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)

1: **Quicksort**$'(A, 1, n)$

Quicksort$'(A, p, r)$

1: **if** $p \geq r$ **then return**
2: $q =$ **Partition**(A, p, r)
3: **Quicksort**$'(A, p, q - 1)$
4: **Quicksort**$'(A, q + 1, r)$

```
Partition(A, p, r)
 1: x = A[r]
 2: i ← p − 1
 3: for j ← p to r − 1 do
 4:     if A[j] ≤ x then {
 5:         i ← i + 1
 6:         Exchange A[i] and A[j] }
 7: Exchange A[i + 1] and A[r]
 8: 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)
1: **Quicksort'**(A, 1, n)

Quicksort'(A, p, r)
-1: Pick t uniformly at random from $\{p, p+1, \ldots, r\}$
0: Exchange A[r] and A[t]
1: **if** $p \geq r$ **then return**
2: q = **Partition**(A, p, r)
3: **Quicksort'**(A, p, q-1)
4: **Quicksort'**(A, q+1, r)