CSE 100: Algorithm Design and Analysis
Chapter 07: Quicksort

Sungjin Im

University of California, Merced

Last Update: 03-01-2023

My brain is open.

– *Paul Erdös*

Quicksort

High-level view

- Introduction of quicksort: *in-place* divide-and-conquer sorting
- Introduction of randomized algorithms
- Revisiting expected (average-case) running time

Quicksort

High-level view

To sort $A[p \cdots r]$.

- Divide: Partition $A[p \cdots r]$ into two subarrays.
  $A[p \cdots q-1] \quad \leq \quad A[q] \quad < \quad A[q+1 \cdots r]$.
- Conquer: Sort $A[p \cdots q-1]$ and $A[q+1 \cdots r]$ by recursive calls to Quicksort.
- Combine: Free!

Quicksort

To sort $A[p \cdots r]$:

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

Example: $A[1...7] = \langle 9, 7, 3, 2, 5, 1, 4 \rangle$

Partitioning in Quicksort

Partition(A, p, r)

Input: $A[p...r]$

Do: Let $x = A[r]$ (pivot value) and (after shuffling)

Return the pivot index, $p \leq q \leq r$ with

$$A[p \cdots q-1] \leq A[q] = x < A[q+1 \cdots r].$$

Example: $A[1...7] = \langle 9, 7, 3, 2, 5, 1, 4 \rangle$

Partitioning in Quicksort

Partition(A, p, r)
Input: $A[p...r]$
Do: Let $x = A[r]$ (pivot value) and (after shuffling)
Return the pivot index, $p \leq q \leq r$ with

$$A[p \cdots q-1] \leq A[q] = x < A[q+1 \cdots r].$$

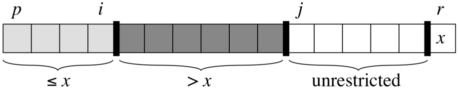
Example: $A[1...7] = \langle 9, 7, 3, 2, 5, 1, 4 \rangle$

We can implement Partition without using more than $O(1)$ auxiliary memory. Thus, Quicksort is an *in-place* sorting algorithm.

Partitioning in Quicksort

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

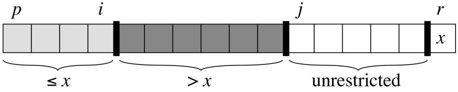
Loop invariant: $A[p \cdots i] \leq A[r] < A[i+1 \cdots j-1]$



Partitioning in Quicksort

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
        if A[j] ≤ x
                i = i + 1
                exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Loop invariant: $A[p \cdots i] \le A[r] < A[i+1 \cdots j-1]$

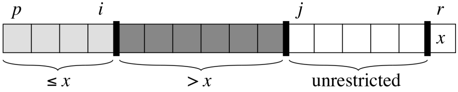


Running time of Partitioning:

Partitioning in Quicksort

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Loop invariant: $A[p \cdots i] \leq A[r] < A[i+1 \cdots j-1]$



Running time of Partitioning: $\Theta(r-p)$, so if the subarray has n elements, then $\Theta(n)$.

Effect of Partitioning on Quicksort's Running Time

Running time of Partitioning: $\Theta(n)$.

Running time of Quicksort:

Effect of Partitioning on Quicksort's Running Time

Running time of Partitioning: $\Theta(n)$.

Running time of Quicksort: depends on the partition...

Effect of Partitioning on Quicksort's Running Time

Running time of Partitioning: $\Theta(n)$.

Running time of Quicksort: depends on the partition... If the pivot is close to a median, the partition will be effective.

- Worst-case partitioning: one subproblem with $n - 1$ elements and one with 0 elements.
- Best-case partitioning: each subproblem with less than $n/2$ elements.
- 'Balanced' partitioning: each subproblem with less than say $9/10$ elements.

Effect of Partitioning on Quicksort's Running Time

(For simplicity, we assume that all elements in the input are distinct. See Problems 7-2 in the textbook to remove this assumption.)

Worst-case partitioning:

($n-1$ smaller elements) Pivot (no larger elements)

or

(no smaller elements) Pivot ($n-1$ larger elements)

Effect of Partitioning on Quicksort's Running Time

(For simplicity, we assume that all elements in the input are distinct. See Problems 7-2 in the textbook to remove this assumption.)

Worst-case partitioning:

$$(n-1 \text{ smaller elements}) \text{ Pivot } (\text{no larger elements})$$

or

$$(\text{no smaller elements}) \text{ Pivot } (n-1 \text{ larger elements})$$

If the worst-case partitioning keeps occurring,

$$T(n) = T(n-1) + \Theta(n)$$

Effect of Partitioning on Quicksort's Running Time

(For simplicity, we assume that all elements in the input are distinct. See Problems 7-2 in the textbook to remove this assumption.)

Worst-case partitioning:

$$(n-1 \text{ smaller elements}) \text{ Pivot } (\text{no larger elements})$$

or

$$(\text{no smaller elements}) \text{ Pivot } (n-1 \text{ larger elements})$$

If the worst-case partitioning keeps occurring,

$$T(n) = T(n-1) + \Theta(n) \to T(n) = \Theta(n^2).$$

In the worst case, RT $= \Theta(n^2)$

Effect of Partitioning on Quicksort's Running Time

Best-case partitioning:

($n/2$ smaller elements) Pivot ($n/2$ larger elements)

Effect of Partitioning on Quicksort's Running Time

Best-case partitioning:

($n/2$ smaller elements) Pivot ($n/2$ larger elements)

If the best-case partitioning keeps occurring,

$$T(n) = 2T(n/2) + \Theta(n)$$

Effect of Partitioning on Quicksort's Running Time

Best-case partitioning:

$$(n/2 \text{ smaller elements}) \text{ Pivot } (n/2 \text{ larger elements})$$

If the best-case partitioning keeps occurring,

$$T(n) = 2T(n/2) + \Theta(n) \to T(n) = \Theta(n \log n).$$

In the best case, RT $= \Theta(n \log n)$

Effect of Partitioning on Quicksort's Running Time

'Balanced'-case partitioning:

$(\frac{1}{10}n$ smaller elements) Pivot $(\frac{9}{10}n$ larger elements)

or

$(\frac{1}{10}n+1$ smaller elements) Pivot $(\frac{9}{10}n-1$ larger elements)

or

$\ldots$

$(\frac{9}{10}n$ smaller elements) Pivot $(\frac{1}{10}n$ larger elements)

Effect of Partitioning on Quicksort's Running Time

'Balanced'-case partitioning:

$$(\tfrac{1}{10}n \text{ smaller elements}) \text{ Pivot } (\tfrac{9}{10}n \text{ larger elements})$$
or
$$(\tfrac{1}{10}n+1 \text{ smaller elements}) \text{ Pivot } (\tfrac{9}{10}n-1 \text{ larger elements})$$
or
$$\ldots$$
$$(\tfrac{9}{10}n \text{ smaller elements}) \text{ Pivot } (\tfrac{1}{10}n \text{ larger elements})$$

If the first case (worst balanced partitioning) keeps occurring,

$$T(n) = T(\tfrac{1}{10}n) + T(\tfrac{9}{10}n) + \Theta(n)$$

Effect of Partitioning on Quicksort's Running Time

'Balanced'-case partitioning:

$$(\tfrac{1}{10}n \text{ smaller elements}) \text{ Pivot } (\tfrac{9}{10}n \text{ larger elements})$$
or
$$(\tfrac{1}{10}n + 1 \text{ smaller elements}) \text{ Pivot } (\tfrac{9}{10}n - 1 \text{ larger elements})$$
or
$$\ldots$$
$$(\tfrac{9}{10}n \text{ smaller elements}) \text{ Pivot } (\tfrac{1}{10}n \text{ larger elements})$$

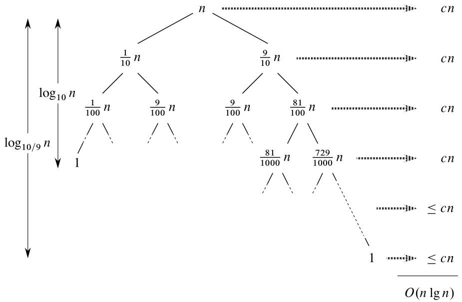
If the first case (worst balanced partitioning) keeps occurring,

$$T(n) = T(\tfrac{1}{10}n) + T(\tfrac{9}{10}n) + \Theta(n) \rightarrow T(n) = O(n \log n).$$

In the 'balanced' case, RT $= \Theta(n \log n)$

Effect of Partitioning on Quicksort's Running Time

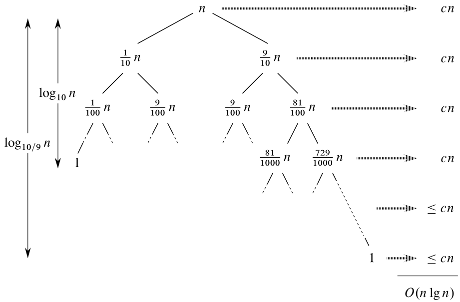
'Balanced'-case partitioning:



$$T(n) = T(\tfrac{1}{10}n) + T(\tfrac{9}{10}n) + \Theta(n)$$

Effect of Partitioning on Quicksort's Running Time

'Balanced'-case partitioning:



$$T(n) = T(\tfrac{1}{10}n) + T(\tfrac{9}{10}n) + \Theta(n) \rightarrow T(n) = O(n \log n).$$

In the 'balanced' case, RT $= \Theta(n \log n)$

Effect of Partitioning on Quicksort's Running Time

If the input is a random permutation of n elements, we will have a 'balanced'-case with probability $\geq \frac{8}{10}$. So in expectation, the tree depth will be $\frac{10}{8} \cdot O(\log_{10/9} n) = O(\log n)$.

Effect of Partitioning on Quicksort's Running Time

If the input is a random permutation of n elements, we will have a 'balanced'-case with probability $\geq \frac{8}{10}$. So in expectation, the tree depth will be $\frac{10}{8} \cdot O(\log_{10/9} n) = O(\log n)$.

A slightly less handwavy argument:
$T(n) \leq 0.8(T(\frac{1}{10}n) + T(\frac{9}{10}n) + \Theta(n)) + 0.2T(n-1)$

- 80% chance: balanced partitioning; worst balanced partitioning is $(1/10)n$ and $(9/10)n$.
- 20% chance: un-balanced partitioning; worst unbalanced partitioning $n-1$ and 0.

$\Rightarrow T(n) \leq T(\frac{1}{10}n) + T(\frac{9}{10}n) + \Theta(n)$
$\Rightarrow T(n) = \Theta(n \log n)$

The average RT $= O(n \log n)$ when the input is a random permutation of n elements. But can we get similar running time for any arbitrary instances? Yes! by using randomized algorithms.

- The average RT $= O(n \log n)$ when the input is a random permutation of n elements (i.e. when all permutations are equally likely).
- But quicksort's worst case running time is still $\Theta(n^2)$.
- We add randomization to quicksort so that the average running time is $\Theta(n \log n)$ for **any** input

Deterministic algorithms: you always get the same output for the same input.
Randomized algorithms: deterministic algorithms with access to "random" coins/functions

Quicksort

Randomized partitioning

Randomly pick an element as the pivot. The random pivot will lead to a 'balanced' partition with probability $\geq \frac{8}{10}$.

RANDOMIZED-PARTITION(A, p, r)

```
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

Quicksort

Randomized partitioning

Randomly pick an element as the pivot. The random pivot will lead to a 'balanced' partition with probability $\geq \frac{8}{10}$.

```
RANDOMIZED-PARTITION(A, p, r)
  i = RANDOM(p, r)
  exchange A[r] with A[i]
  return PARTITION(A, p, r)
```

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

Quicksort

The average/expected/average-case RT of the randomized quicksort is $\Theta(n \log n)$ for *any* input. (The randomness is internal to the algorithm and is independent of the input).

vs.

The deterministic quicksort has an average running time $\Theta(n \log n)$ over all inputs when all permutations. are equally likely. The deterministic quicksort has a (worst-case) running time $\Theta(n^2)$.

Quicksort

- Deterministic Quicksort. Let $T(I)$ denote the algorithm's RT on input I.
  - $\max_{I \text{ of size } n} T(I) = \Theta(n^2)$.
  - $\mathbb{E}_{I \text{ is a random permutation of } n \text{ elements}} T(I) = \Theta(n \log n)$.
- Randomized Quicksort. Let $T(I, r)$ denote the algorithm's RT with random coins r:
  - For any input I of size n,
    $\mathbb{E}_{\text{internal random coins } r} \; T(I, r) = \Theta(n \log n)$.

Overview of Randomized Quicksort Analysis

- Quicksort is a comparison based algorithm: the final ordering is only determined by comparisons between the input elements.
- For simplicity, we can assume wlog that $1, 2, 3, \ldots, n$ are the elements to be sorted and we aim to count the number of comparisons made throughout the execution to asymptotically bound RT.
- Let X_{ij} is the indicator var such that $X_{ij} = 1$ if i is compared to j, otherwise 0.
- We want to bound $\mathbb{E} \sum_{1 \leq i < j \leq n} X_{ij} = \sum_{1 \leq i < j \leq n} \mathbb{E} X_{ij}$ by linearity of expectation.
- Show that $\mathbb{E} X_{ij} = \frac{2}{j-i+1}$.
- Show that $\mathbb{E} \sum_{1 \leq i < j \leq n} \frac{2}{j-i+1} = O(n \log n)$.

Therefore, Randomized Quicksort makes $O(n \log n)$ comparisons in expectation, meaning that its expected running time is $O(n \log n)$.