Quicksort

Chapter 7 of Cormen's book

Giulia Bernardini
giulia.bernardini@units.it

Algorithmic Design
a.y. 2022/2023

Quicksort

```
QUICKSORT(A, p, r)
if p < r
    q = 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.

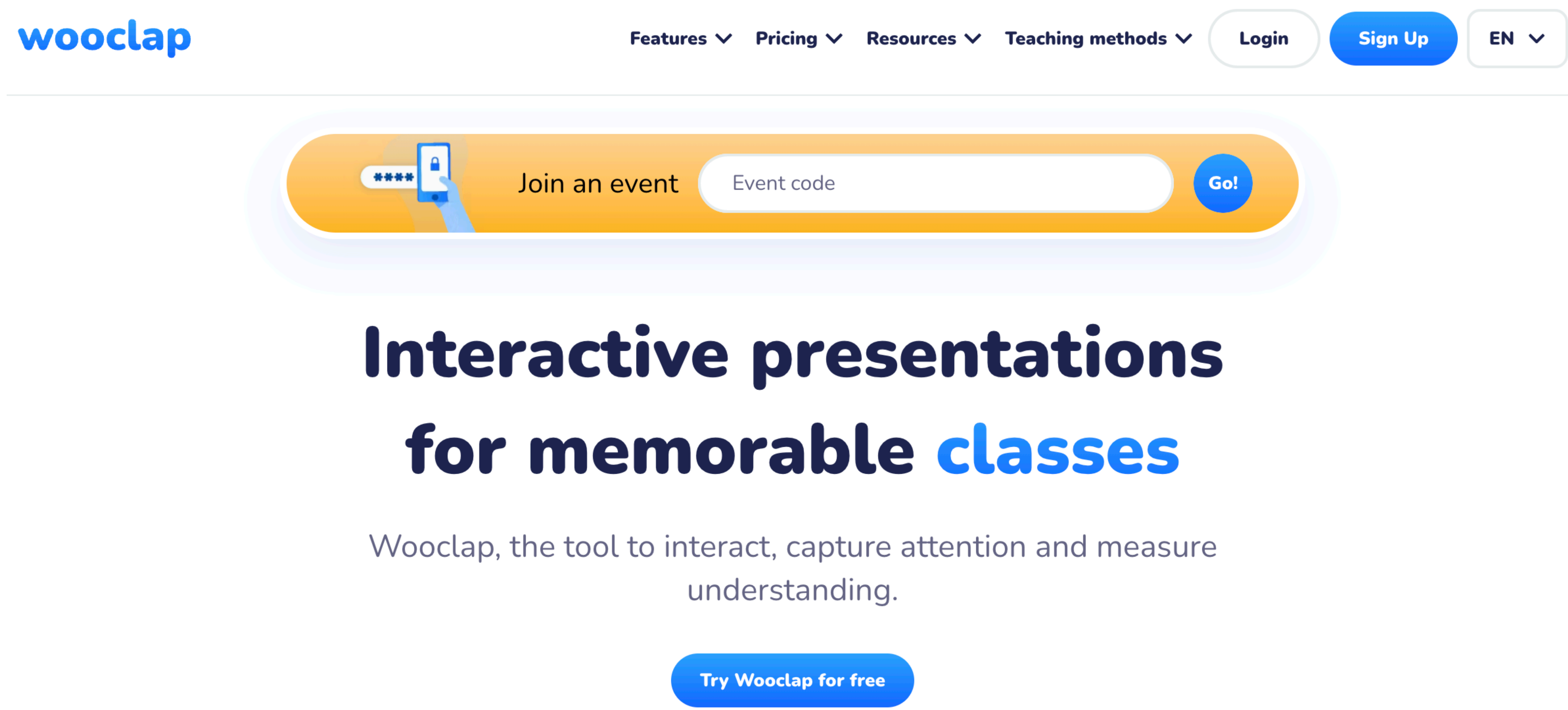
Basic Quicksort

```
PARTITION(A, p, r)
x = A[r]            (x is the pivot)
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

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

Quiz Time

Please go to **www.wooclap.com,** use the code **BERNARDINI3** 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)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

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

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    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. 2022/2023

Lower Bounds on Sorting

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

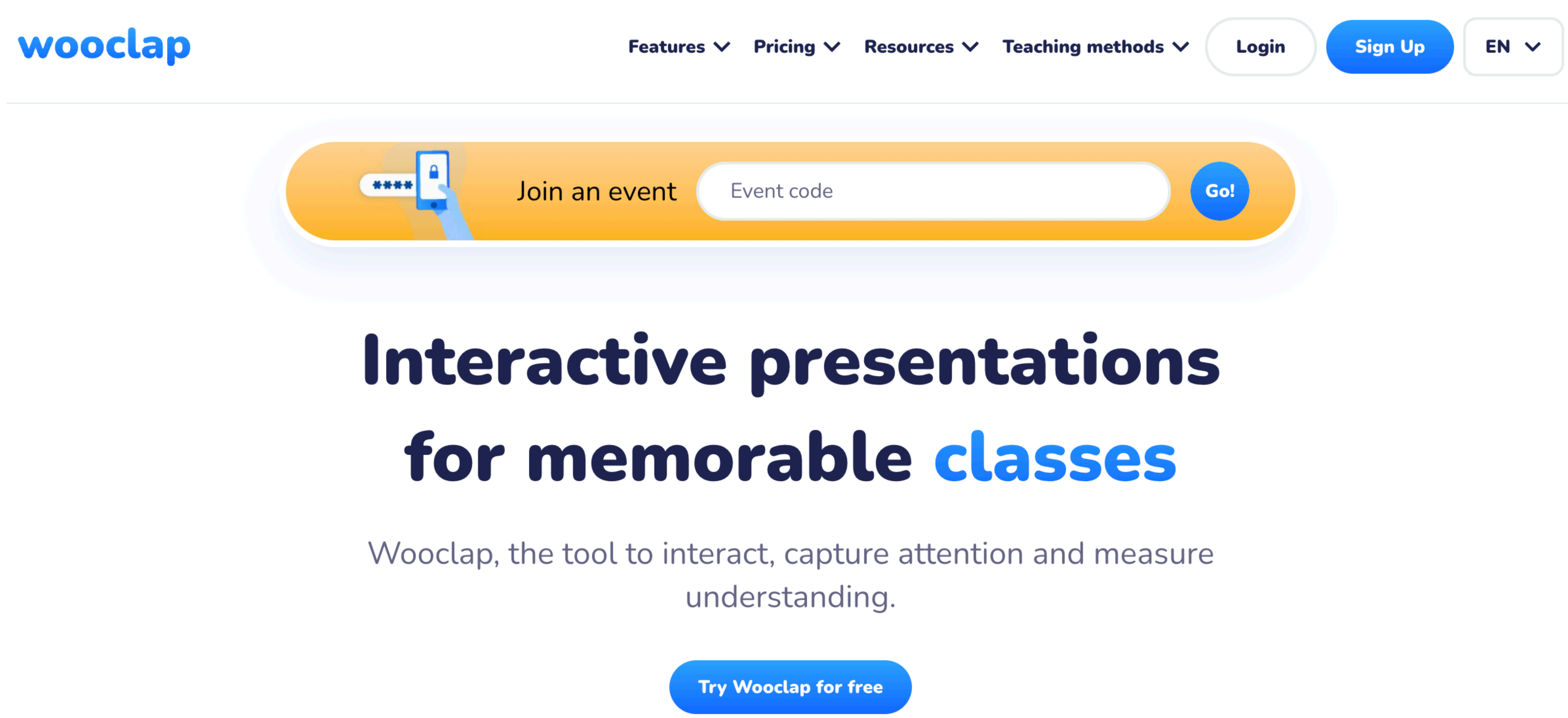
We prove that any sorting algorithm requires $\Omega(n \log n)$ comparisons in the worst case.

Lower Bounds on Sorting

Decision Tree	Algorithm
Internal node	Binary decision
Leaf	Answer found
Root-to-leaf path	Single execution
Length of the root-to-leaf path	Running time of one execution
Height of the tree	Worst-case running time

Quiz Time

Please go to **www.wooclap.com,** use the code **BERNARDINI3** 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)
    let C[0..k] be a new array
    for i = 0 to k
        C[i] = 0
    for j = 1 to A.length
        C[A[j]] = C[A[j]] + 1
    // C[i] now contains the number of elements equal to i.
    for i = 1 to k
        C[i] = C[i] + C[i - 1]
    // C[i] now contains the number of elements less than or equal to i.
    for j = A.length downto 1
        B[C[A[j]]] = A[j]
        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)

for $i = 1$ **to** d

    use a stable sort to sort array A on digit i