Chapter 7: Quicksort

Quicksort is a divide-and-conquer sorting algorithm in which division is dynamically carried out (as opposed to static division in Mergesort).

The three steps of Quicksort are as follows:

Divide: Rearrange the elements and split the array into two subarrays and an element in between such that so that each element in the left subarray is less than or equal the middle element and each element in the right subarray is greater than the middle element.

Conquer: Recursively sort the two subarrays.

Combine: None.

The Algorithm

Quicksort(A, n)

1: **Quicksort**$'(A, 1, n)$

Quicksort$'(A, p, r)$

1: **if** $p \geq r$ **then return**
2: $q =$ **Partition**(A, p, r)
3: **Quicksort**$'(A, p, q-1)$
4: **Quicksort**$'(A, q+1, r)$

The subroutine Partition

Given a subarray $A[p..r]$ such that $p \leq r-1$, this subroutine rearranges the input subarray into two subarrays, $A[p..q-1]$ and $A[q+1..r]$, so that

- each element in $A[p..q-1]$ is less than or equal to $A[q]$ and
- each element in $A[q+1..r]$ is greater than or equal to $A[q]$

Then the subroutine outputs the value of q.

Use the initial value of $A[r]$ as the "pivot," in the sense that the keys are compared against it. Scan the keys $A[p..r-1]$ from left to right and flush to the left all the keys that are greater than or equal to the pivot.

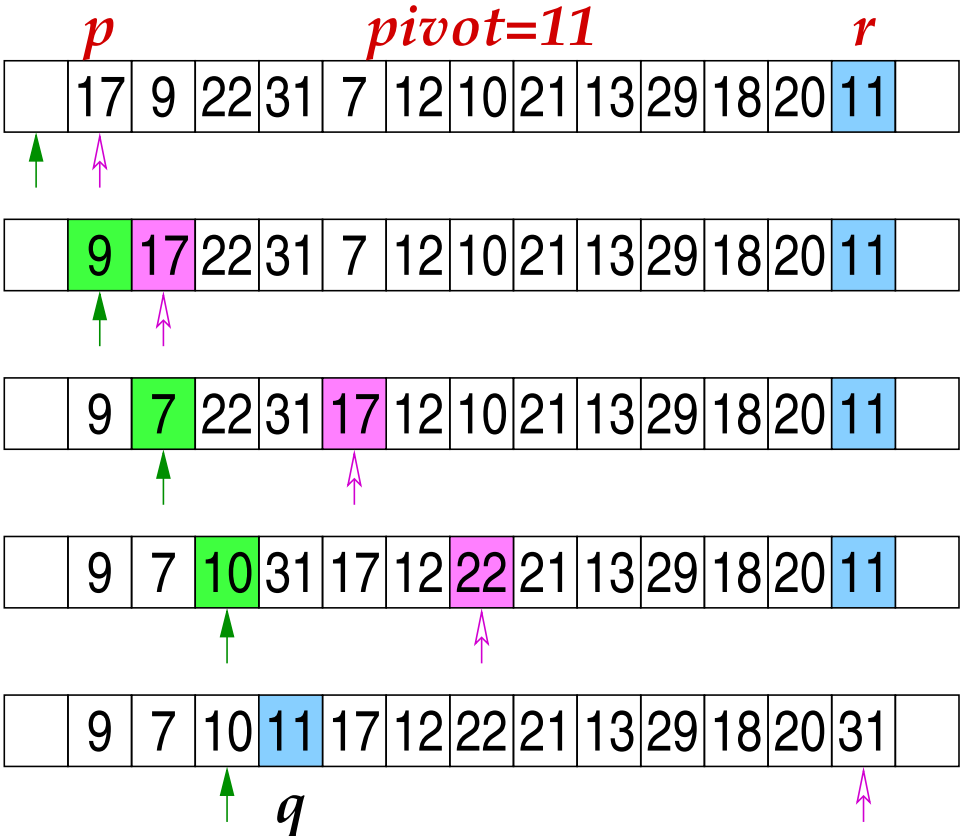
The Algorithm

Partition(A, p, r)

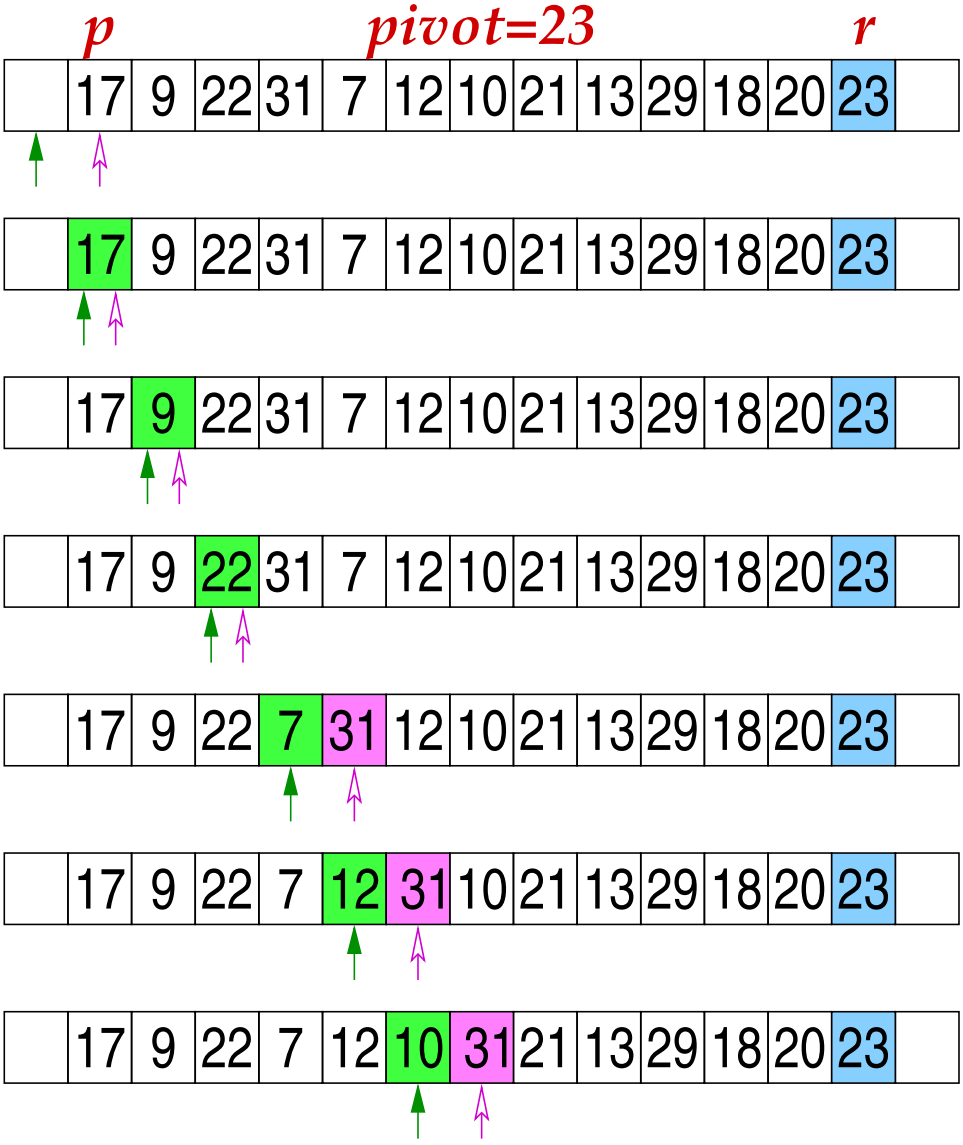
```
 1: x = A[r]
 2: i ← p − 1
 3: for j ← p to r − 1 do
 4:     if A[j] ≤ x then {
 5:         i ← i + 1
 6:         Exchange A[i] and A[j] }
 7: Exchange A[i + 1] and A[r]
 8: return i + 1
```

During the for-loop $i + 1$ is the position at which the next key that is greater than or equal to the pivot should go to.

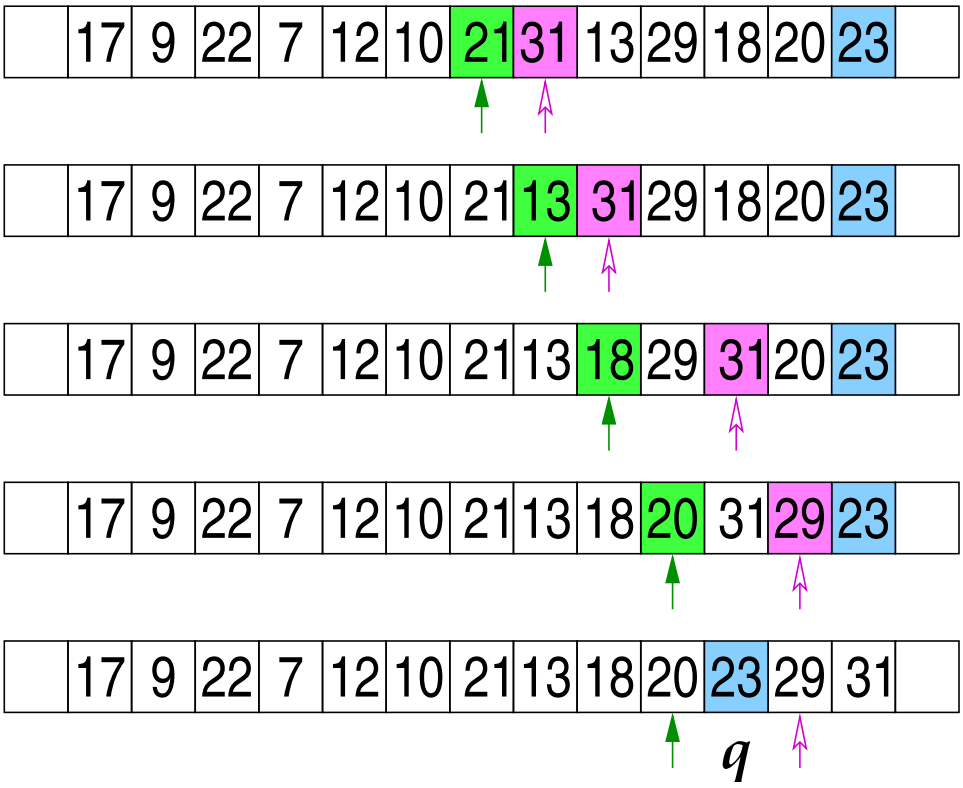
An Example:


p
pivot=11
r
17 9 22 31 7 12 10 21 13 29 18 20 11
9 17 22 31 7 12 10 21 13 29 18 20 11
9 7 22 31 17 12 10 21 13 29 18 20 11
9 7 10 31 17 12 22 21 13 29 18 20 11
9 7 10 11 17 12 22 21 13 29 18 20 31
q

Another Example:



Another Example (cont'd):



Proving Correctness of Partition

Let (A, p, r) be any input to **Partition** and let q be the output of **Partition** on this input. Suppose $1 \leq p < r$. Let $x = A[r]$. We will prove the correctness using loop invariant. The loop invariant we use is: at the beginning of the for-loop, for all k, $p \leq k \leq r$, the following properties hold:

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 $A[k] = x$.

Initialization

The initial value of i is $p-1$ and the initial value of j is p. So, there is no k such $p \leq k \leq i$ and there is no k such that $i+1 \leq k \leq j-1$. Thus, the first conditions are met. The initial value of $A[r] = x$, is so the last one is met.

Maintenance

Suppose that the three conditions are met at the beginning and that $j \leq r - 1$.

Suppose that $A[j] > x$. The value of i will not be changed, so (1) holds. The value of j becomes $j + 1$. Since $A[j] > x$, (2) will for the new value of j. Also, $A[r]$ is unchanged so (3) holds.

Suppose that $A[j] \leq x$. Then $A[i + 1]$ and $A[j]$ will be exchanged. By (2), $A[i + 1] > x$. So, after exchange $A[i + 1] \leq x$ and $A[j] > x$. Both i and j will be incremented by 1, so (1) and (2) will be preserved. Again (3) still holds.

Termination

At the end, $j = r$. So, for all k, $1 \leq k \leq i$, $A[k] \leq x$ and for all k, $i + 1 \leq k \leq r - 1$, $A[k] > x$.

Running Time Analysis

The running time of quicksort is a linear function of the array size, $r - p + 1$, and the distance of q from p, $q - p$. This is $\Theta(r - p + 1)$.

What are the worst cases of this algorithm?

Worst-case analysis

Let T be the worst-case running time of Quicksort. Then

$$T(n) = T(1) + T(n-1) + \Omega(n).$$

By unrolling the recursion we have

$$T(n) = nT(1) + \Omega(\sum_{i=2}^{n} n).$$

Since $T(1) = O(1)$, we have

$$T(n) = \Omega(n^2).$$

Thus, we have:

Theorem A The worst-case running time of Quicksort is $\Omega(n^2)$.

Since each element belongs to a region in which **Partition** is carried out at most n times, we have:

Theorem B The worst-case running time of Quicksort is $O(n^2)$.

The Best Cases

The best cases are when the array is split half and half. Then each element belongs to a region in which **Partition** is carried out at most $\lceil \log n \rceil$ times, so it's $O(n \log n)$.

Randomized-Quicksort

The idea is to **turn pessimistic cases into good cases by picking up the pivot randomly.**

We add the following two lines at the beginning of the algorithm:

-2: Pick $t \in [p, r]$ under the uniform distribution
-1: Exchange $A[r]$ and $A[t]$

Expected Running Time of Randomized-Quicksort

Let n be the size of the input array. Suppose that the elements are pairwise distinct.*

Let $T(n)$ be the expected running time of Randomized-Quicksort on inputs of size n. By convention, let $T(0) = 0$.

Let x be the pivot. Note that the size of the left subarray after partitioning is the rank of x minus 1.

*A more involved analysis is required if this condition is removed.

Making a hypothesis

We claim that the expected running time is at most $cn \log n$ for all $n \geq 1$. We prove this by induction on n. Let a be a constant such that partitioning of a size n subarray requires at most an steps.

For the base case, we can choose a value of c so that the claim hold.

For the induction step, let $n \geq 3$ and suppose that the claim holds for all values of n less than the current one.

The expected running time satisfies the following:

$$\begin{aligned} & T(n) \\ & \leq \quad an + \frac{\sum_{k=0}^{n-1}(T(k) + T(n-1-k))}{n} \\ & = \quad an + \frac{2}{n}\sum_{k=1}^{n-1} T(k). \end{aligned}$$

By our induction hypothesis, this is at most

$$an + \frac{2c}{n}\sum_{k=1}^{n-1} k \log k.$$

Note that

$$\sum_{k=1}^{n-1} k \lg k \leq \int_1^n x \lg x dx.$$

The integration is equal to

$$\frac{1}{2}n^2 \lg n - \frac{n^2}{4} + \frac{1}{4}.$$

This is at most

$$\frac{1}{2}n^2 \lg n - \frac{n^2}{8}.$$

By plugging this bound, we have

$$T(n) \leq cn \lg n + (a - \frac{c}{4})n.$$

Choose c so that $c > 4a$. Then,

$$T(n) \leq cn \lg n.$$

Thus, we have proven:

Theorem C Randomized Quicksort has the expected running time of $\Theta(n \lg n)$.