Sorting: Quick Sort

Quick Sort

Divide: Partition the array into two sub-arrays

A[p . . q-1] and A[q+1 . . r] such that each element of

A[p . . q-1] is less than or equal to A[q], which in turn

less than or equal to each element of A[q+1 . . r]

Quick Sort

Conquer: Sort the two sub-arrays A[p . . q-1] and

A[q+1 . . r] by recursive calls to quick sort.

Quick Sort

Combine: Since the sub-arrays are sorted in place, no work is needed to combine them.

Quick Sort

```
QUICKSORT(A, p, r)

if p< r

   then q ← PARTITION(A, p, r)

            QUICKSORT(A, p, q-1)

            QUICKSORT(A, q+1, r)
```

Quick Sort

PARTITION(A, p, r)

x ← A[r]

i ← p-1

Quick Sort

for j □ p to r-1

do if A[j] <= x

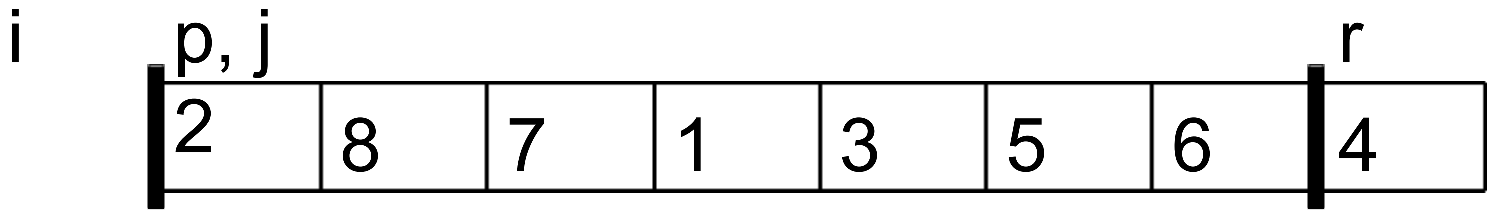
then i □i+1

exchange A[i] □ □ A[j]

exchange A[i+1] □ □ A[r]

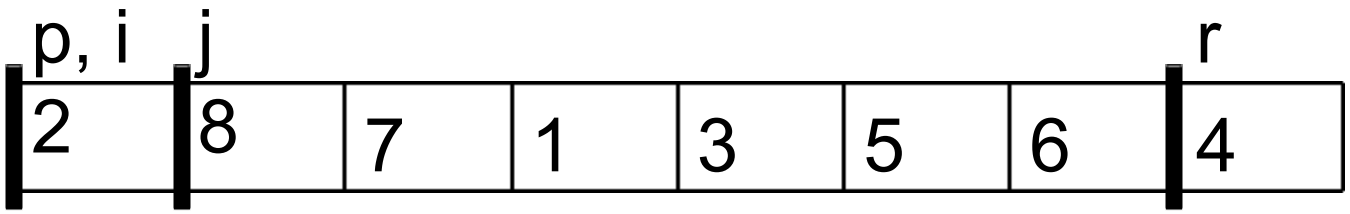
return i+1

Quick Sort



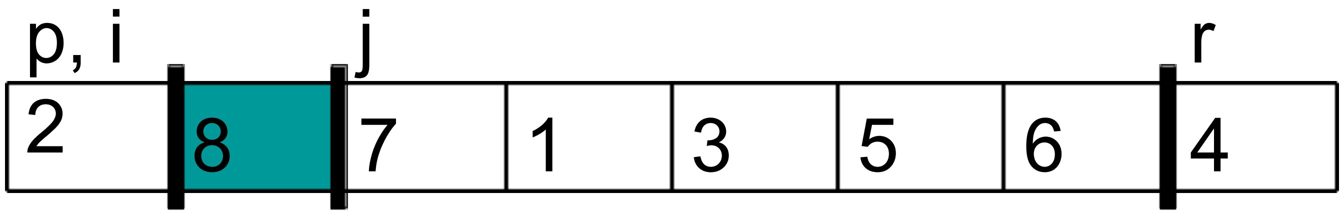
(a)

Quick Sort



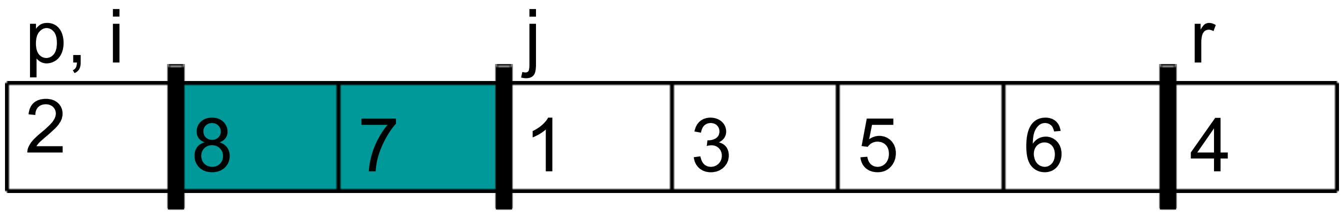
(b)

Quick Sort



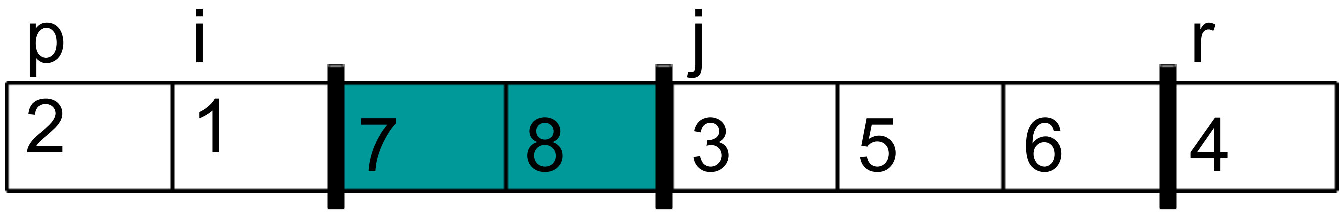
(c)

Quick Sort



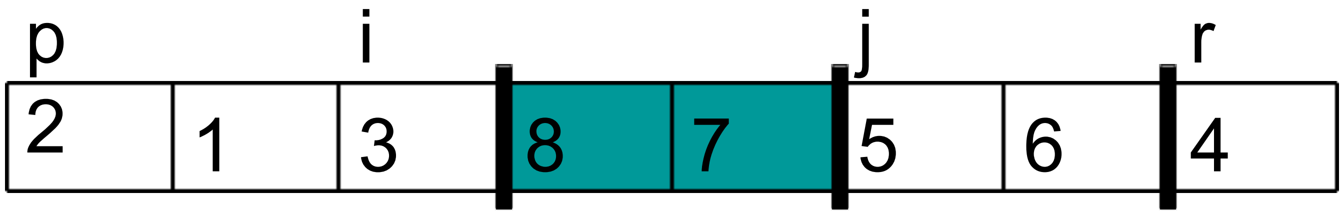
(d)

Quick Sort



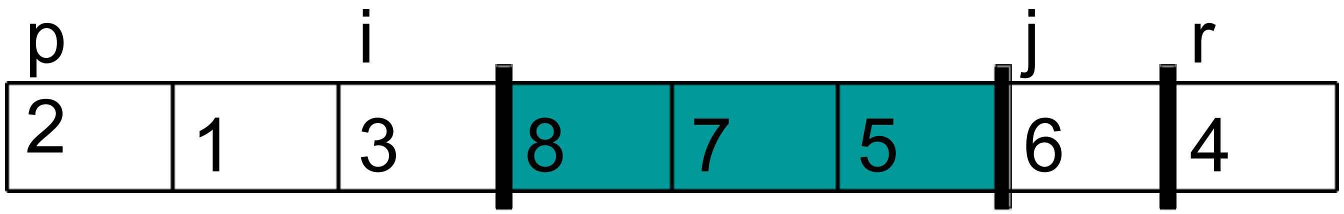
(e)

Quick Sort



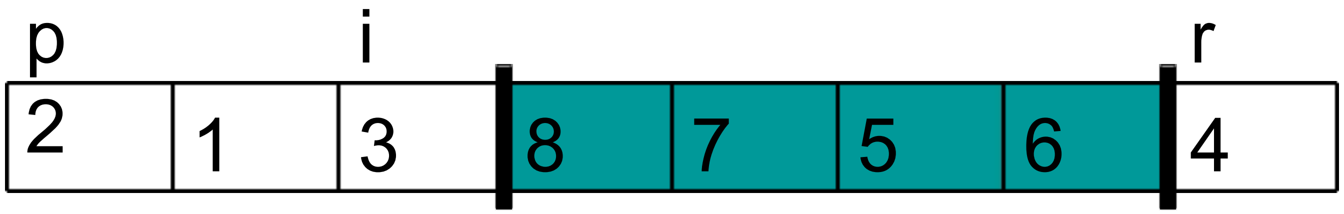
(f)

Quick Sort



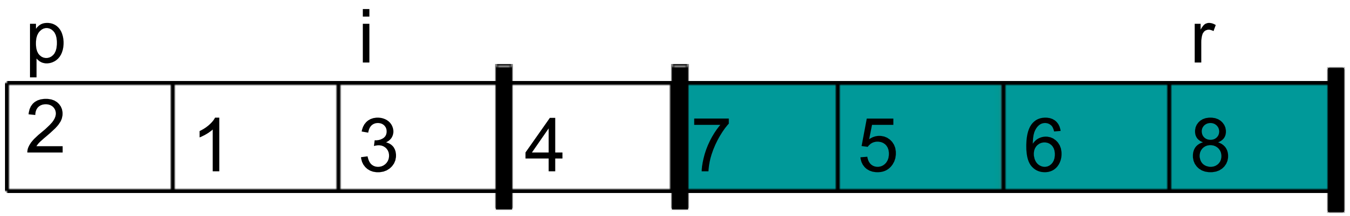
(g)

Quick Sort



(h)

Quick Sort



(i)

Quick Sort

Worst-case partitioning:

The partitioning routine produces one sub-problem with n-1 elements and another sub-problem with 0 elements. So the partitioning costs $\theta(n)$ time.

Quick Sort

Worst-case partitioning:

The recurrence for the running time

$$T(n) = T(n-1) + T(0) + \theta(n)$$

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

$$= \text{-----------------} \ \theta(n^2)$$

Quick Sort

Worst-case partitioning:

The $\theta(n^2)$ running time occurs when the input array is already completely sorted – a common situation in which insertion sort runs in $O(n)$ time

Quick Sort

Best-case partitioning:

The partitioning procedure produces two

sub-problems, each of size not more than n/

2.

Quick Sort

Best-case partitioning:

The recurrence for the running time

$T(n) <= 2T(n/2) + \theta(n)$

$= \text{-----}\ O(n \lg n)$

Quick Sort

Best-case partitioning:

The equal balancing of the two sides of the partition at every level of the recursion produces faster algorithm.

Quick Sort

Balanced partitioning:

Suppose, the partitioning algorithm always produces 9-to-1 proportional split, which seems quite unbalanced.

Quick Sort

Balanced partitioning:

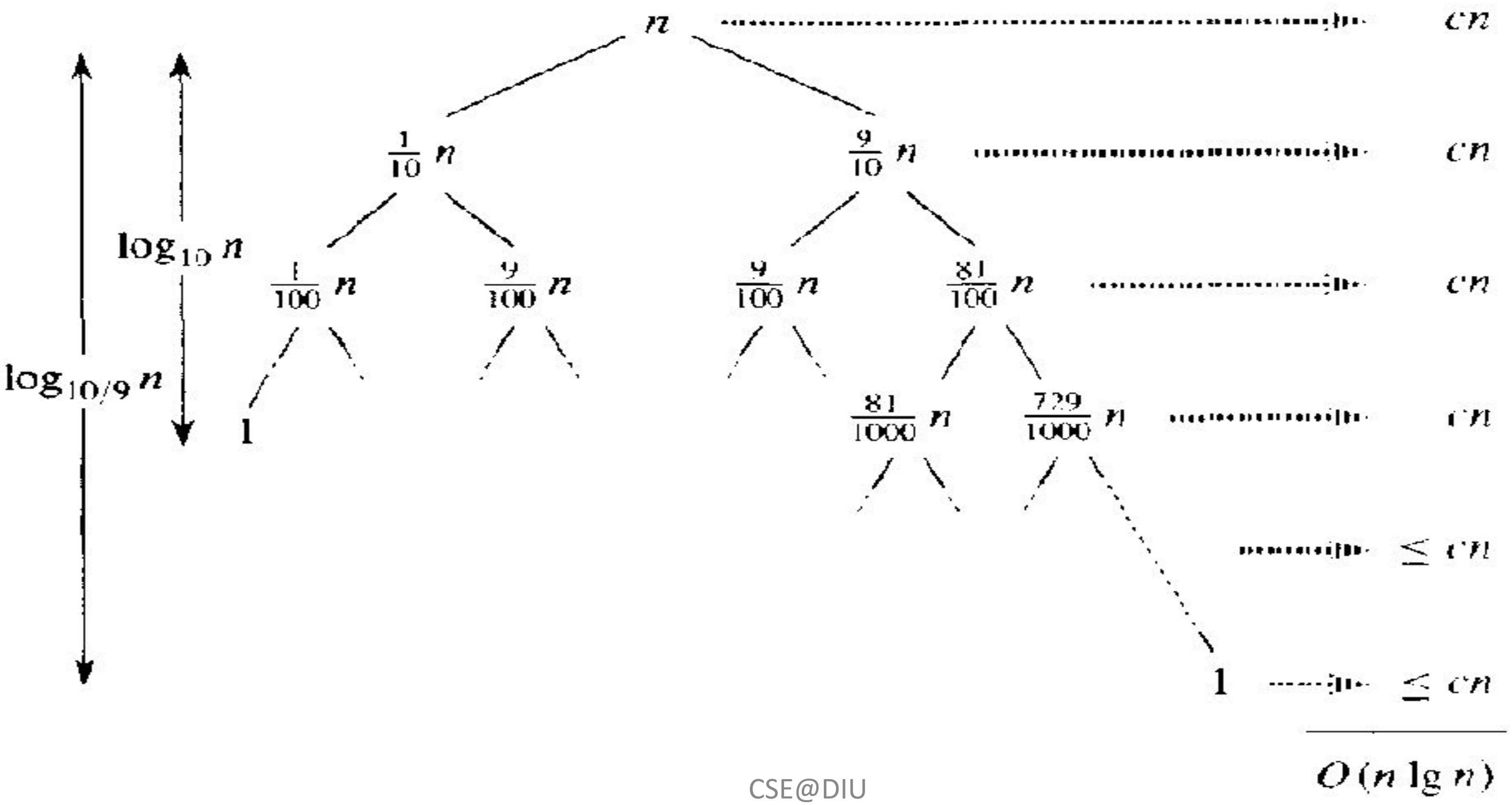
The recurrence for the running time

$T(n) <= T(9n/10) + T(n/10) + cn$

$= \text{------------} O(n \lg n)$

Quick Sort

Balanced partitioning: The recursion tree



Quick Sort

Balanced partitioning:

In fact, a 99-to-1 split yields an $O(n \lg n)$ running time. Any split of constant proportionality yields a recursion tree of depth $\theta(\lg n)$

Quick Sort

Intuition for the average case:

It is unlikely that the partitioning always happens

in the same way at every level.

Quick Sort

Intuition for the average case:

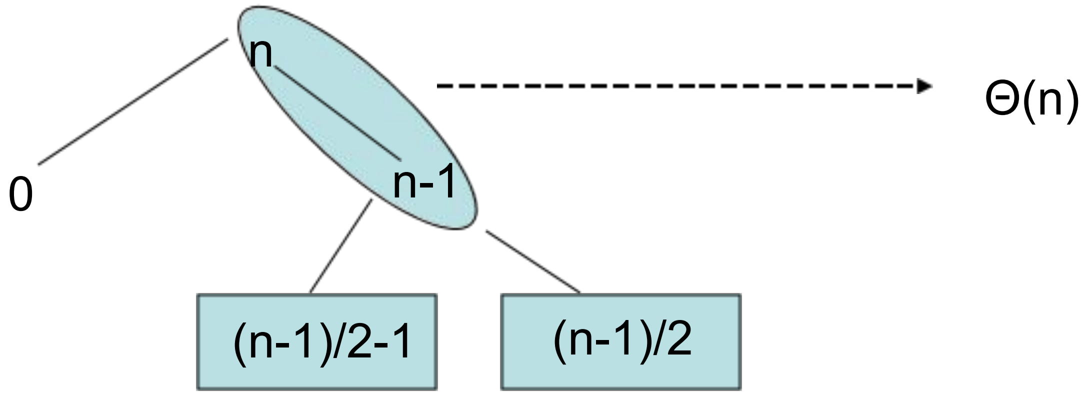
In the average case, PARTION produces a mix of

“good” and “bad” splits.

Quick Sort

Intuition for the average case:

The combination of the bad split followed by the good split produces three arrays of sizes 0, (n-1)/2-1, and (n-1)/2 at a combined partitioning cost of $\theta(n) + \theta(n-1) = \theta(n)$

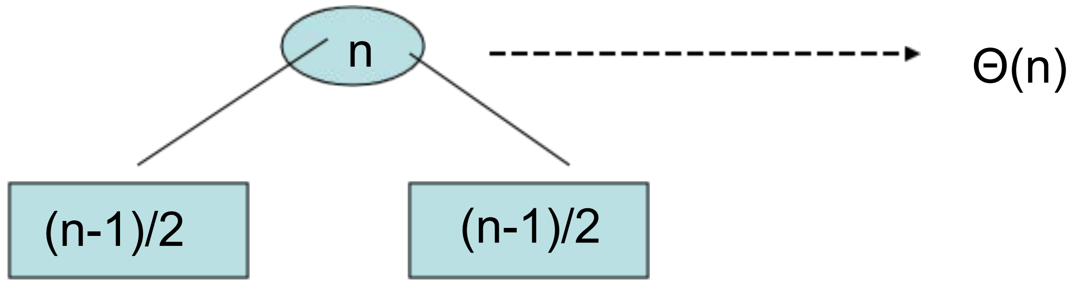


Quick Sort

Intuition for the average case:

A single level of partitioning produces two sub-arrays of size

(n-1)/2 at a cost of $\theta(n)$.



A Randomized Version of Quick Sort

Instead of always using A[r] as the pivot, we will use a randomly chosen element from the sub-array A[p..r].

A Randomized Version of Quick Sort

Because the pivot element is randomly chosen,

we expect the split of the input array to be

reasonably well balanced on average.

A Randomized Version of Quick Sort

RANDOMIZED-PARTITION(A, p, r)

i □ RANDOM(p, r)

exchange A[r] □ □ A[i]

return PARTITION(A, p, r)

A Randomized Version of Quick Sort

```
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)
```

Textbooks & Web References

- Text Book (Chapter 3)
- www.visualgo.net

Thank you