Quicksort Practice

Basic Idea

5	3	7	1	2	4	6	8

Let's carefully trace the PARTITION algorithm on the last page of this packet for this array.

lt	
rt	
p	
i	
j	

31	8	2	5	40	1	50	71	85	4	70	60	15	90	45	55

Assuming element comparisons as the basic operation, what is the time complexity of PARTITION on an n-element array?

$$\Theta(\quad)$$

Given what you see in the PARTITION algorithm, does it look like QUICKSORT is a stable sorting algorithm?

What is the role of the *pivot* element in QUICKSORT?

What is the best case for a pivot element?

State a recurrence for the number of element comparisons made by all calls to PARTITION for an instance of QUICKSORT for which every pivot results in best case behavior.

$$C_{best}(n) =$$

By the Master Theorem, $C_{best}(n) \in \Theta(\qquad\qquad)$

What is the worst case for a pivot element?

Determine the number of element comparisons made by all calls to PARTITION for an instance of QUICKSORT for which every pivot results in worst case behavior.

$$C_{worst}(n) =$$

Average case analysis, assuming the pivot for each PARTITION step is equally likely to land in each of the n slots.

$C_{avg}(n) =$

Strategies for improved pivot selection:

Strategies for making QUICKSORT more efficient:

Quicksort space overhead: $\Theta(\qquad)$

ALGORITHM QUICKSORT(A, lt, rt)
//Input: an array $A[0..n-1]$
//Input: lower rt and upper rt bounds of the subarray to sort
//The initial call would be with $lt = 0$ and $rt = n - 1$
if $lt < rt$ **then**
$s \leftarrow Partition(A, lt, rt)$
// s is pivot element location
$QuickSort(A, lt, s-1)$
$QuickSort(A, s+1, rt)$

ALGORITHM PARTITION(A, lt, rt)
//Input: an array $A[0..n-1]$
//Input: lower lt and upper rt bounds of the subarray
// to partition
$p \leftarrow A[lt]$ // select pivot
$i \leftarrow lt$
$j \leftarrow rt + 1$
repeat
repeat
$i \leftarrow i + 1$
until $i = rt$ **or** $A[i] \geq p$
repeat
$j \leftarrow j - 1$
until $j = lt$ **or** $A[j] \leq p$
$swap(A[i], A[j])$
until $i \geq j$
$swap(A[i], A[j])$ // undo last swap
$swap(A[lt], A[j])$ // place pivot
return j