Quicksort: a divide and conquer sorting algorithm

1. Introduction

Q: What does Quicksort do?

A: Quicksort will place elements in an array in order (smallest to largest or largest to smallest).

Quicksort properties:

- does not require extra sorting space (in place sorting)
- uses divide-and-conquer
- almost always presented as recursive
- uses partitioning

Q: What is partitioning?

A: Splitting array elements into (at least) two groups.

Before partitioning: array has no particular order

$A[1]$	$A[2]$	$A[3]$	$\ldots$	$A[n]$

After partitioning (2 groups): elements A[1..p] have a propery and elements A[(p + 1)..n] lack property

$A[1]$	$A[2]$	$\dots$	$A[p]$	$A[p+1]$	$\dots$	$A[n]$

These have the property. These do not have the property.

Example

starting array: A =

12	6	9	18	11	3	4	10

(1) Partitioning according to whether number is odd or not

after partitioning: A =

11	9	3	18	12	6	4	10

elements A[1..3] are odd and elements A[4..8] are even

(2) Partitioning with respect to pivot 10

Pivot: a number a used for partitioning.

After partitioning: numbers at left end of A are at most a and numbers at right end of A are at least a.

after partitioning: A =

4	9	3	6	10	18	11	12

elements A[1..5] are less than or equal to 10 and elements A[6..8] are greater than

□

Partitioning in quicksort is done using a pivot a that is an element of the array.

A[1]	A[2]	. . .	α	. . .	A[n-1]	A[n]

These are $\leq \alpha$.

These are $> \alpha$.

Consider partitioning using final element *A*[n] as pivot:

starting array:

$A[1]$	$A[2]$	$A[3]$	$\ldots$	$\alpha = A[n]$

after partitioning when pivot a is at its correct location:

$A[1]$	$A[2]$	$\ldots$	$A[k-1]$	$A[k] = \alpha$	$A[k+1]$	$\ldots$	$A[n]$

elements $A[1..k]$ are less than or equal to α

k elements

elements $A[(k+1)..n]$ are greater than α:

(n-k) elements

Insight: α is in correct position if we want to sort entire array from smallest to largest.

2. Partitioning

Pseudocode

```
PARTITION(A, L, R)
input number array A, L is index of leftmost element to be
handled, R is index of rightmost element to be handled
/* We partition subarray A[L..R] using A[R] as the pivot, which we
denote as α. Let the final location of pivot α be k, L ≤ k ≤ R.
After execution elements A[L..(k − 1)] are less than or equal to α
and elements A[(k + 1)..R] are greater than α. The procedure
returns location k. */
α = A[R], cut = L − 1
for j = L to R − 1
   if A[j] ≤ α then
      cut = cut + 1, swap elements A[cut] and A[j]
   end
end
k = cut + 1, swap elements A[k] and A[R]
return k
```

cut = the current dividing line between the small elements and the large elements

Example

starting array: A =

10	4	2	6	9	3	8	5

compute PARTITION(A, 1, 8)

step	code line(s)	computation	array A ↑ = j ↑ = cut
1	—	L = 1, R = 8	10 | 4 | 2 | 6 | 9 | 3 | 8 | 5
2	9	α = 5, cut = 0	
3	10, 11	j = 1	10 | 4 | 2 | 6 | 9 | 3 | 8 | 5
3	10, 11, 12	j = 2, cut = 1, A[1] = 4, A[2] = 10	10 | 4 | 2 | 6 | 9 | 3 | 8 | 5
4	10, 11, 12	j = 3, cut = 2, A[2] = 2, A[3] = 10	4 | 10 | 2 | 6 | 9 | 3 | 8 | 5
5	10, 11	j = 4	4 | 2 | 10 | 6 | 9 | 3 | 8 | 5
6	10, 11	j = 5	4 | 2 | 10 | 6 | 9 | 3 | 8 | 5
7	10, 11, 12	j = 6, cut = 3, A[3] = 3, A[6] = 10	4 | 2 | 10 | 6 | 9 | 3 | 8 | 5
8	10, 11	j = 7	4 | 2 | 3 | 6 | 9 | 10 | 8 | 5

9 15 $k = 4$, $A[4] = 5$, $A[8] = 6$

4	2	3	6	9	10	8	5

4	2	3	5	9	10	8	6

$k = 4$

□

Comments on PARTITION:

- choice of pivot element (α) is often randomized
- one goal: keep number of element swaps small
- PARTITION uses one approach for swapping; there are others
- to put pivot into correct location only requires one swap (line 15)
- similar approach can be used for other partitioning problems e.g. finding median

3. Quicksort computation

After partitioning when pivot a is at its correct location:

$A[1]$	$A[2]$	$\ldots$	$A[k-1]$	$A[k] = \alpha$	$A[k+1]$	$\ldots$	$A[n]$

elements $A[1..k]$ are less than or equal to α elements $A[(k+1)..n]$ are greater than α;

Insight: a is in correct position if we want to sort entire array from smallest to largest.

Consequence: we can sort the subarrays to left a and the right of a
Divide and conquer!

But ...

Are *A*[1..k] and *A*[(k+1)..n] (roughly) the same size?

Depends on choice of pivot.

Quicksort: description

1. We start with array $A[1..n]$ of numbers in no particular order. We want to rearrange the array so that the numbers are in order from smallest to largest.
2. We partition $A[1..n]$ with respect to $A[n] = \alpha$, such that after partitioning: (i) α is in its correct position k, (ii) elements $A[1..(k-1)]$ are less than or equal to α, (iii) elements $A[(k+1)..n]$ are greater than ~~or equal to~~ α.
3. We can now apply the same algorithm to subarrays $A[1..(k-1)]$ and $A[(k+1)..n]$. We continue recursing until the size of the subarray is 1 or 0.

Quicksort: pseudocode

```
QUICKSORT(A, L, R)
input number array A, L is index of leftmost element to be
handled, R is index of rightmost element to be handled
/* We sort the subarray A[L..R] from smallest to largest using the
quicksort algorithm. */
if L < R then
   k = PARTITION(A, L, R)
   QUICKSORT(A, L, k-1)
   QUICKSORT(A, k+1, R)
end
```

Example

starting array: A =

10	4	2	6	9	3	8	5

compute QUICKSORT(A, 1, 8)

step	recursion level	code line(s)	computation	array A ↑ = k
1	1	–	L = 1, R = 8	10, 4, 2, 6, 9, 3, 8, 5
2	1	7	k = PARTITION(A,1,8) k = 4	4, 2, 3, 5, 9, 10, 8, 6
3	1	8	QUICKSORT(A,1,3)	
4	2	7	k = PARTITION(A,1,3) k = 2	2, 3, 4, 5, 9, 10, 8, 6
5	2	8	QUICKSORT(A,1,1)	
6	2	9	QUICKSORT(A,3,3)	
7	1	9	QUICKSORT(A,5,8)	
8	2	7	k = PARTITION(A,5,8) k = 5	2, 3, 4, 5, 6, 10, 8, 9
9	2	8	QUICKSORT(A,5,4)	
10	2	9	QUICKSORT(A,6,8)	
11	3	7	k = PARTITION(A,6,8) k = 7	2, 3, 4, 5, 6, 8, 9, 10
⋮				

□

Comments on QUICKSORT:

- amount of pseudocode deceptive since PARTITION does all the work
- interpretation: recursively find final locations of each element
- variation: call to QUICKSORT replaced by call to simpler sorting algorithm (often insertion sort) when array size ’small’