized-Partition (A,p,r)

1. i=Random(p,r)
2. swap A[r] and A[i]
3. return Partition(A,p,r)

Now modify the QuickSort algorithm to call Randomized-Partition (A,p,r) in step 2 instead

Thinking Assignments

Quicksort can be modified to obtain an elegant and efficient linear O(n) algorithm **QuickSelect** for the selection problem.

Quickselect(A, p, r, k)

{p & r – starting and ending indexes of array A; goal is to find k-th smallest number in non-empty array A; 1≤k≤(r-p+1)}

```
if p=r then return A[p]
else
    q=Partition(A,p,r)
    pivotDistance=q-p+1
    if k=pivotDistance then
        return A[q]
    else if k<pivotDistance then
        return Quickselect(A,p,q—1,k)
    else
        return Quickselect(A,q+1,r, k-pivotDistance)
```

Thinking Assignments

1. Understand how QuickSelect works by drawing a Recursion Tree for a specific input.
2. Develop its recurrences, assuming as in the case of QuickSort that Partition divides the array evenly.
3. Solve the recurrences to show that it is a linear algorithm (it is the second fastest algorithm to solve the selection problem).

Ch. 7 Reading Assignments

Read 7.1-7.3

Omit 7.4

Ch. 7 Thinking Assignments

Exercises: 7.1-1:7.1-4,

7.2-2 & 7.2-3,

7.3-2