Quicksort

Partitioning

Divide array A into three sections: a section with values $\leq$ pivot, the pivot, and a section with values $>$ pivot

PARTITION(A, l, r)

$p \leftarrow A[l]$
$i \leftarrow l$
for j from $l+1$ to r **do**
    if $A[j] \leq p$ **then**
        $i \leftarrow i+1$
        swap $A[i]$ with $A[j]$
swap $A[i]$ with $A[l]$
return i // returns the location that the pivot ends up

Quicksort

strategy: partition full array, and then partition left and right resulting partitions

QUICKSORT(A, l, r)

1: **if** $l < r$ **then**
2: $\quad i \leftarrow$ PARTITION(A, l, r)
3: $\quad$ QUICKSORT$(A, l, i-1)$
4: $\quad$ QUICKSORT$(A, i+1, r)$

correctness? runtime complexity?

QUICKSORT Recursion Tree

Draw branches of recursive call

Calculate running time of a single call

Best case?

Partition splits in half, $O(n \lg n)$ run time

Worst case: $O(n^2)$ run time

QUICKSORT Recursion Tree

Draw branches of recursive call

Calculate running time of a single call

Best case?

Partition splits in half, $O(n \lg n)$ run time

Worst case: $O(n^2)$ run time

QUICKSORT Recursion Tree

Draw branches of recursive call

Calculate running time of a single call

Best case?

Partition splits in half, $O(n \lg n)$ run time

Worst case: $O(n^2)$ run time

QUICKSORT Recursion Tree

Draw branches of recursive call

Calculate running time of a single call

Best case?

Partition splits in half, $O(n\lg n)$ run time

Worst case: $O(n^2)$ run time

Correctness

Property to prove: The partition algorithm partitions `A[l..r]`.
Assumptions: $r > l$.
Invariant property: At the beginning of the for-loop, values in the range `A[l+1..i]`$\leq$`p` and values in the range `A[i+1..j-1]>p`.
Initialization: Both the `A[l+1..i]` and `A[i+1..j-1]` ranges are empty, so the two parts of the invariant property are trivially true.
Maintenance: Assume that the property is true up to an index of k: `A[l+1..i]`$\leq$p, `A[i+1..k]>p`. Show that it is true up to $k+1$ after running the for-loop one time.
If `A[k+1]>p`, no values are swapped and i is unchanged. The >p partition has increased in size by 1 and the $\leq$p partition has not changed, so the invariant property remains true. If `A[k+1]`$\leq$p, both i and j are incremented, which increases the $\leq$p partition's size by 1. The current value is swapped into that partition, and a value >p (the value at i) is swapped to the >p partition. Therefore the invariant property stays correct in that case too.
Termination: We looped through the full `A[l+1..r]` range. Therefore `A[l+1..i]`$\leq$p and `A[i+1..r]>p`. p is swapped with `A[i]`, which ends our algorithm with the condition that `A[l..i-1]`$\leq$p, `A[i]=p`, `A[i+1..r]>p`. The range is correctly partitioned.

Correctness

Property to prove: The partition algorithm partitions `A[l..r]`.
Assumptions: $r > l$.
Invariant property: At the beginning of the for-loop, values in the range `A[l+1..i]`≤`p` and values in the range `A[i+1..j-1]>p`.
Initialization: Both the `A[l+1..i]` and `A[i+1..j-1]` ranges are empty, so the two parts of the invariant property are trivially true.
Maintenance: Assume that the property is true up to an index of k: `A[l+1..i]`≤p, `A[i+1..k]>p`. Show that it is true up to $k+1$ after running the for-loop one time.
If `A[k+1]>p`, no values are swapped and i is unchanged. The >p partition has increased in size by 1 and the ≤p partition has not changed, so the invariant property remains true. If `A[k+1]`≤p, both i and j are incremented, which increases the ≤p partition's size by 1. The current value is swapped into that partition, and a value >p (the value at i) is swapped to the >p partition. Therefore the invariant property stays correct in that case too.
Termination: We looped through the full `A[l+1..r]` range. Therefore `A[l+1..i]`≤p and `A[i+1..r]>p`. p is swapped with `A[i]`, which ends our algorithm with the condition that `A[l..i-1]`≤p, `A[i]=p`, `A[i+1..r]>p`. The range is correctly partitioned.

Correctness

Property to prove: The partition algorithm partitions `A[l..r]`.
Assumptions: $r > l$.
Invariant property: At the beginning of the for-loop, values in the range `A[l+1..i]`≤`p` and values in the range `A[i+1..j-1]>p`.
Initialization: Both the `A[l+1..i]` and `A[i+1..j-1]` ranges are empty, so the two parts of the invariant property are trivially true.
Maintenance: Assume that the property is true up to an index of k: `A[l+1..i]`≤`p`, `A[i+1..k]>p`. Show that it is true up to $k+1$ after running the for-loop one time.
If `A[k+1]>p`, no values are swapped and i is unchanged. The `>p` partition has increased in size by 1 and the ≤`p` partition has not changed, so the invariant property remains true. If `A[k+1]`≤`p`, both i and j are incremented, which increases the ≤`p` partition's size by 1. The current value is swapped into that partition, and a value `>p` (the value at i) is swapped to the `>p` partition. Therefore the invariant property stays correct in that case too.
Termination: We looped through the full `A[l+1..r]` range. Therefore `A[l+1..i]`≤`p` and `A[i+1..r]>p`. p is swapped with `A[i]`, which ends our algorithm with the condition that `A[l..i-1]`≤`p`, `A[i]=p`, `A[i+1..r]>p`. The range is correctly partitioned.

Correctness

Property to prove: The partition algorithm partitions `A[l..r]`.
Assumptions: $r > l$.
Invariant property: At the beginning of the for-loop, values in the range `A[l+1..i]`≤`p` and values in the range `A[i+1..j-1]>p`.
Initialization: Both the `A[l+1..i]` and `A[i+1..j-1]` ranges are empty, so the two parts of the invariant property are trivially true.
Maintenance: Assume that the property is true up to an index of k: `A[l+1..i]`≤`p`, `A[i+1..k]>p`. Show that it is true up to $k+1$ after running the for-loop one time.
If `A[k+1]>p`, no values are swapped and i is unchanged. The `>p` partition has increased in size by 1 and the ≤`p` partition has not changed, so the invariant property remains true. If `A[k+1]`≤`p`, both i and j are incremented, which increases the ≤`p` partition's size by 1. The current value is swapped into that partition, and a value `>p` (the value at i) is swapped to the `>p` partition. Therefore the invariant property stays correct in that case too.
Termination: We looped through the full `A[l+1..r]` range. Therefore `A[l+1..i]`≤`p` and `A[i+1..r]>p`. p is swapped with `A[i]`, which ends our algorithm with the condition that `A[l..i-1]`≤`p`, `A[i]=p`, `A[i+1..r]>p`. The range is correctly partitioned.

Randomized Partition

Modify partitioning to choose a random pivot between l and r (inclusive):

PARTITION(A, l, r)

$z \leftarrow rand(l, r)$
swap $A[l]$ with $A[z]$
$p \leftarrow A[l]$
$i \leftarrow l$
for j from $l+1$ to r **do**
    if $A[j] \leq p$ **then**
        $i \leftarrow i+1$
        swap $A[i]$ with $A[j]$
swap $A[i]$ with $A[l]$
return i

Expected Runtime

Unlikely to hit worst case with random pivot choices: $2/n$ chance to end up with empty partition for each choice of pivot

Expected runtime:

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k(T(k) + T(n-1-k) + \Theta(n))\right]$$
$$= O(n \lg n)$$

(More details are in textbook p. 175)

Searching

Common Operation Runtimes

Delete assumes that we have already found the value we want to delete in the data structure.

Structure	Find	Insert	Delete
List(unsorted)			
List(sorted)			
Array(unsorted)			
Array(sorted)			
Heap			
BST (unbalanced)			
BST (balanced)			