Lecture 8

Quicksort

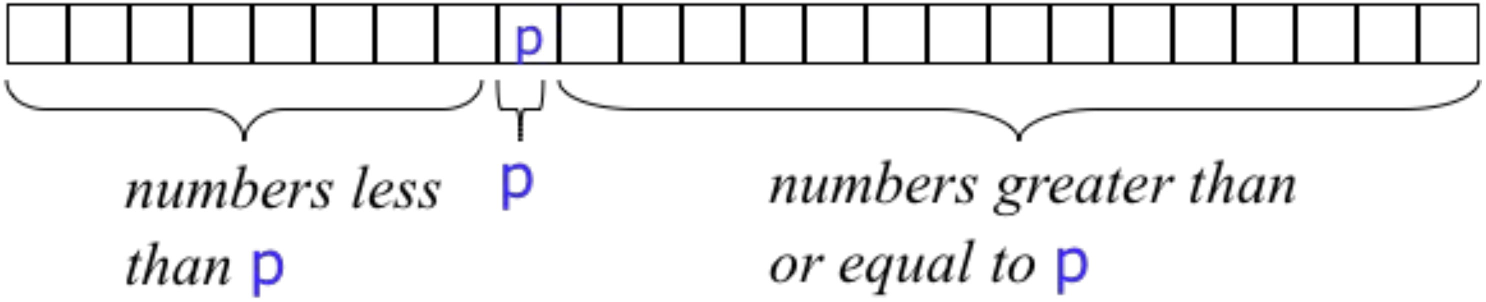
Divide: A[p…r] is partitioned into two nonempty subarrays A[p…q-1] and A[q+1…r] s.t. each element of A[p…q-1] is less than or equal to each element of A[q+1…r]. Index q is computed here, called pivot.

Conquer: two subarrays are sorted by recursive calls to quicksort.

Combine: unlike merge sort, no work needed since the subarrays are sorted in place already.

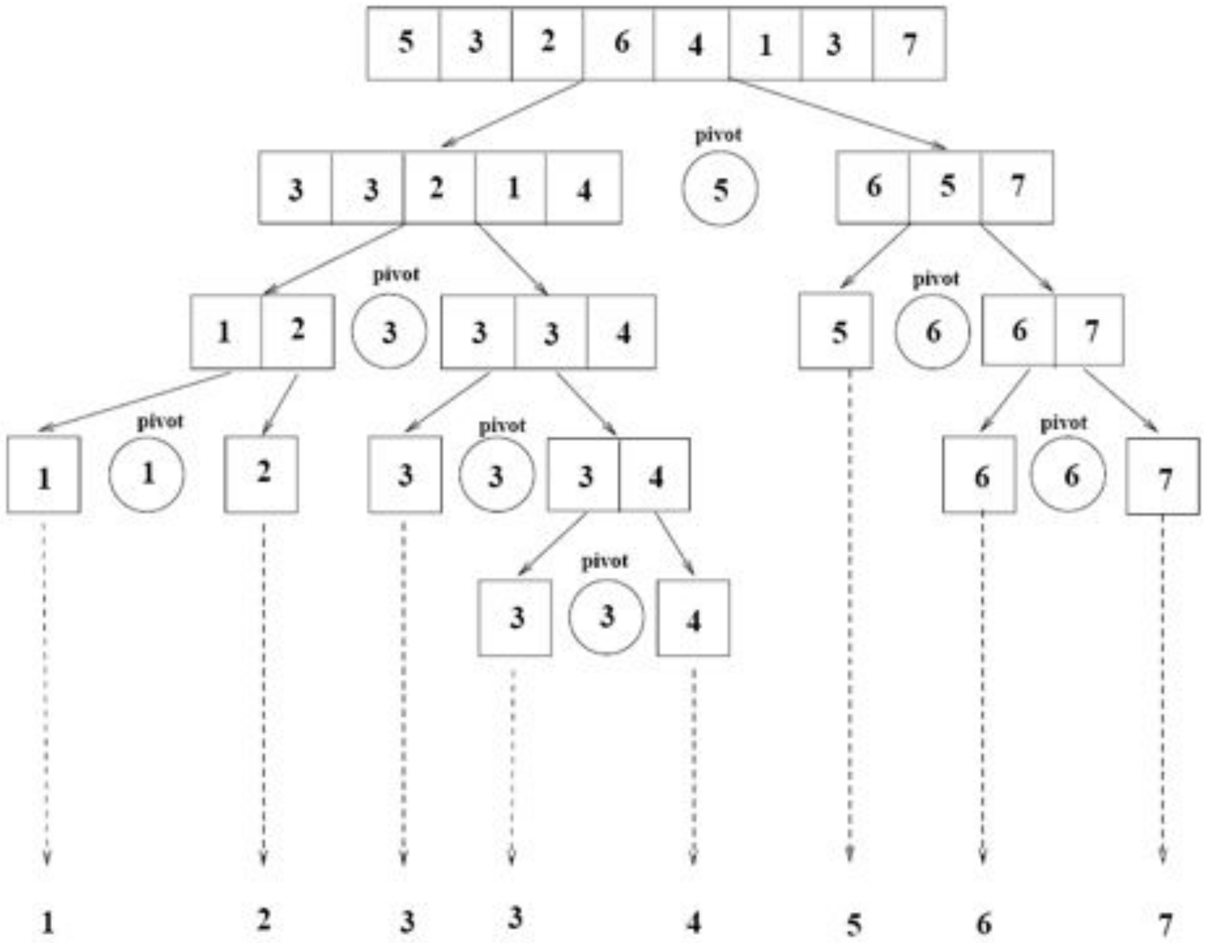
Basic Idea:

- Pick some number p from the array
- Move all numbers less than p to the beginning of the array
- Move all numbers greater than (or equal to) p to the end of the array
- Quicksort the numbers less than p
- Quicksort the numbers greater than or equal to p



Sort using the Quick Sort:

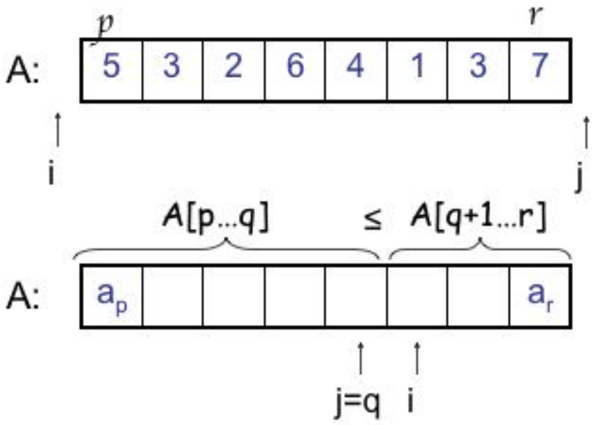
5	3	2	6	4	1	3	7


5 3 2 6 4 1 3 7
pivot
3 3 2 1 4
5
6 5 7
pivot
1 2
3
3 3 4
pivot
5
6
6 7
pivot
1
1
2
pivot
3
3
3 4
pivot
6
6
7
pivot
3
3
4
1
2
3
3
4
5
6
7

Partitioning the Array

Alg. PARTITION (A, p, r)

```
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
        do repeat j ← j - 1
              until A[j] ≤ x
        do  repeat i ← i + 1
              until A[i] ≥ x
        if i < j
              then exchange A[i] ↔ A[j]
         else return j
```



Each element is visited once!

Running time: $\Theta(n)$

$n = r - p + 1$

x

Recurrence

Alg.: QUICKSORT(A, p, r)

Initially: p=1, r=n

if p < r

then $q \leftarrow$ PARTITION(A, p, r)

QUICKSORT (A, p, q)

QUICKSORT (A, q+1, r)

Recurrence:

$$T(n) = T(q) + T(n - q) + n$$

Analysing Quicksort: The Worst Case

The choice of a pivot is most critical:

- The wrong choice may lead to the worst-case quadratic time complexity.
- A good choice equalises both sublists in size and leads to linearithmic (“n log n”) time complexity.

The worst-case choice: the pivot happens to be the largest (or smallest) item.

- Then one subarray is always empty.
- The second subarray contains $n - 1$ elements, i.e. all the elements other than the pivot.
- Quicksort is recursively called only on this second group.

However, quicksort is fast on the “randomly scattered” pivots.

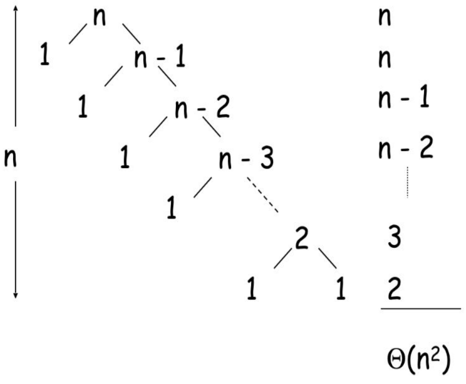
Worst Case Partitioning

- Worst-case partitioning
  - One region has one element and the other has n – 1 elements
  - Maximally unbalanced
- Recurrence: q=1

T(n) = T(1) + T(n – 1) + n,

T(1) = Θ(1)

T(n) = T(n – 1) + n



$$n + \left(\sum_{k=1}^{n} k \right) - 1 = \Theta(n) + \Theta(n^2) = \Theta(n^2)$$

When does the worst case happen?

Best Case analysis

For any pivot position i; $i \in \{0, \ldots, n-1\}$:

- Time for partitioning an array : cn
- The head and tail subarrays contain i and $n-1-i$ items, respectively: $T(n) = cn + T(i) + T(n-1-i)$

Average running time for sorting (a more complex recurrence):

$$T(n) = \frac{1}{n}\sum_{i=0}^{n-1}\left(T(i) + T(n-1-i) + cn\right)$$

$$= \frac{2}{n}\left(T(0) + T(1) + \ldots + T(n-2) + T(n-1)\right) + cn, \text{ or}$$

$$nT(n) = 2\left(T(0) + T(1) + \ldots + T(n-2) + T(n-1)\right) + cn^2$$

$$(n-1)T(n-1) = 2\left(T(0) + T(1) + \ldots + T(n-2)\right) + c(n-1)^2$$

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + 2cn - c \approx 2T(n-1) + 2cn$$

Thus, $nT(n) \approx (n+1)T(n-1) + 2cn$, or $\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}$

Best Case Partitioning

- Best-case partitioning
  - Partitioning produces two regions of size n/2
- Recurrence: q=n/2

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

  $T(n) = \Theta(n\lg n)$ (Master theorem)