Quicksort

Chapter 7

Read 7.1-7.3

Omit 7.4

7.1 Description of quicksort

- *Divide*
- *Conquer*
- *Combine*

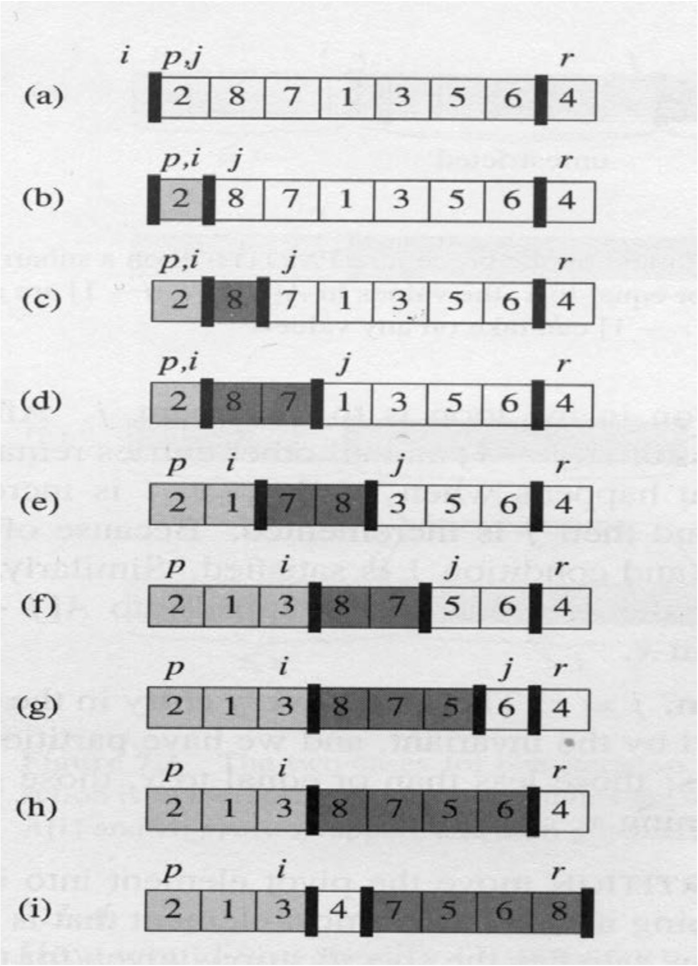
```
QUICKSORT(A,p,r)
if   p < r
then   q = PARTITION(A, p, r)
           QUICKSORT(A,p,q-1)
           QUICKSORT(A,q+1,r)
```

Partition(A, p, r)

```
x = A[r]
i  = p – 1
for j = p to r -1
    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 $n = r - p + 1$

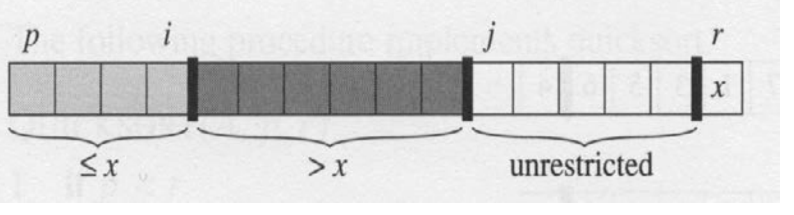
The operation of *Partition* on a sample array



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,

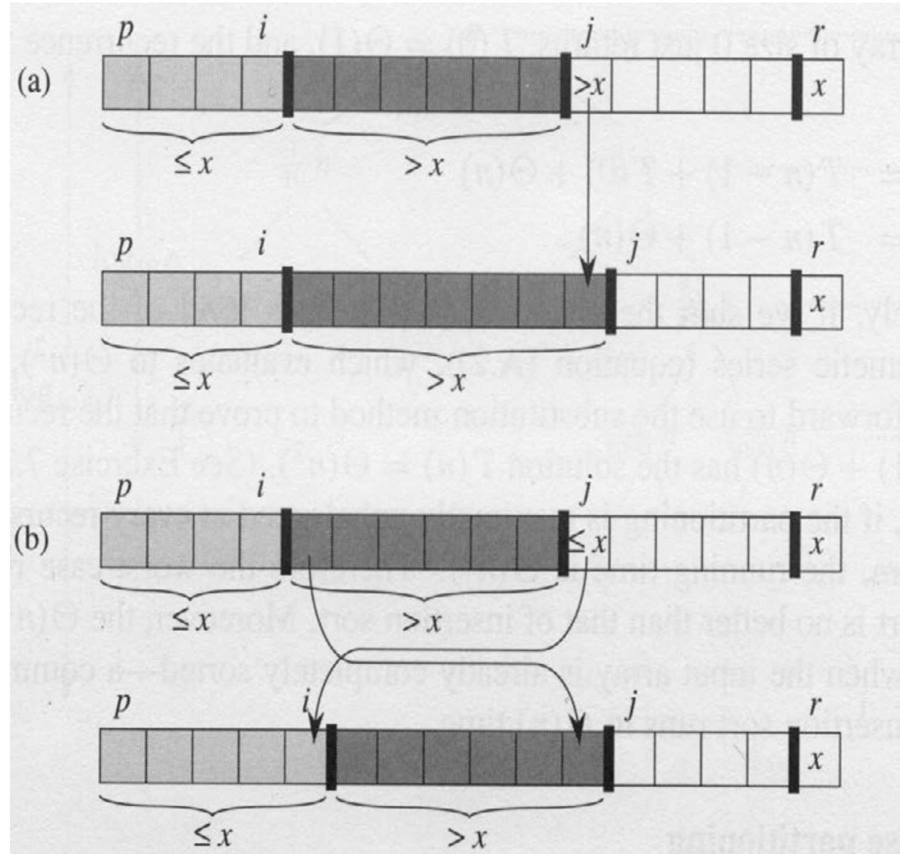
1. if $p \leq k \leq i$, then $A[k] \leq x$.
2. if $i + 1 \leq k \leq j - 1$, then $A[k] > x$.
3. if $k = r$, then $A[k] = x$.



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

How efficient is quicksort?

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

Quicksort Recurrences

- $T(n) = T(\text{size of left partition}) + T(\text{size of right partition}) + \Theta(n)$
- $T(1) = \Theta(1)$

- What are the possibilities for the partition sizes?

Quicksort Recurrences

- $T(n) = T(0) + T(n-1) + \Theta(n)$
  $= T(n-1) + \Theta(n)$
  $= T(n-1) + cn$
- $T(1) = \Theta(1) = c$

- You can easily show by backward substitution method (do this as an exercise to improve your skills) that these recurrences have the solution $T(n) = \Theta(n^2)$

- This is the worst case partitioning!
- Can you think of an input that will produce this kind of partition in every 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

- $T(n) \le 2T(n/2) + \Theta(n)$
- $T(1) = \Theta(1)$

- You can easily show by applying the master method (do this as an exercise to improve your skills) that if $T(n) = 2T(n/2) + \Theta(n)$ then $T(n) = \Theta(n\lg n)$. So in this case Quicksort is $O(n\lg n)$

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

Median-of-Three Pivot Picking

Median-of-Three-Partition (A,p,r)

1 first=A[p]

2 m=floor((p+r)/2)

3 middle=A[m]

4 last=A[r]

5 Median-of-Three=median(first,middle,last)

6 if Median-of-Three≠last then

  7 if Median-of-Three=first then index=p else index=m

  8 swap A[r] and A[index]

9 return Partition(A,p,r)

- Modify the quicksort algorithm to call this partition procedure 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)

- Modify the quicksort algorithm to call this partition procedure 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; 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 to show that it is $O(n)$ using any method