Algorithms and Data Structures 2

7 – QUICKSORT

Dr Michele Sevegnani

School of Computing Science
University of Glasgow

michele.sevegnani@glasgow.ac.uk

Outline

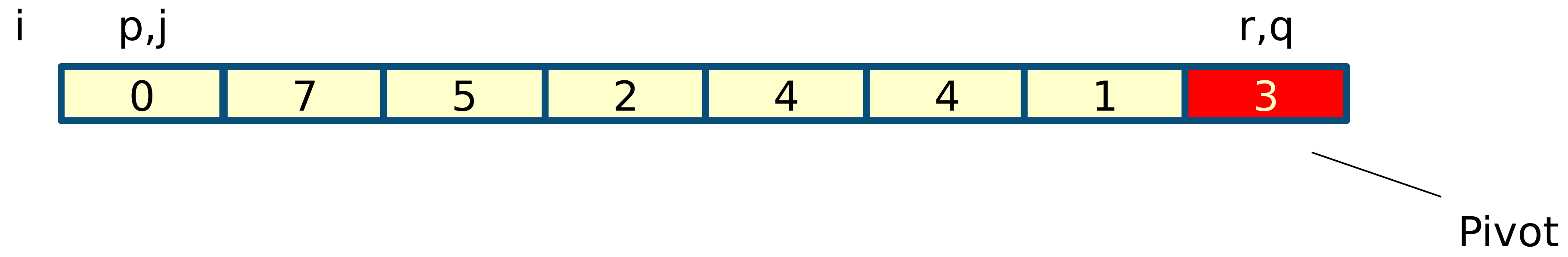
- **QUICKSORT**
  - Properties
  - Alternative partitioning schemes

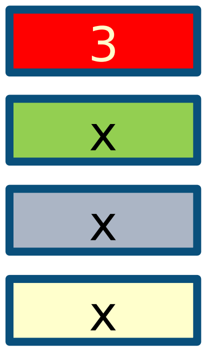
QUICKSORT

- **Efficient divide-and-conquer sorting algorithm**
  - Originally invented by Hoare in 1962
  - Implementation details explained by Sedgewick in “Implementing quicksort programs”, Communications of the ACM, 1978
- **It operates as follows to sort a subarray A[p..r]**
  - Divide: Pick an index q and partition the array in two subarrays A[p..q-1] and A[q+1..r] such that A[p..q-1] contains all the elements less than or equal to A[q], which is less than or equal to each element of A[q+1..r]
  - Conquer: Sort subarrays A[p..q-1] and A[q+1..r] recursively using QUICKSORT
  - Combine: no work is needed as the entire array is already sorted
- **The key operation of the QUICKSORT algorithm is the partitioning of the input array in the Divide step**

Partition example

- **Input array is A[0,7,5,2,4,4,1,3] p=0 and r=7**
- **Select q = r**
  - This is only one possible partitioning scheme: we will study other methods later in this lecture





  - Pivot A[q]=3
  - Elements x ≤ 3
  - Elements x > 3
  - Unrestricted elements

Partition example

- **Input array is A[0,7,5,2,4,4,1,3] p=0 and r=7**
- **Select q = r**
  - This is only one possible partitioning scheme: we will study other methods later in this lecture

p,i	j						r,q
0	7	5	2	4	4	1	3

  - $0 \leq 3$, increase i, swap A[i] with A[j] and then increase j (swap 0 with itself in this case)
  - Expand green region

Partition example

- **Input array is A[0,7,5,2,4,4,1,3] p=0 and r=7**
- **Select q = r**
  - This is only one possible partitioning scheme: we will study other methods later in this lecture

p,i		j					r,q
0	7	5	2	4	4	1	3

  - 7 > 3, increase j
  - Expand grey region

Partition example

- **Input array is A[0,7,5,2,4,4,1,3] p=0 and r=7**
- **Select q = r**
  - This is only one possible partitioning scheme: we will study other methods later in this lecture

p,i			j				r,q
0	7	5	2	4	4	1	3

  - 5 > 3, increase j
  - Expand grey region

Partition example

- **Input array is A[0,7,5,2,4,4,1,3] p=0 and r=7**
- **Select q = r**
  - This is only one possible partitioning scheme: we will study other methods later in this lecture

p	i			j			r,q
0	2	5	7	4	4	1	3

  - $2 \leq 3$, increase i, swap A[i] with A[j] and then increase j
  - Expand green region

Partition example

- **Input array is A[0,7,5,2,4,4,1,3] p=0 and r=7**
- **Select q = r**
  - This is only one possible partitioning scheme: we will study other methods later in this lecture

p	i				j		r,q
0	2	5	7	4	4	1	3

  - 4 > 3, increase j
  - Expand grey region

Partition example

- **Input array is A[0,7,5,2,4,4,1,3] p=0 and r=7**
- **Select q = r**
  - This is only one possible partitioning scheme: we will study other methods later in this lecture

p	i					j	r,q
0	2	5	7	4	4	1	3

  - 4 > 3, increase j
  - Expand grey region

Partition example

- **Input array is A[0,7,5,2,4,4,1,3] p=0 and r=7**
- **Select q = r**
  - This is only one possible partitioning scheme: we will study other methods later in this lecture

p		i					r,q
0	2	1	7	4	4	5	3

  - 1 ≤ 3, increase i, swap A[i] with A[j]
  - Expand green region

Partition example

- **Input array is A[0,7,5,2,4,4,1,3] p=0 and r=7**
- **Select q = r**
  - This is only one possible partitioning scheme: we will study other methods later in this lecture

p		i					r,q
0	2	1	3	4	4	5	7

  - No more unrestricted elements left
  - Swap A[i+1] with A[r] to place the pivot in the middle
  - Termination

PARTITION

- **Input: Array A and three indexes p, r for A such that $p \le r$**
  - No assumptions on the input
- **Output: index q such that**
  - $A[p..q-1] \le A[q] < A[q+1..r]$

- **A is rearranged in place**
- **Running time is O(n)**

```
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
```

QUICKSORT

- **Input: Array A and two indexes p, r for A such that p ≤ r**
- **Output: sorted array A[p..r]**

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

- **To sort an array A with n elements the initial call is QUICKSORT(A,0,n-1)**
- **After each partition, the first recursive call operates on the green region while the second call operates on the grey region of A**

QUICKSORT recursion tree

- **Try to derive the recursion tree of QUICKSORT(A,0,6) with A= [6,5,0,4,1,8,3]**

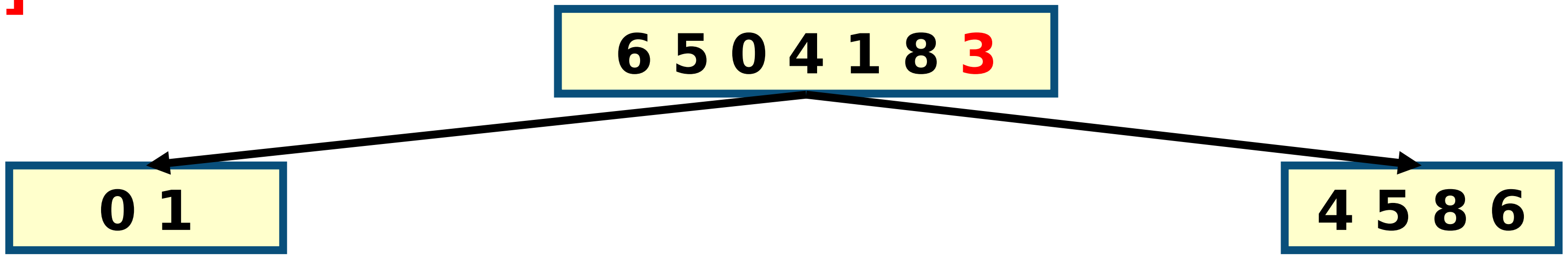
QUICKSORT recursion tree

- **Try to derive the recursion tree of QUICKSORT(A,0,6) with A= [6,5,0,4,1,8,3]**

6 5 0 4 1 8 3

QUICKSORT recursion tree

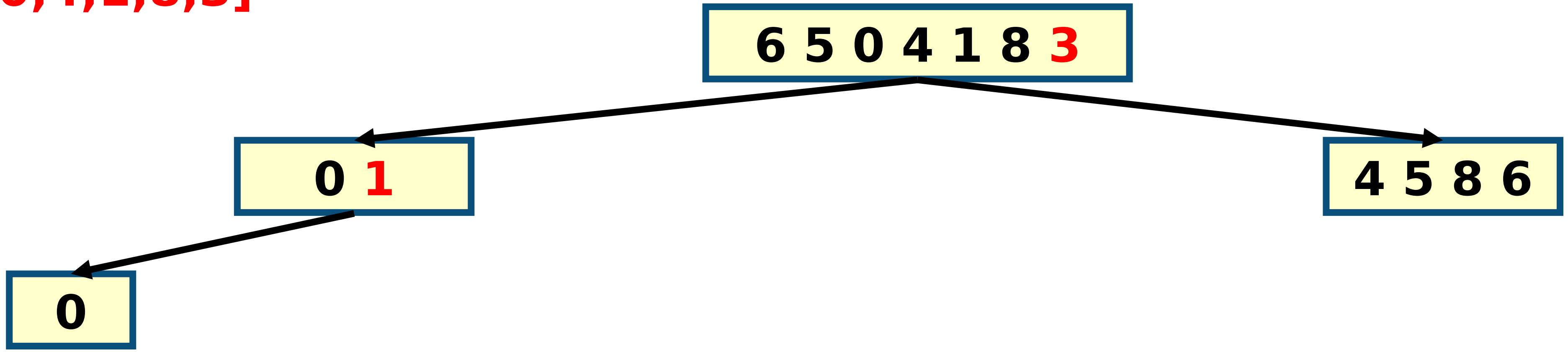
- **Try to derive the recursion tree of QUICKSORT(A,0,6) with A= [6,5,0,4,1,8,3]**



  - Partition of [6,5,0,4,1,8,3] with pivot [3] yields [0,1] and [4,5,8,6]

QUICKSORT recursion tree

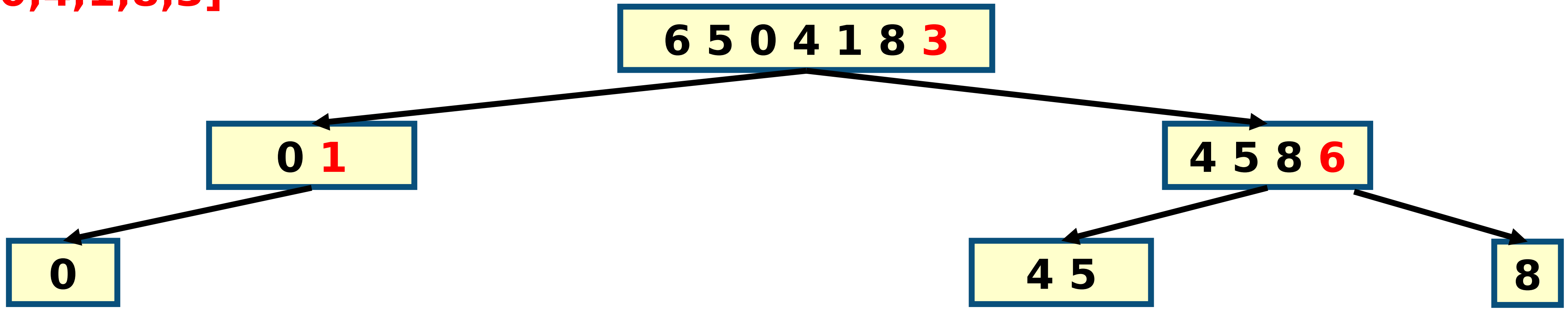
- Try to derive the recursion tree of QUICKSORT(A,0,6) with A= [6,5,0,4,1,8,3]



  - Partition of [0,1] with pivot [1] yields [0] and []

QUICKSORT recursion tree

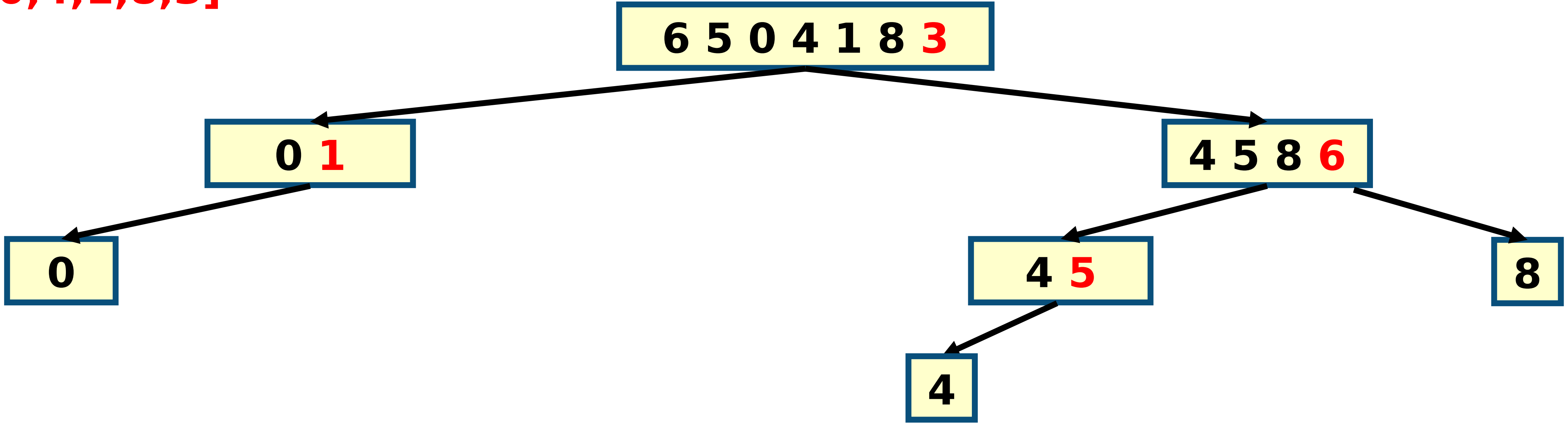
- **Try to derive the recursion tree of QUICKSORT(A,0,6) with A= [6,5,0,4,1,8,3]**



  - Partition of [4,5,8,6] with pivot [6] yields [4,5] and [8]

QUICKSORT recursion tree

- **Try to derive the recursion tree of QUICKSORT(A,0,6) with A= [6,5,0,4,1,8,3]**



  - Partition of [4,5] with pivot [5] yields [4] and []
  - Termination. A sorted in place: [0,1,3,4,5,6,8]
  - Remember the pivot is swapped with the first element of the grey region at each recursive call

Running time

- **Cost of partitioning is O(n)**
- **Best case**
  - Pivot in the middle - median
  - Subarrays exactly half size of the original
  - Recurrence equation as for MERGE-SORT: $T(n) = 2T(n/2) + O(n) = O(n \log n)$
- **Worst case**
  - Unbalanced partitioning in each recursive call
  - One subarray with n-1 elements and one with 0 elements
  - Recurrence equation: $T(n) = T(n-1) + T(0) + O(n) = O(n^2)$
  - This happens when input array is already sorted. Remember INSERTION-SORT is O(n) in this case!

> Prove with the iterative method

Average case running time

- **We need to consider all possible permutation of array and calculate time taken to sort each permutation**
  - Difficult proof involving the average number of comparisons performed
- **We show informally that the average case running time is much closer to the best case than to the worst case**
  - Suppose that PARTITION always produces an unbalanced 9-to-1 split
  - Recurrence in this case is $T(n) = T(9n/10) + T(n/10) + cn$
  - By analysing the recursion tree we note that
    1. Each level has cost cn until tree depth $\log_{10} n = O(\log n)$
    2. Then, the levels have cost at most cn until tree depth $\log_{10/9} n = O(\log n)$
    3. Summing each level we still obtain $O(n \log n)$

Some alternative partitioning schemes

- **Choose the middle element**
  - Pros: Simple to code, fast to calculate, but slightly slower than standard PARTITION
  - Cons: Still can degrade to $O(n^2)$. Easy for someone to construct an array that will cause it to degrade to $O(n^2)$
- **Choose the median of three (p,q,r)**
  - Pros: Fairly simple to code, reasonably fast to calculate, but slightly slower than previous methods
  - Cons: Still can degrade to $O(n^2)$. Fairly easy for someone to construct an array that will cause it to degrade to $O(n^2)$
- **Choose the pivot randomly**
  - Pros: Simple to code. Harder for someone to construct an array that will cause it to degrade to $O(n^2)$
  - Cons: Selecting a random pivot is fairly slow. Still can degrade to $O(n^2)$.

Improvements on standard QUICKSORT

- **Cutoff to INSERTION-SORT (as in MERGE-SORT). Alternatively:**
  - When calling QUICKSORT on a subarray with fewer than k elements, return without sorting the subarray
  - After the top-level call to QUICKSORT returns, run INSERTION-SORT on the entire array to finish the sorting process
  - Taking advantage of the fast running time of INSERTION-SORT when its input is “nearly” sorted
- **Tail call optimisation convert the code so that it makes only one recursive call**
  - Usually good compilers do that for us
- **Iterative version with the help of an auxiliary stack**

Improvements on standard QUICKSORT (cont.)

- **In 3-WAY-QUICKSORT, an array A[p..r] is divided in 3 parts**
  - A[p..i] elements less than pivot
  - A[i+1..j-1] elements equal to pivot
  - A[j..r] elements greater than pivot.
  - Based on Dutch National Flag algorithm
  - Good when input has many duplicates

Summary

- **QUICKSORT**
  - Properties
  - Alternative partitioning schemes