CS 381

Review: Quicksort

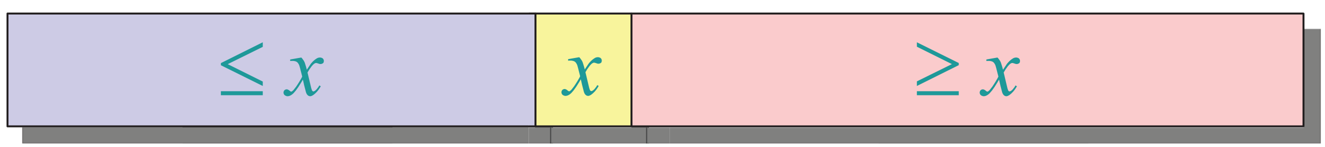
- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts "in place" (like insertion sort, but not like merge sort).
- Very practical (with tuning).

Quicksort an *n*-element array

1. Pick an element, called a *pivot* – *say x* - from the array.

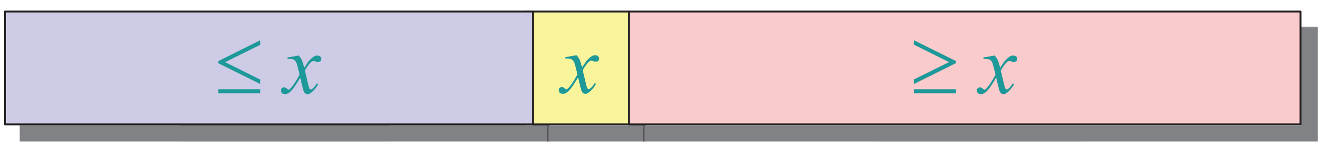
Quicksort an n-element array

1. Pick an element, called a *pivot* – *say x* - from the array.
2. *Partitioning*: reorder the array so that all elements with values less than the pivot are to its and the rest to the right partitioning, the pivot is in its final position. This is called the *partition* operation.



Quicksort an *n*-element array

1. Pick an element, called a *pivot* – *say x* - from the array.
2. *Partitioning*: reorder the array so that all elements with values less than the pivot are to its and the rest to the right partitioning, the pivot is in its final position. This is called the *partition* operation.



3. Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

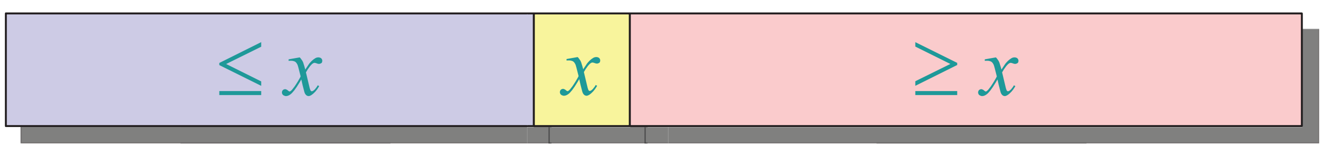
Quicksort an n-element array

Popular choice of pivot: x is a random element of the array

Exercise: write down recurrence for this choice of pivot and prove the expected run time is O(n log n).

- Assume all input elements are distinct.

(following analysis by Daniel Gildea).



The Algorithm

Quicksort(A, n)

Quicksort$'(A, 1, n)$

Quicksort$'(A, p, r)$

if $p \geq r$ **then return**

$q = \text{Partition}(A, p, r)$

Quicksort$'(A, p, q - 1)$

Quicksort$'(A, q + 1, r)$

Partition(A, p, r)

$x = A[r]$
$i \leftarrow p - 1$
for $j \leftarrow p$ **to** $r - 1$ **do**
    if $A[j] \leq x$ **then** {
        $i \leftarrow i + 1$
        Exchange $A[i]$ and $A[j]$ }
Exchange $A[i + 1]$ and $A[r]$
return $i + 1$

Worst Case Runtime

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 $O(n^2)$.

Best Case Runtime

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 $O(n \log n)$.

```
Recurrence:

T(n) = 2 T(n/2) + n
T(0) = T(1) = 0 (best case)
```

Randomized-Quicksort

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

```
Quicksort(A,n)
    Quicksort'(A,1,n)

Quicksort'(A,p,r)
    Pick t uniformly at random from {p, p + 1, ..., r}
    Exchange A[r] and A[t]
    if p ≥ r then return
    q = Partition(A, p, r)
    Quicksort'(A,p,q-1)
    Quicksort'(A,q+1,r)
```

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.

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. **Note:** Convenient to work with log in base e (equivalent by changing the constant c).

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:

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

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

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

Note that $\int x \log(x) = \frac{1}{2}x^2 \log(x) - \frac{x^2}{4} + D$ for any constant D

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 $\Theta(n \lg n)$.

Alternative proof without using integrals?

Exercise:

Bound $\sum_{k=1,n-1} k \cdot \log(k)$ by $\frac{1}{2}\mathrm{n}^2\log(\mathrm{n}) - \mathrm{c}_1 \cdot n^2$