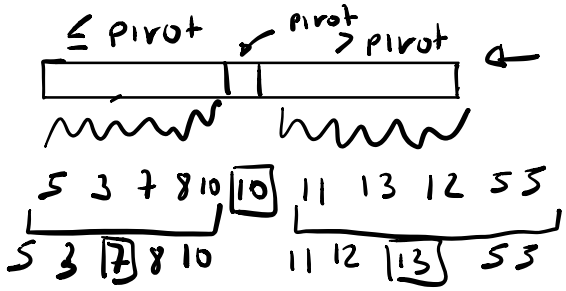
QuickSort

Cormen 7.1, 7.2, 7.3

Another sorting algorithm based on D&C approach

MergeSort
- Divide: Constant time
- Combine: Linear time

QuickSort
- Divide: Linear time
- Combine: Constant time



Divide Partition the array $A[p \ldots r]$ in two (possibly empty) subarrays $A[p \ldots q-1]$ and $A[q+1, \ldots r]$ such that

- each element in $A[p \ldots q-1]$ is $\leq A[q]$
- each element in $A[q+1 \ldots, r]$ is $> A[q]$

Conquer Sort $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ recursively

Combine do nothing

How to choose the pivot ?

it's the elment in position $A[r]$

A before: 1 5 3 2 [3] a

(p = 1, r = 3)

A after: 1 2 [3] 5

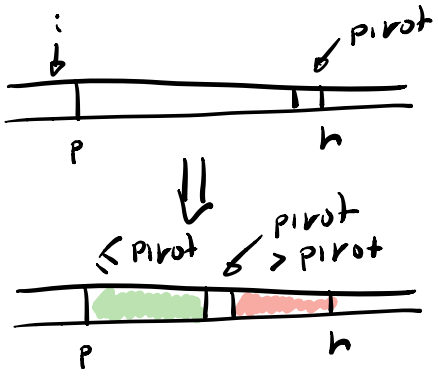
(p = 1, q = 3, r = 4)

QuickSort (A, p, r)

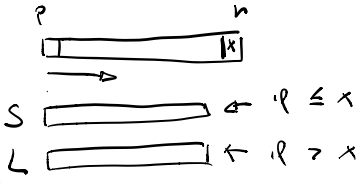
```
if p < r
    q = Partition(A, p, r)
    QuickSort(A, p, q-1)
    QuickSort(A, q+1, r)
```

Partitioning the array

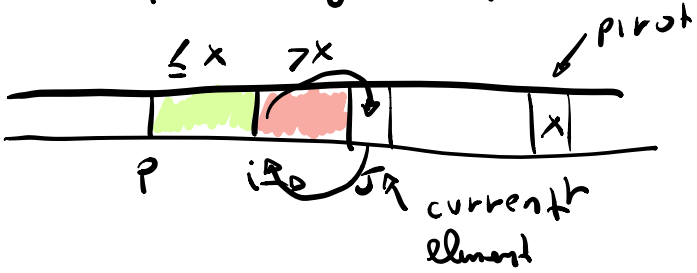
```
Partition (A, p, r)
    x = A[r]
    i = p - 1
    for j = p to r-1
        if A[j] ≤ x
            i = i+1
            swap (A[i], A[j])
    swap (A[i+1], A[r])
    return i+1
```



if not inplace, easy



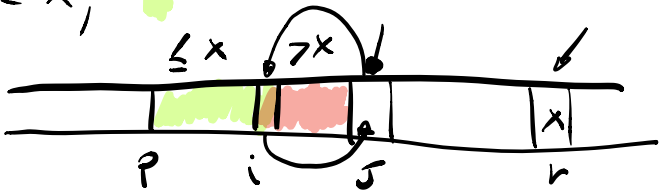
Partion keeps two indexes i and j and following loop invariant

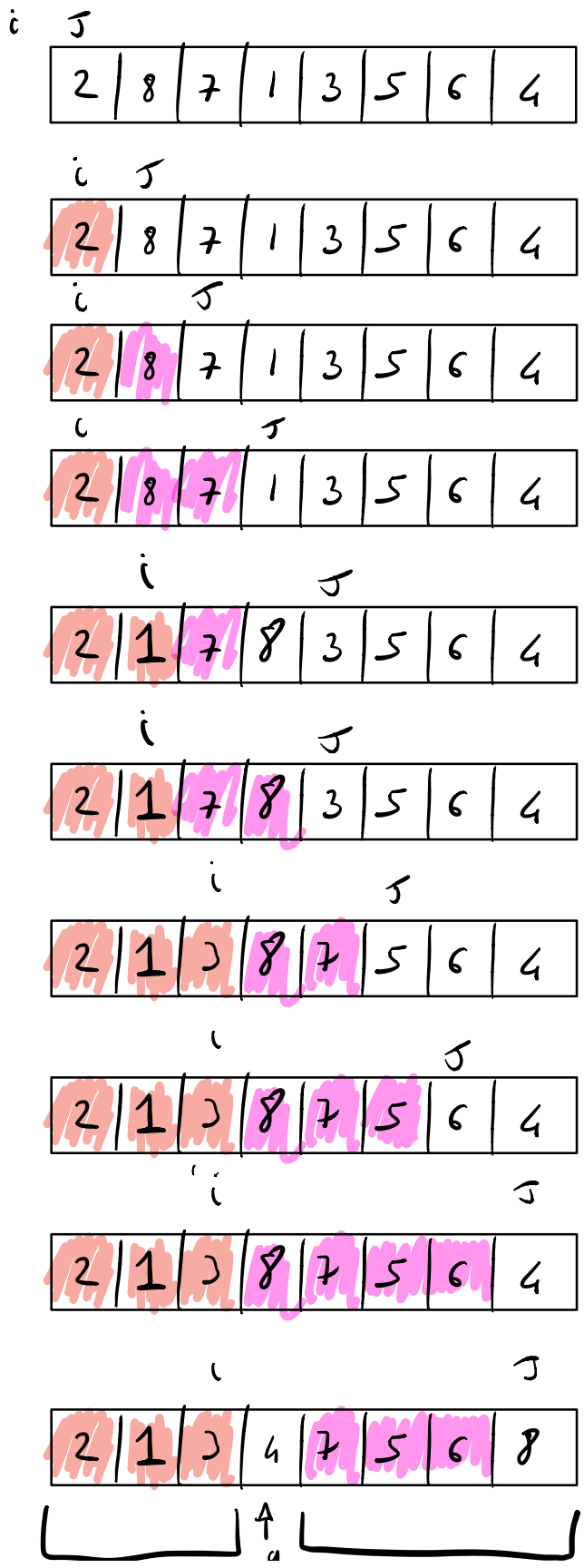


j-th iteration

if A[j] > x, do nothing

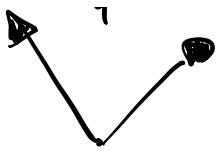
if A[j] ≤ x,






```
Partition (A, p, r)
1  x = A[r]
2  i = p-1
3  for j = p to r-1
4      if A[j] ≤ x
5          i = i+1
6          swap(A[i], A[j])
7  swap(A[i+1], A[r])
8  return i+1
```

stable? NO!!



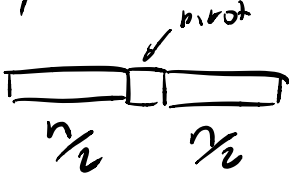
they are in their
correct relative order

Time complexity of Partition $\Theta(n)$

- Stable? No
- Inplace? Yes

Analysis of Quick Sort

Best Case: Every time Pivot is such that

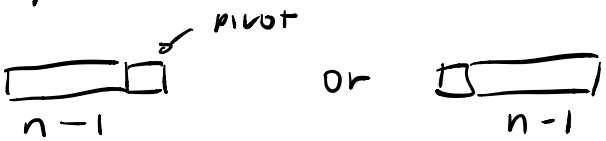


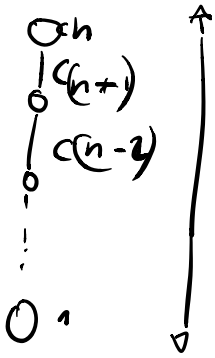
Partitions are balanced

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \quad \text{(as MS)}$$

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

Worst Case: Every time pivot is such that





partitions as imbalanced as possible

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

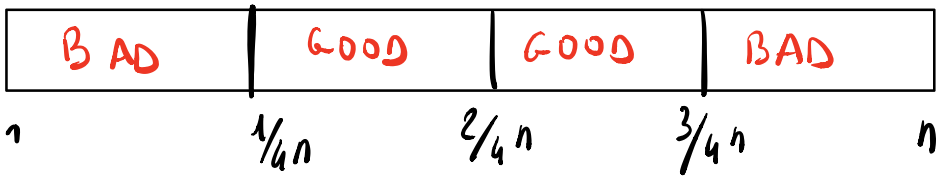
$$\Rightarrow T(n) = \Theta(n^2)$$

Balanced (or almost balanced) subproblem

Good

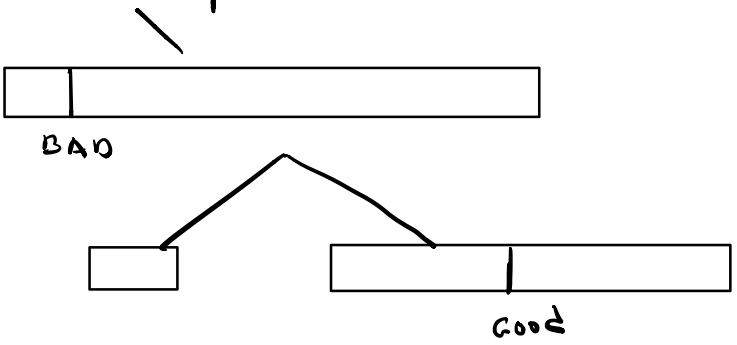
unbalanced BAD

Average Case analysis (just intuition)



$P(\text{GOOD}) = P(\text{BAD}) = \frac{1}{2}$

Benefict of a good pivot >>
penulty of a Bad pivot



by selecting a bad pivot
we just waste a
recursive call

Q5 expected running time is $O(n \log n)$