INSERTION SORT, MERGE SORT

Finding a median

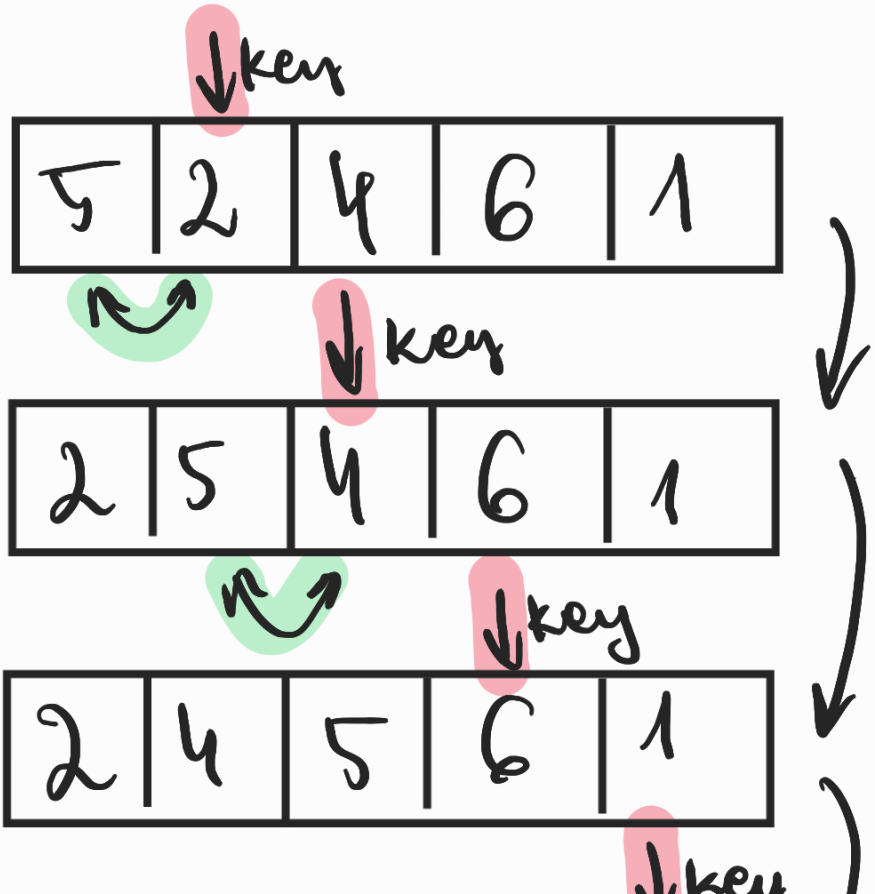
array $A[0:n] \rightarrow B[0:n]$

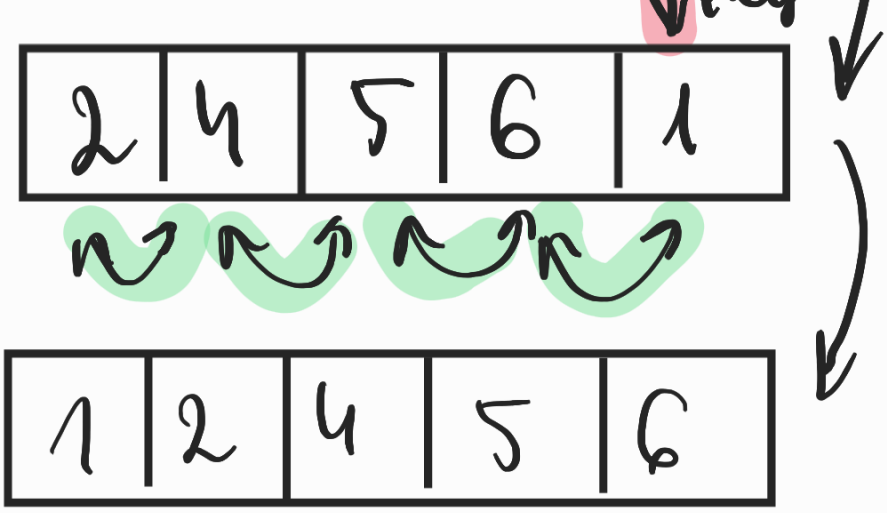
unsorted

INSERTION SORT

For $i = 1, 2 \ldots n$

insert $A[i]$ into sorted array $A[0:i-1]$ by pairwise swaps down to the correct position





$\Theta(n)$ steps (key positions).

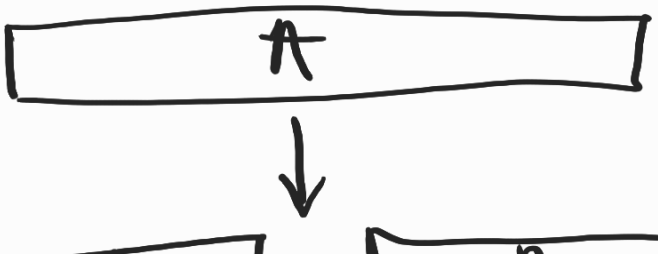
Each step is $\Theta(n)$ swaps → compares & swaps

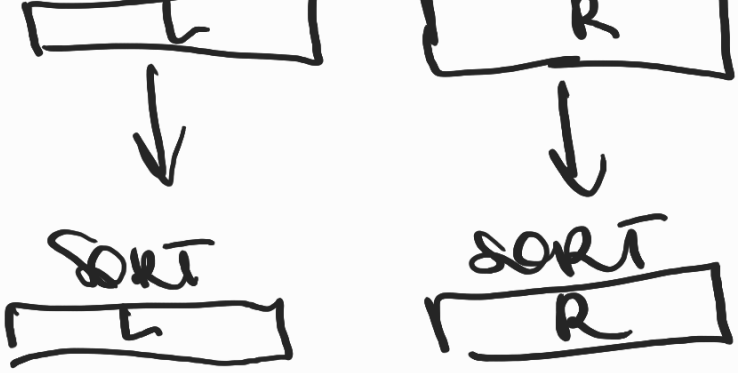
Compares >> swaps $\Theta(n^2)$

Do a binary search on A[0: i-1] already sorted in $\Theta(\lg i)$ time THEN $\Theta(n \cdot \lg n)$

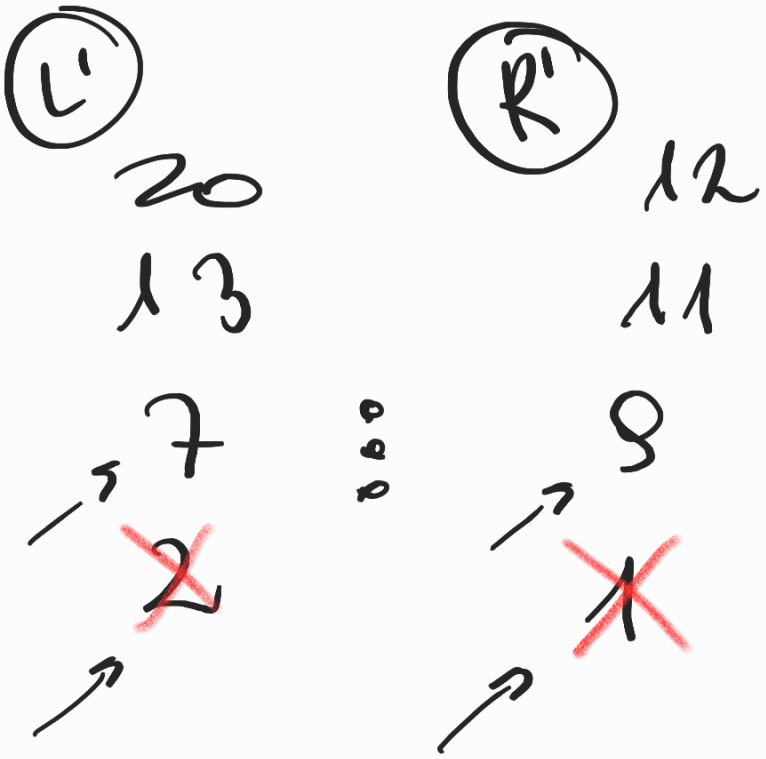
MERGE SORT

Divide & Conquer





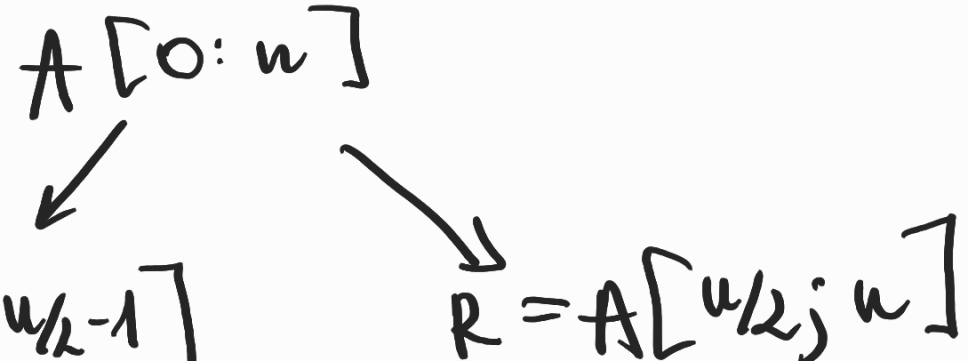
MERGE Two sorted arrays as input



Two finger algo:

1 2 7 9 11 12 13 20

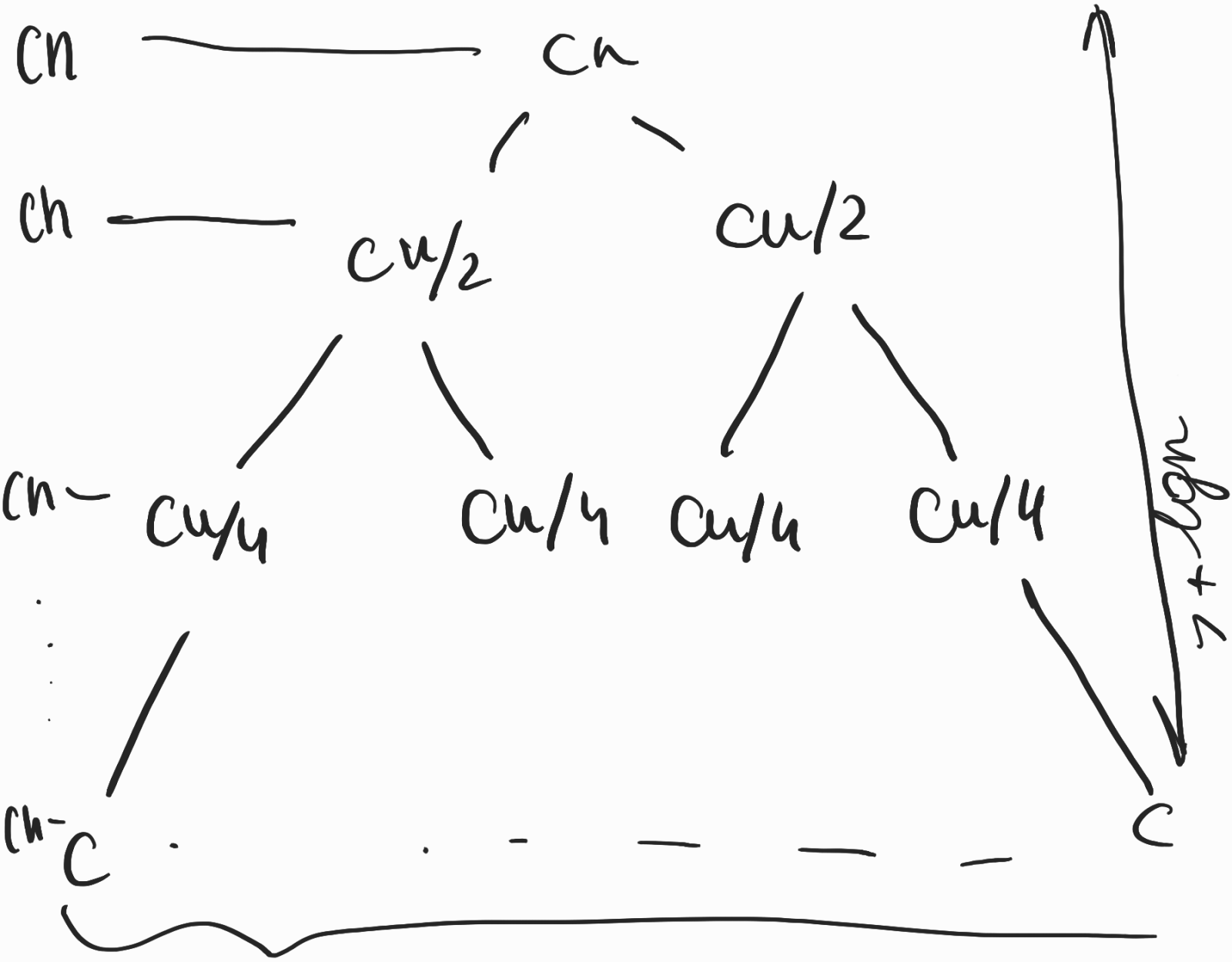
complexity $\Theta(n)$



Complexity:

$$T(n) = \underbrace{c_1}_{\text{divide}} + 2T(n/2) + \underbrace{c \cdot n}_{\text{merge}} =$$

$$= 2T(n/2) + \Theta(n) + \Theta(1)$$



n leaves

$$T(n) = (1 + \log n) * c_n = \Theta(n \log n)$$

MERGE SORT

$\Theta(n)$ auxiliary space

in-place sort => $\Theta(1)$ auxiliary space

pomocniczy

IN-PLACE MERGE-SORT

MERGESORT in python

$2.2\, n \log(n)$ µs

INSERTATION in python: $0.2n^2$ µs

C: $0,01n^2$ µs

MERGESORT VS INSERTATION SORT

TIME COMPLEXITY:

Mergesort the worstcase:

average & Best

case is $O(n \lg n)$

Insertation sort — the worst: $O(n^2)$
average: $O(n^2)$
best: $O(n)$

$n \lg n$ when we use binary search

SPACE COMPLEXITY:

Mergesort: $O(n)$ auxiliary space

Insersort: $O(1)$ sorts array using an extra variable

DATASETS:

Mergesort: preffered for huge data sets.

Insertsort: preferred for fewer elements

EFFICIENCY: (WEBAJNESS)

Merge sort is efficient in terms of time

Insert sort is efficient in terms of space.

STABILITY:

Mergesort is stable as two elements with equal value appear in the same order in sorted output as they were in the input unsorted array

Insertionsort takes $O(n^2)$ time on both (array and linked list).

If the CPU has an efficient memory block move function then the array may be quicker.