# Quicksort

CS 4231, Fall 2017

Mihalis Yannakakis

# Quicksort

- Based on divide and conquer
- practical, fast,
- sorts in place

# Quicksort

- Divide: Partition the input array A of elements with respect to a pivot element x into two parts:

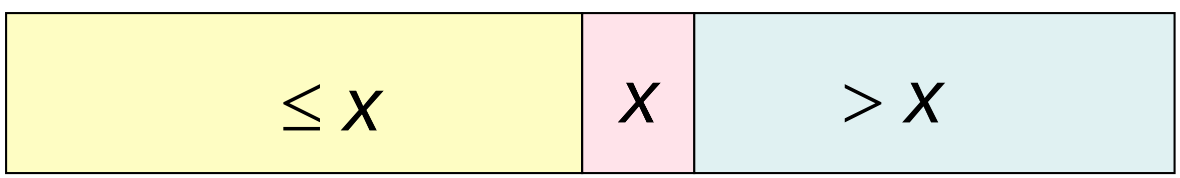
| | | |
|---|---|---|
| $\leq x$ | $x$ | $> x$ |

- Conquer: Sort recursively the two parts using Quicksort
- Combine: Trivial

# Partitioning the array in place

PARTITION(A,p,r)

- Input: Array A[1..n], indices p,r
- Effect: Modifies subarray A[p...r] in place so that subarray partitioned with respect to pivot element x=A[r]




- Output: index of new position of pivot

# PARTITION(A,p,r)

x = A[r]

i = p-1

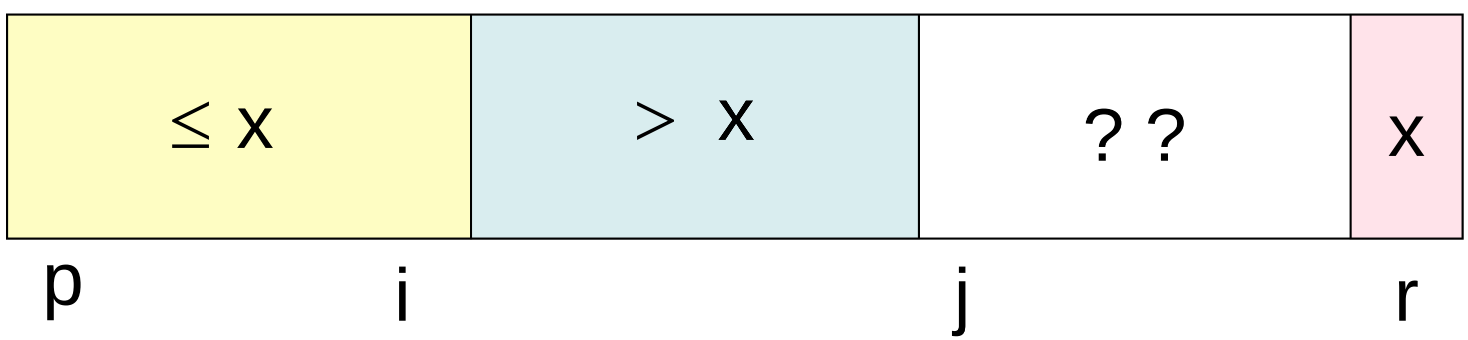
**for** j=p **to** r-1

    **if** A[j] ≤ x **then** { i=i+1, exchange A[i] and A[j] }

exchange A[i+1] and A[r]

return i+1

## Invariant



# Partitioning example

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 8 | 4 | 2 | 9 | 7 | 6 |

$i$ $j$

# Partitioning example

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 8 | 4 | 2 | 9 | 7 | 6 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 8 | 4 | 2 | 9 | 7 | 6 |
| $i$ | $j$ | | | | | |

# Partitioning example

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 8 | 4 | 2 | 9 | 7 | 6 |
| 3 | 8 | 4 | 2 | 9 | 7 | 6 |
| 3 | 8 | 4 | 2 | 9 | 7 | 6 |
| 3 | 4 | 8 | 2 | 9 | 7 | 6 |
| 3 | 4 | 2 | 8 | 9 | 7 | 6 |
| 3 | 4 | 2 | 8 | 9 | 7 | 6 |
| 3 | 4 | 2 | 8 | 9 | 7 | 6 |
| 3 | 4 | 2 | 6 | 9 | 7 | 8 |

# Quicksort

- QUICKSORT(A,p,r)

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

Main Call: QUICKSORT(A,1,n)

# Analysis of Quicksort: Partition routine

- PARTITION (A,p,r)     partitions subarray A[p..r]

  x = A[r]

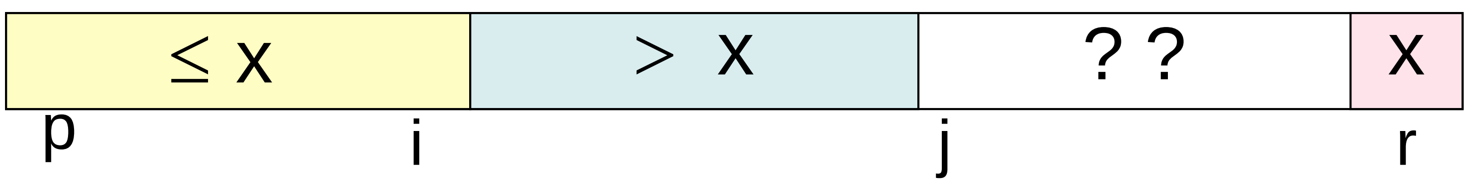
  i = p-1

  **for** j=p **to** r-1

      **if** A[j] $\leq$ x **then** { i=i+1, exchange A[i] and A[j] }

  exchange A[i+1] and A[r]

  return i+1




Complexity: $\Theta$(|subarray|)

# Analysis of Quicksort

- $T(n) = T(|\text{left part}|) + T(|\text{right part}|) + \Theta(n)$

- Worst Case: When we partition each subarray,

last element is largest or smallest $\Rightarrow$ unbalanced partition

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

Solution: $T(n) = \Theta(n^2)$

- Lucky Case: ½: ½ partition $\Rightarrow$ $T(n) = \Theta(n\log n)$
- "Typical" Case: In-between, e.g. ¼:¾ $\Rightarrow$ $T(n) = \Theta(n\log n)$
- For distinct items and random input permutation the average time complexity is $\Theta(n\log n)$

# Randomized Quicksort

- Partition around a randomly chosen pivot element

RANDOMIZED-PARTITION(A,p,r)

i = random index in {p,…,r}

exchange A[i] and A[r]

PARTITION(A,p,r)

# RANDOMIZED-QUICKSORT(A,p,r)

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

Main Call: RANDOMIZED-QUICKSORT(A,1,n)

# Randomized algorithm

- Makes random choices (coin flips, random numbers ..)
- Different random choices are assumed independent
- Outcome of algorithm and running time depends on random choices (besides input)

- Correctness: Show termination and correct answer for all random choices

# Time complexity of randomized algorithms

- Running time: Depends on input *I* and random choices *w* : time *t(I,w)*
- Expected running time for an input I:

  expected time w.r.t. random choices *w:* $\bar{t}(I) = E_w\, t(I,w)$
- Expected time complexity of the algorithm

  Two versions:

  – Worst-case expected time $T(n) = \max_{|I|=n} E_w\, t(I,w)$

  (worst-case expected time over inputs of size n)

  – Average-case expected time $\overline{T}(n) = E_{|I|=n} E_w\, t(I,w)$

  assumes a probability distribution on inputs of size *n,* expected time also w.r.t. inputs

# Time Complexity of Randomized-Quicksort

- Assume all input elements distinct

- Worst-case expected time $T(n)=\Theta(n\log n)$

Thus, for every input *I:* $\bar{t}(I) = O(n\log n)$

# Analysis of Randomized-Quicksort

- Consider an input $I = A[1\ldots n]$
- Suffices to bound # comparisons $X$

$t(I,w) = O(n+X)$

*Proof:*

At most $n$ calls to PARTITION routine

Work of each call = O( size of subarray ) =

= O( # comparisons in the call)

# Analysis of Randomized-Quicksort ctd.

Elements in sorted order : $z_1, z_2, \ldots, z_n$

indicator random variable $X_{ij} = \begin{cases} 1 & \text{if } z_i \text{ is compared to } z_j \\ 0 & \text{otherwise} \end{cases}$

$$X = \sum_{i<j} X_{ij}$$

$$E[X] = \sum_{i<j} E[X_{ij}] = \sum_{i<j} \Pr[z_i \text{ is compared to } z_j]$$

# When is $z_i$ compared to $z_j$ ?

- If and only if $z_i$ *or* $z_j$ is the first element chosen as pivot among $z_i, z_{i+1}, \ldots, z_j$
  - The set stays together till the first pivot in this set
  - If we choose first another pivot $z_k$ between $z_i, z_j$ then $z_i, z_j$ will be split in different parts
  - The first pivot element in the set is compared with all of the other elements
- All elements of set equally likely to be first pivot $\Rightarrow$

  $\Pr[z_i$ is compared to $z_j] = 2/(j-i+1)$

# Analysis of Randomized-Quicksort ctd.

$$E[X] = \sum_{i<j} \Pr[z_i \text{ is compared to } z_j] = \sum_{i<j} \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k}$$

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

recall harmonic series

$$1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = O(\log n)$$

$$= \sum_{i=1}^{n-1} O(\log n)$$

$$= O(n \log n)$$

$$\ln(n+1) < H_n < 1 + \ln n$$

# Expected time complexity of Randomized-Quicksort

- For every input *I* , $\bar{t}(I) = O(E(X)) = O(n \log n)$

  $$\Rightarrow \quad T(n) = O(n \log n)$$

- Conversely, for every input *I*, $\bar{t}(I) = \Omega(n \log n)$

  in fact $\forall$ input *I*, $\forall$ random choice *w*, $t(I,w) = \Omega(n \log n)$

  $$\Rightarrow \quad T(n) = \Omega(n \log n)$$

$$T(n) = \Theta(n \log n)$$

# Quicksort - other points

- Duplicate elements

3-way Partition

| | | |
|---|---|---|
| $< x$ | $= x$ | $> x$ |

Recurse on left and right part

- Choice of pivot
- Termination of recursion