Quicksort Chapter 7 of Cormen's book Giulia Bernardini giulia.bernardini@units.it

Algorithmic Design a.y. 2024/2025

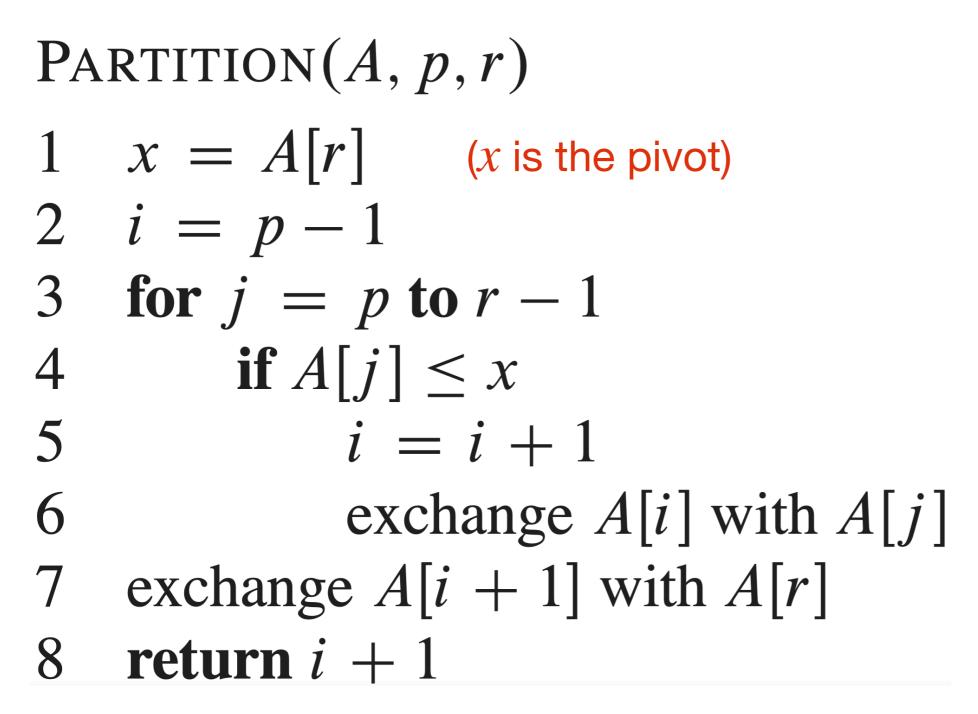
Quicksort

QUICKSORT(A, p, r)

1 **if** p < rq = PARTITION(A, p, r)QUICKSORT(A, p, q - 1)QUICKSORT(A, q + 1, r)

Quicksort is a divide-and-conquer algorithm. Al the work is done in the divide step.

Basic Quicksort



Partition is an in-place procedure. https://visualgo.net/en/sorting



Please go to **www.wooclap.com**, use the code **BERNARDINI03** and answer the question (it is anonymous unless you decide to use your name). You do not need to create an account!



Randomized Quicksort

RANDOMIZED-PARTITION(A, p, r)

- 1 i = RANDOM(p, r)
- 2 exchange A[r] with A[i]
- 3 **return** PARTITION(A, p, r)

The new quicksort calls RANDOMIZED-PARTITION in place of PARTITION:

RANDOMIZED-QUICKSORT(A, p, r)

- 1 **if** *p* < *r*
- 2 q = RANDOMIZED-PARTITION(A, p, r)
- 3 RANDOMIZED-QUICKSORT (A, p, q 1)
- 4 RANDOMIZED-QUICKSORT (A, q + 1, r)

Counting and Radix Sort Chapters from 8.1 to 8.3 of Cormen's book

> Giulia Bernardini giulia.bernardini@units.it

> > Algorithmic Design a.y. 2024/2025

Lower Bounds on Sorting

Comparison model: the only operations are comparisons. The running time of an algorithm is the number of comparisons it does.

We prove that any sorting algorithm requires $\Omega(n \log n)$ comparisons in the worst case.

Lower Bounds on Sorting

Decision Tree

Internal node

Leaf

Root-to-leaf path

Length of the root-to-leaf path

Height of the tree

Algorithm

Binary decision

Answer found

Single execution

Running time of one execution

Worst-case running time



Please go to **www.wooclap.com**, use the code **BERNARDINI03** and answer the question (it is anonymous unless you decide to use your name). You do not need to create an account!



Lower Bounds on Sorting

Theorem: Given *n* elements, sorting them requires $\Omega(n \log n)$ time (comparisons) in the worst case.

Proof:

- The decision tree is binary
- Its height is at least log(number of leaves)
- The number of leaves is at least the number of permutations of *n* elements

Counting Sort

COUNTING-SORT(A, B, k)

- 1 let C[0..k] be a new array
- 2 **for** i = 0 **to** k
- 3 C[i] = 0

5

- 4 for j = 1 to A.length
 - C[A[j]] = C[A[j]] + 1
- 6 // C[i] now contains the number of elements equal to i.
- 7 **for** i = 1 **to** k
- 8 C[i] = C[i] + C[i-1]
- 9 // C[i] now contains the number of elements less than or equal to i.
- 10 for j = A.length downto 1
- 11 B[C[A[j]]] = A[j]
- 12 C[A[j]] = C[A[j]] 1

Efficient to sort *n* integers between 0 and *k*, with k = O(n).

Radix Sort

RADIX-SORT(A, d)

- 1 **for** i = 1 **to** d
- 2 use a stable sort to sort array A on digit i