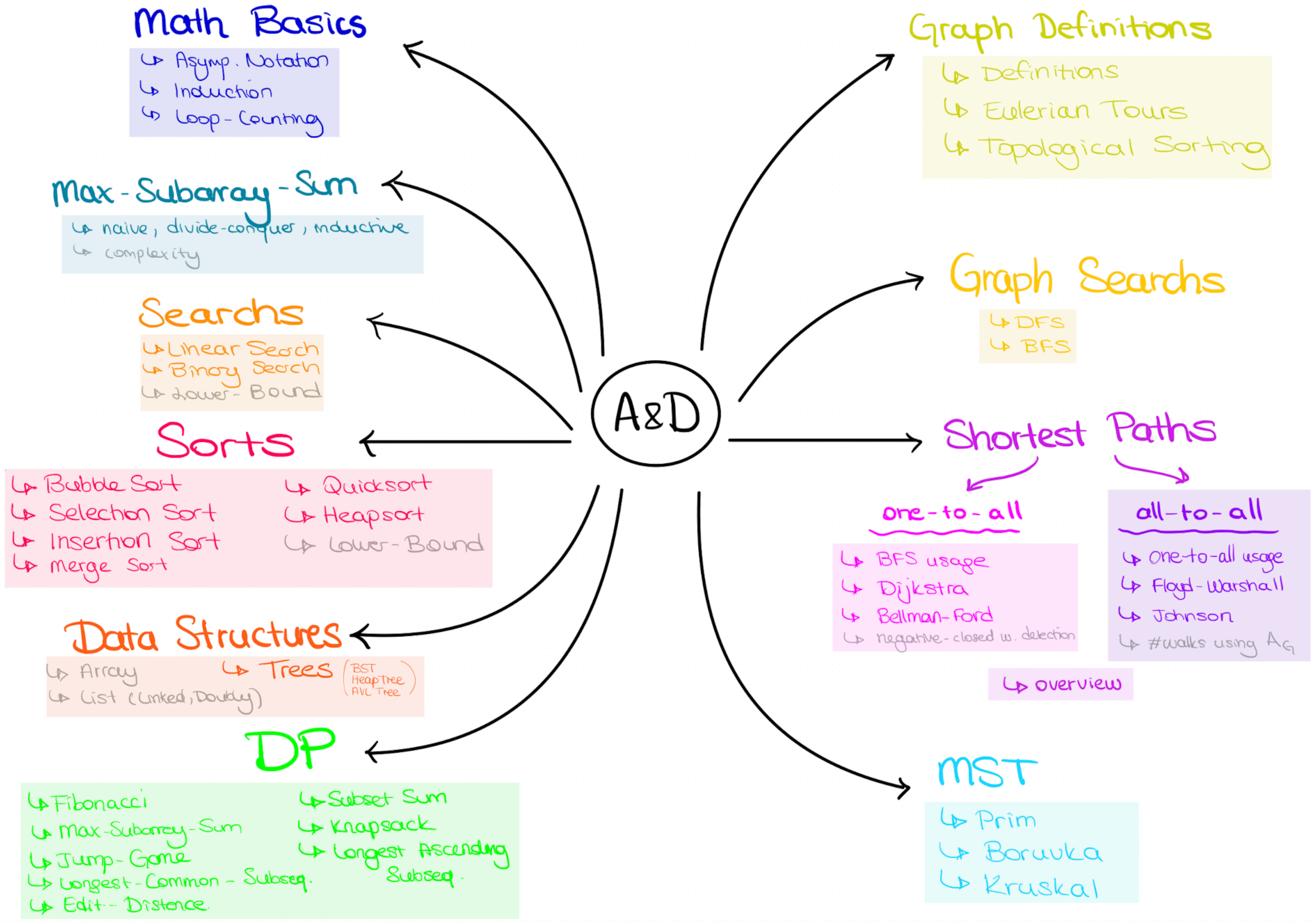
A&D

Exercise Session 5

Nil Ozer

A&D Overview



A&D

Math Basics
- Asymp. Notation
- Induction
- Loop-Counting

Max-Subarray-Sum
- naive, divide-conquer, inductive
- complexity

Searchs
- Linear Search
- Binary Search
- Lower-Bound

Sorts
- Bubble Sort
- Selection Sort
- Insertion Sort
- merge Sort
- Quicksort
- Heapsort
- Lower-Bound

Data Structures
- Array
- List (Linked, Doubly)
- Trees (BST, Heap Tree, AVL Tree)

DP
- Fibonacci
- max-Subarray-Sum
- Jump-Game
- Longest-Common-Subseq.
- Edit-Distance
- Subset Sum
- Knapsack
- Longest Ascending Subseq.

Graph Definitions
- Definitions
- Eulerian Tours
- Topological Sorting

Graph Searchs
- DFS
- BFS

Shortest Paths

one-to-all
- BFS usage
- Dijkstra
- Bellman-Ford
- negative-closed w. detection

all-to-all
- one-to-all usage
- Floyd-Warshall
- Johnson
- #walks using A_G

- overview

MST
- Prim
- Boruuka
- Kruskal

Outline

- Quiz

- Exercise Sheet 3 Bonus Feedback
- Exercise Sheet 4 - non bonus

- Sorts II
- Sorting Algorithms Kahoot !

- Data Structures I

Quiz

Exercise Sheet 3

Bonus Feedback

- Follow the task description.
  - If it says use the definition, use the definition !
  - Check the master solution to learn how to argue with definitions

- Loop Counting
  - You can switch to Θ-Approximation earlier, but this is risky !
    - Check the master solution

- Pay attention to the little notes. Keep up the good work !

Exercise Sheet 4

Non Bonus

- 4.1 Applying the master theorem

- 4.2 Asymptotic Notation Quiz

Exercise Sheet 4

Peer Grading

- 4.4 this week
- Emails are already sent
- New groups !

Sorts II

Quick Sort

💡 Idea : No merging, Pivot !!!

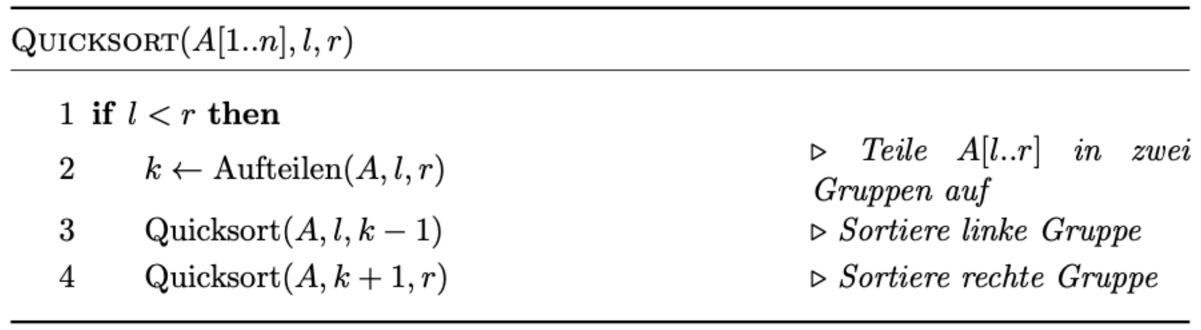
Input : unsorted array Output : sorted array

Runtime: Depends on the pivot !

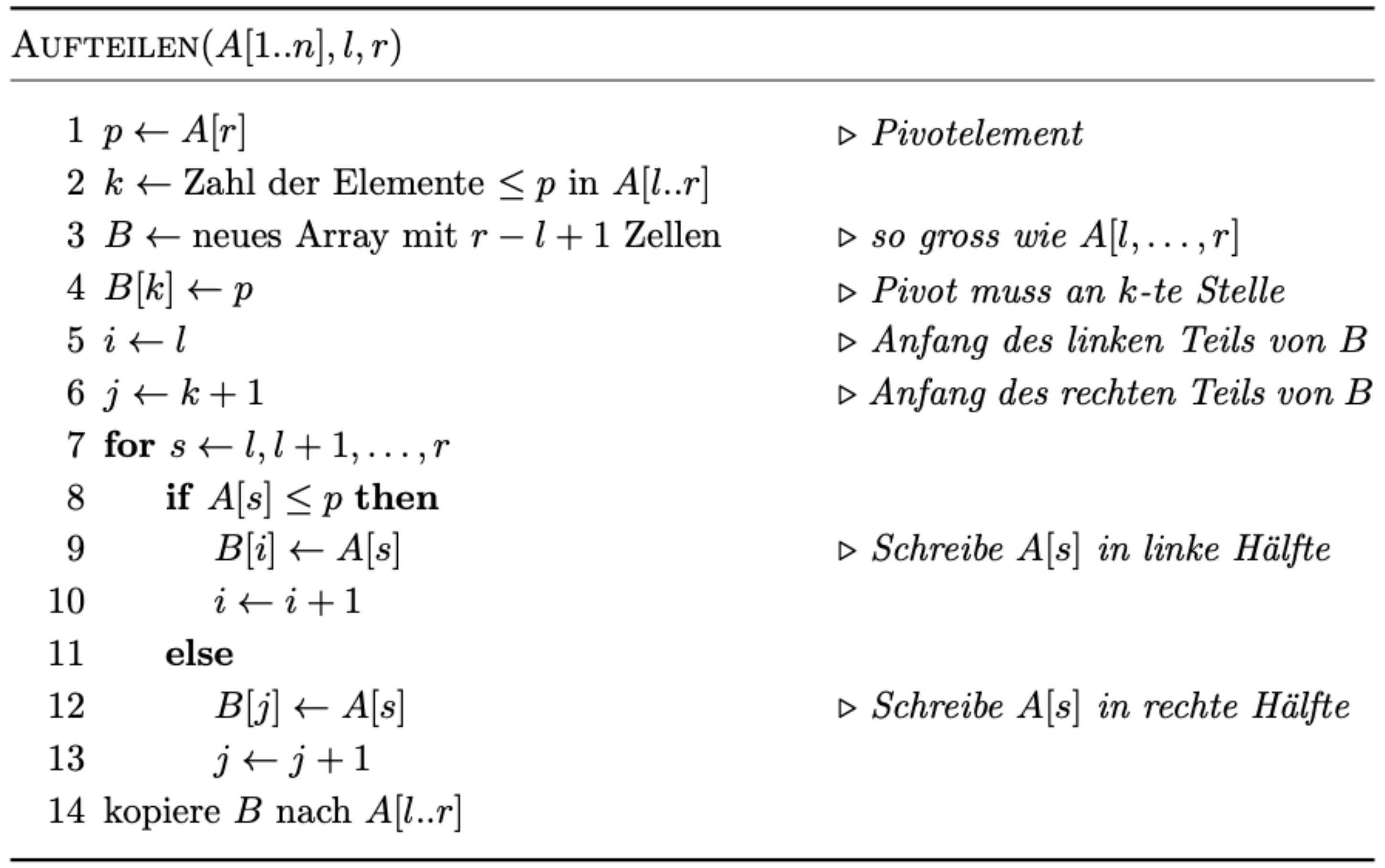
when the pivot element divides the array into two equal halves : $O(n \log n)$

when the smallest or largest element is always chosen as the pivot : (e.g., sorted arrays). $O(n^2)$

Pseudocode :



```
QUICKSORT($A[1..n], l, r$)
if $l < r$ then
    $k \leftarrow$ Aufteilen($A, l, r$)          ▷ Teile $A[l..r]$ in zwei Gruppen auf
    Quicksort($A, l, k-1$)                       ▷ Sortiere linke Gruppe
    Quicksort($A, k+1, r$)                       ▷ Sortiere rechte Gruppe
```

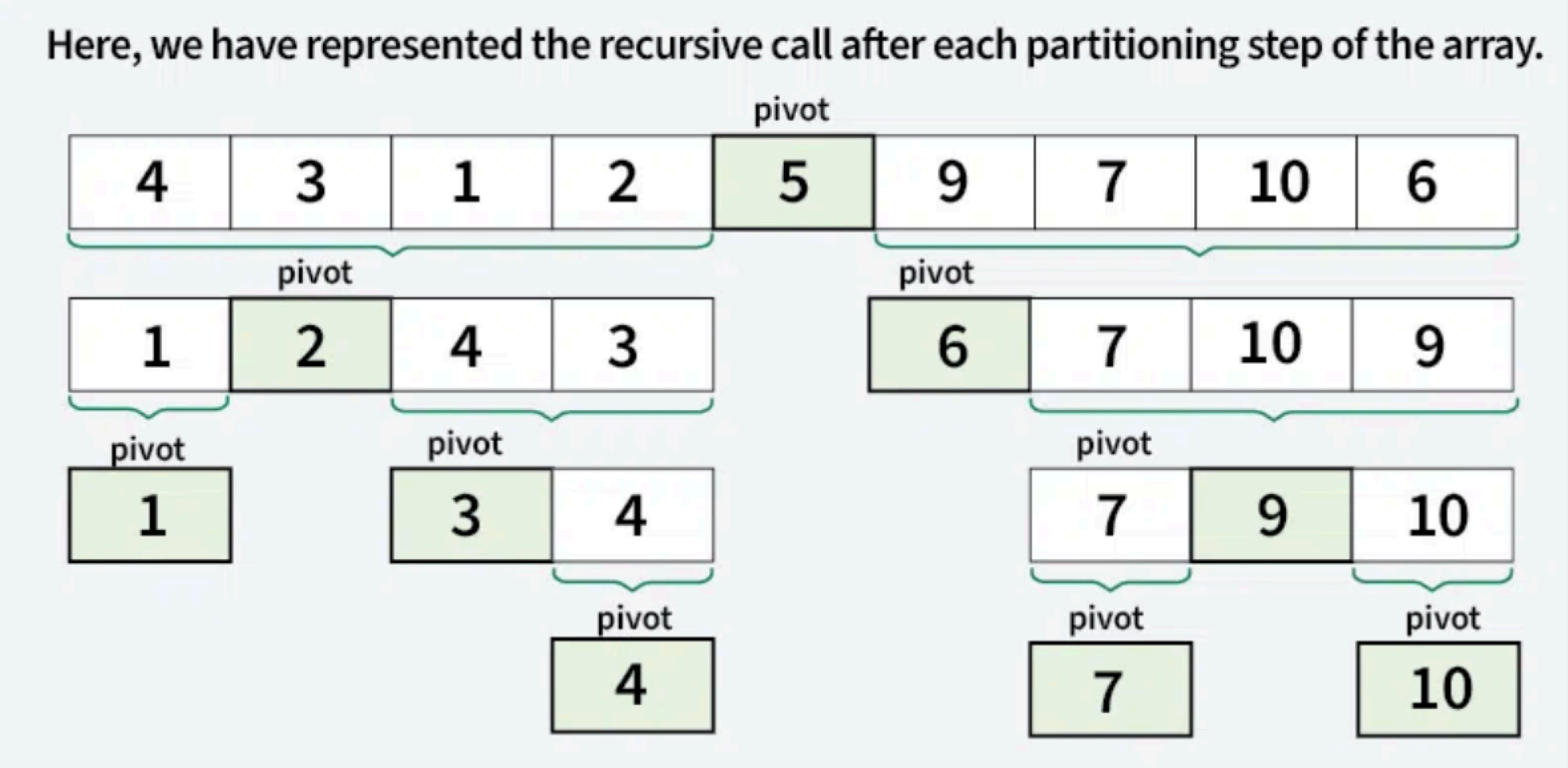


```
AUFTEILEN($A[1..n], l, r$)
$p \leftarrow A[r]$                                        ▷ Pivotelement
$k \leftarrow$ Zahl der Elemente $\leq p$ in $A[l..r]$
$B \leftarrow$ neues Array mit $r - l + 1$ Zellen          ▷ so gross wie $A[l, \dots, r]$
$B[k] \leftarrow p$                                        ▷ Pivot muss an $k$-te Stelle
$i \leftarrow l$                                           ▷ Anfang des linken Teils von $B$
$j \leftarrow k + 1$                                       ▷ Anfang des rechten Teils von $B$
for $s \leftarrow l, l+1, \dots, r$
    if $A[s] \leq p$ then
        $B[i] \leftarrow A[s]$                             ▷ Schreibe $A[s]$ in linke Hälfte
        $i \leftarrow i + 1$
    else
        $B[j] \leftarrow A[s]$                             ▷ Schreibe $A[s]$ in rechte Hälfte
        $j \leftarrow j + 1$
kopiere $B$ nach $A[l..r]$
```

Illustration

Quick Sort

Illustration

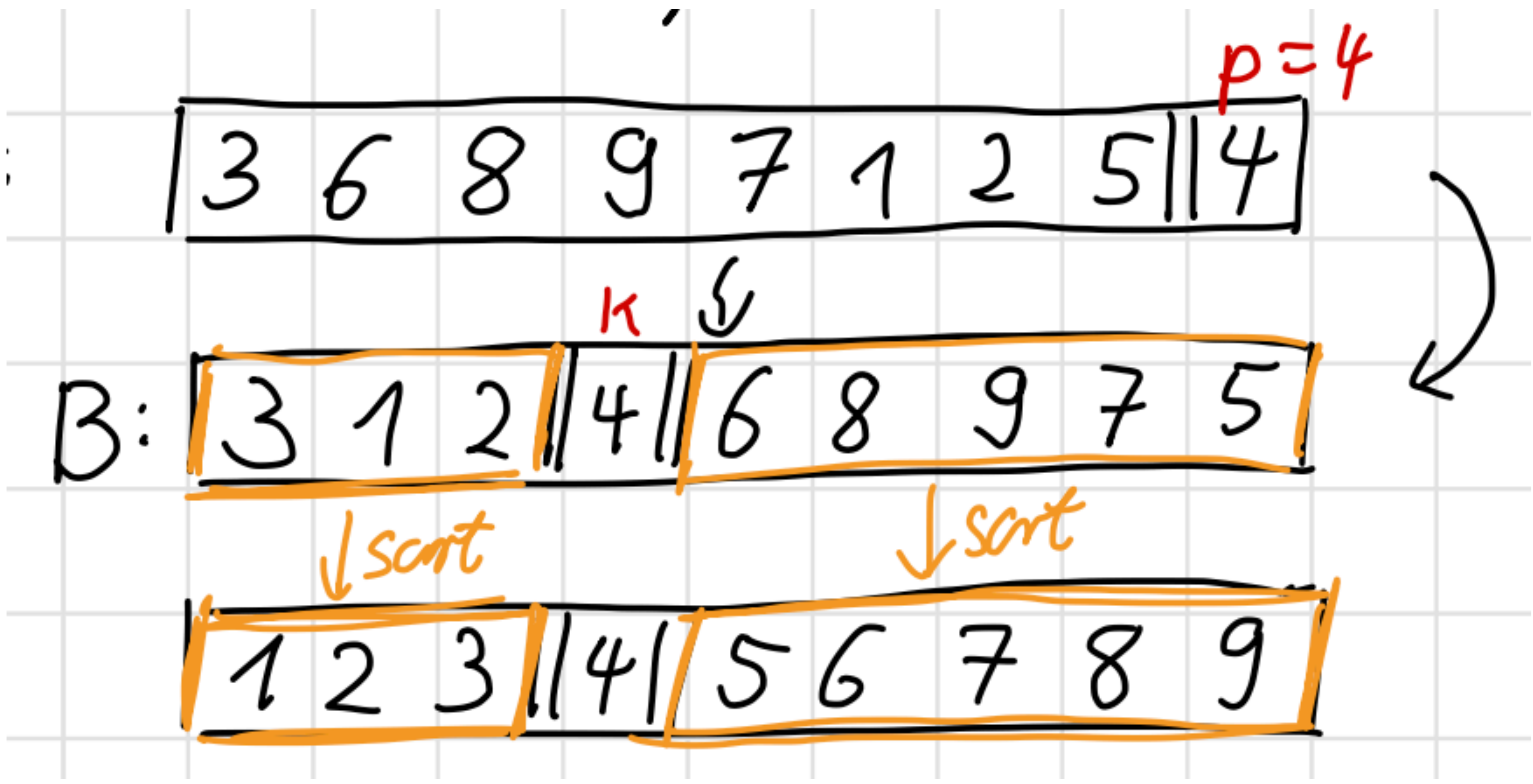


Here, we have represented the recursive call after each partitioning step of the array.

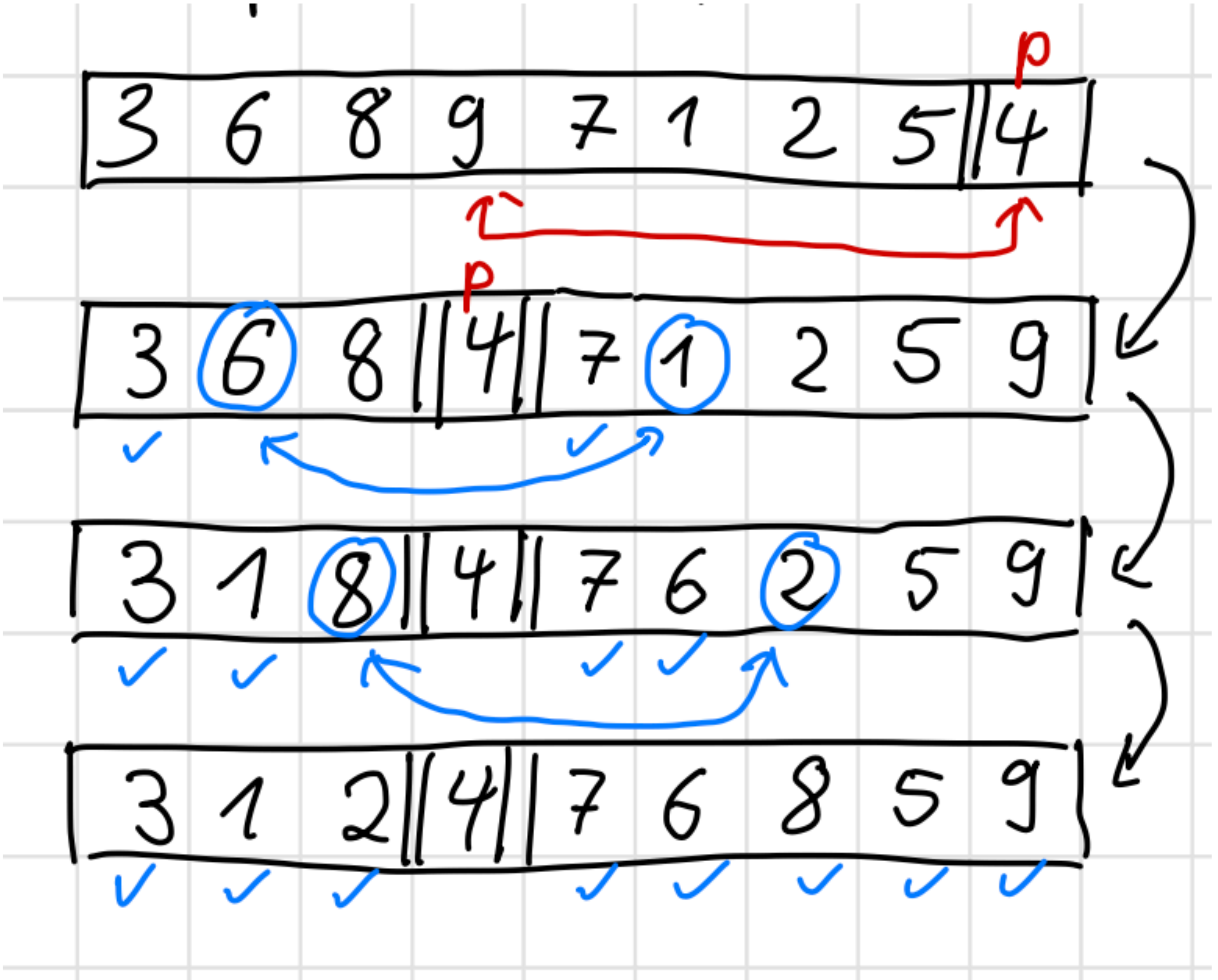
Quick Sort

Illustration

With helper array



In-place



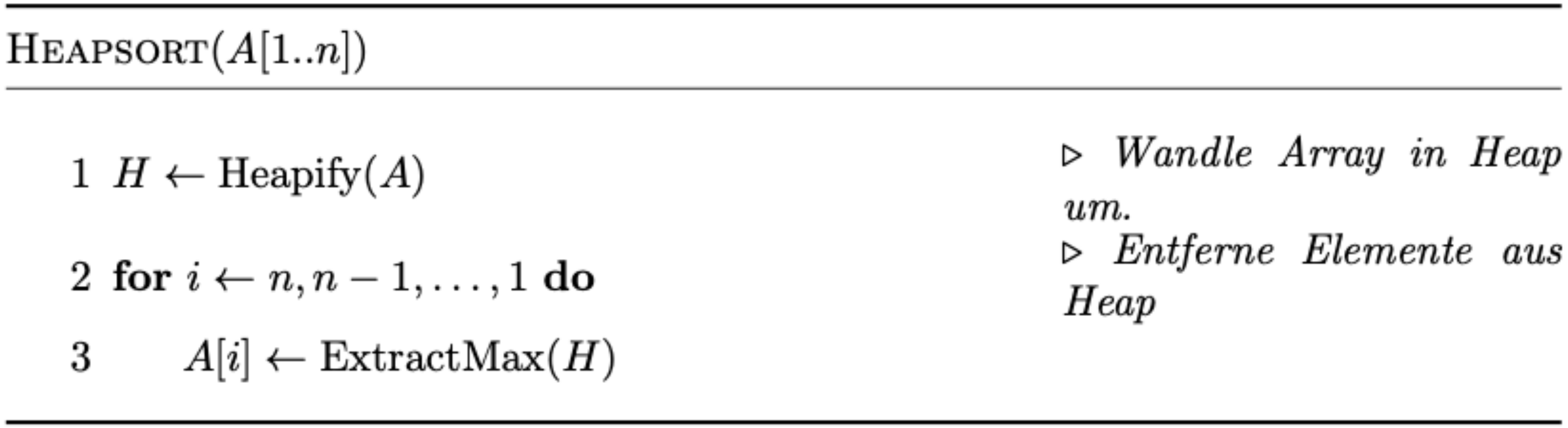
Heap Sort

Idea : Selection sort + Heap ! (Finding maximum faster)

Input : unsorted array   Output : sorted array

Runtime: O(nlogn)

Pseudocode :



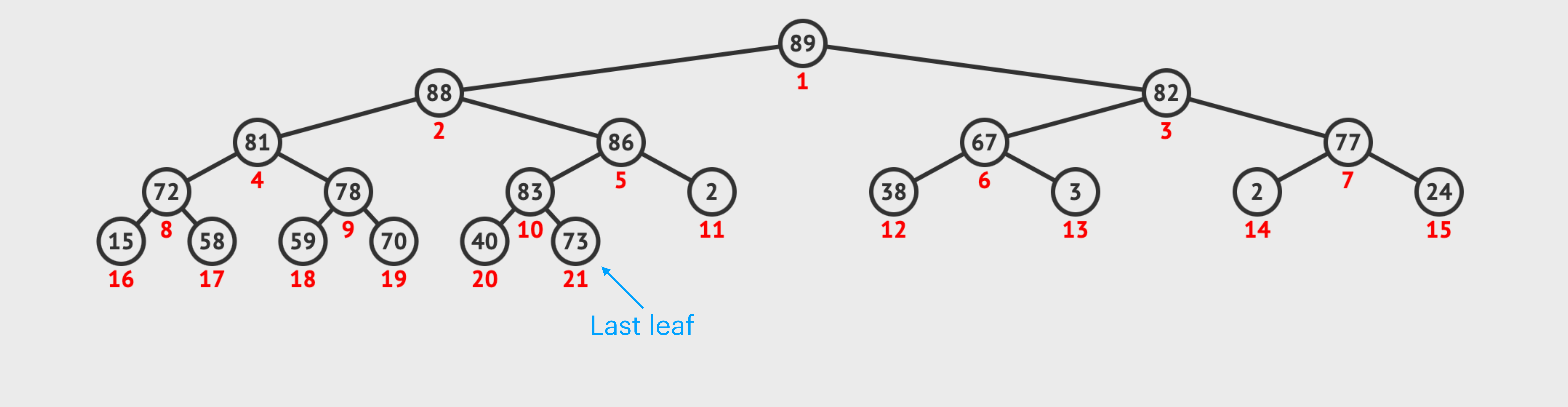
```
HEAPSORT($A[1..n]$)

$H \leftarrow$ Heapify($A$)                      ▷ Wandle Array in Heap um.
for $i \leftarrow n, n-1, \dots, 1$ do           ▷ Entferne Elemente aus Heap
    $A[i] \leftarrow$ ExtractMax($H$)
```

Sorting Algorithms Kahoot

Let’s take a break

Data Structures I

Heap (here : Maxheap)
Terminology



Root Node: The topmost node of the heap. Holds the maximum element !

Parent Node: A node that has one or more child nodes.

Child Node: A node directly connected to another node when moving away from the root.

Leaf Node: A node with no children (located at the bottom level).

Sibling Nodes: Nodes that share the same parent.

Level: The depth or layer of the node, where the root is at level 0.

Height: The longest path from the root node to a leaf.

Heap (here : Maxheap)

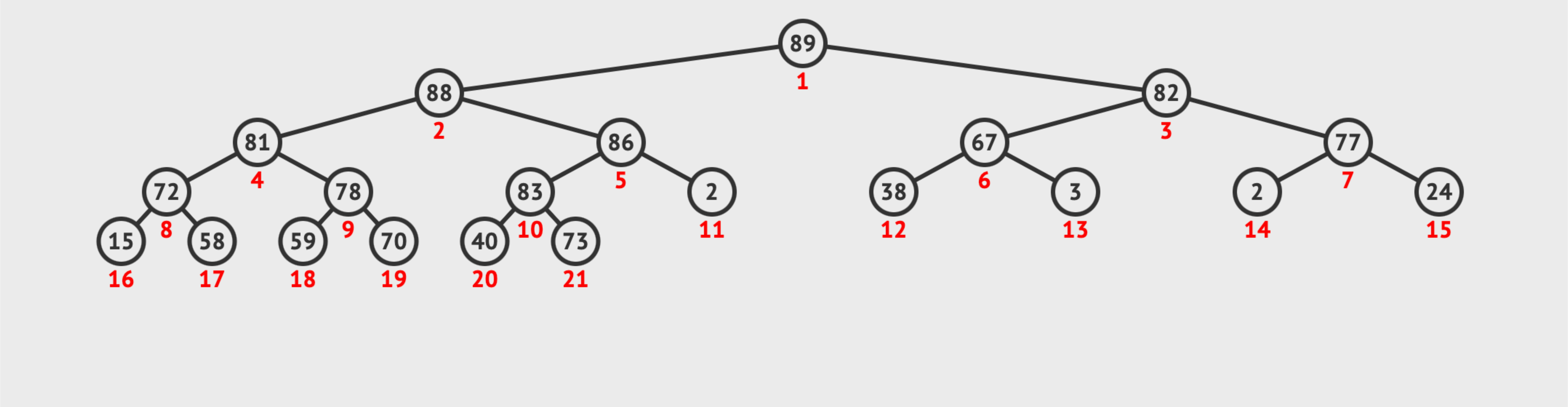
Heap Condition

For every node n in the heap, the value of the parent is greater than or equal to the value of n

val(n) ≥ val(children(n) for all n

Heap (here : Maxheap)

Heap Condition : val(n) ≥ val(children(n))



Root Node: The topmost node of the heap. Holds the maximum element !

Parent Node: A node that has one or more child nodes.

Child Node: A node directly connected to another node when moving away from the root.

Leaf Node: A node with no children (located at the bottom level).

Sibling Nodes: Nodes that share the same parent.

Level: The depth or layer of the node, where the root is at level 0.

Height: The longest path from the root node to a leaf.

Heap (here : Maxheap)

ExtractMax( )

Heap Condition : val(n) ≥ val(children(n))

Max is at the root !

1. Swap the root with the last leaf
2. Swap the parent that does not satisfy the heap condition with the bigger child !
3. Repeat 2 until every node satisfies the heap condition !

Heap (here : Maxheap)

Heap Condition : val(n) ≥ val(children(n))

insert( )

1. Place the node to the last free position
2. Swap the node with the parent, if it doesn't satisfy the heap condition
3. Repeat 2 until every node satisfies the heap condition !

Creating a heap means inserting one by one

Trees

Exam Tipps

- Know the tree condition , always keep in mind !
- Know how to insert, know how to delete
- Be able to illustrate an example by hand !
- Don’t mix up the trees !!!
- Is it min or max ?

Heap

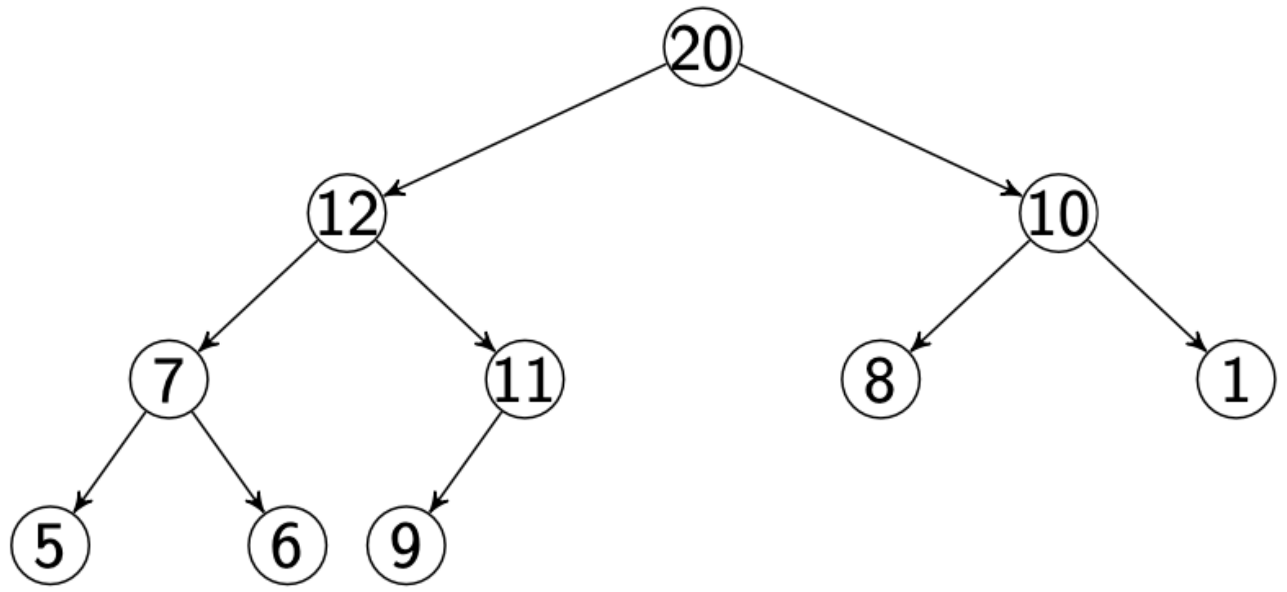
Exam Question (FS19)

/ **1 P**

a) *Min-Heap*: Draw the Min-Heap that is obtained when inserting into an empty heap the keys `8, 3, 2, 7, 4, 1` in this order.

/ **1 P**

b) *Max-Heap*: Draw the resulting Max-Heap obtained from the following Max-Heap by performing the operation `DELETE-MAX` **twice**.

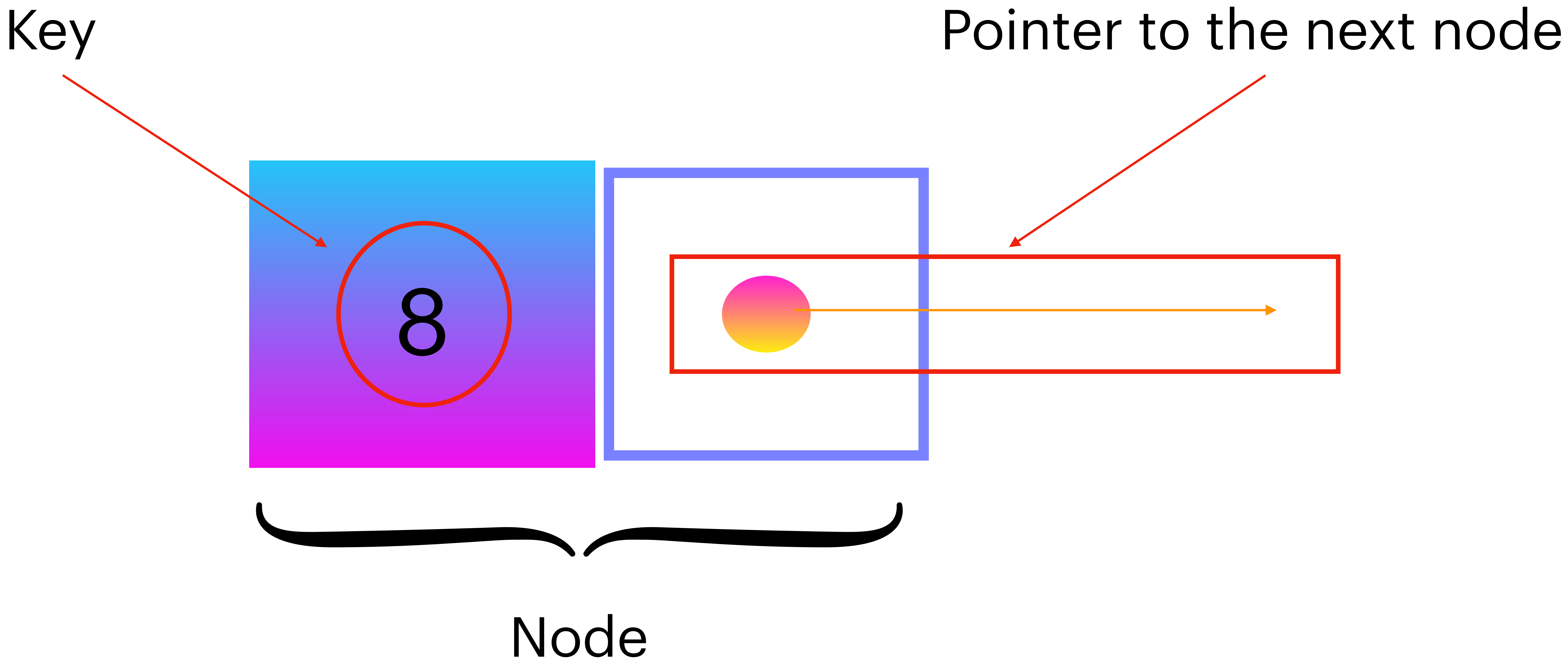


Abstract Data Types vs Data Structures

- List
- Stack
- Queue
- Priority Queue

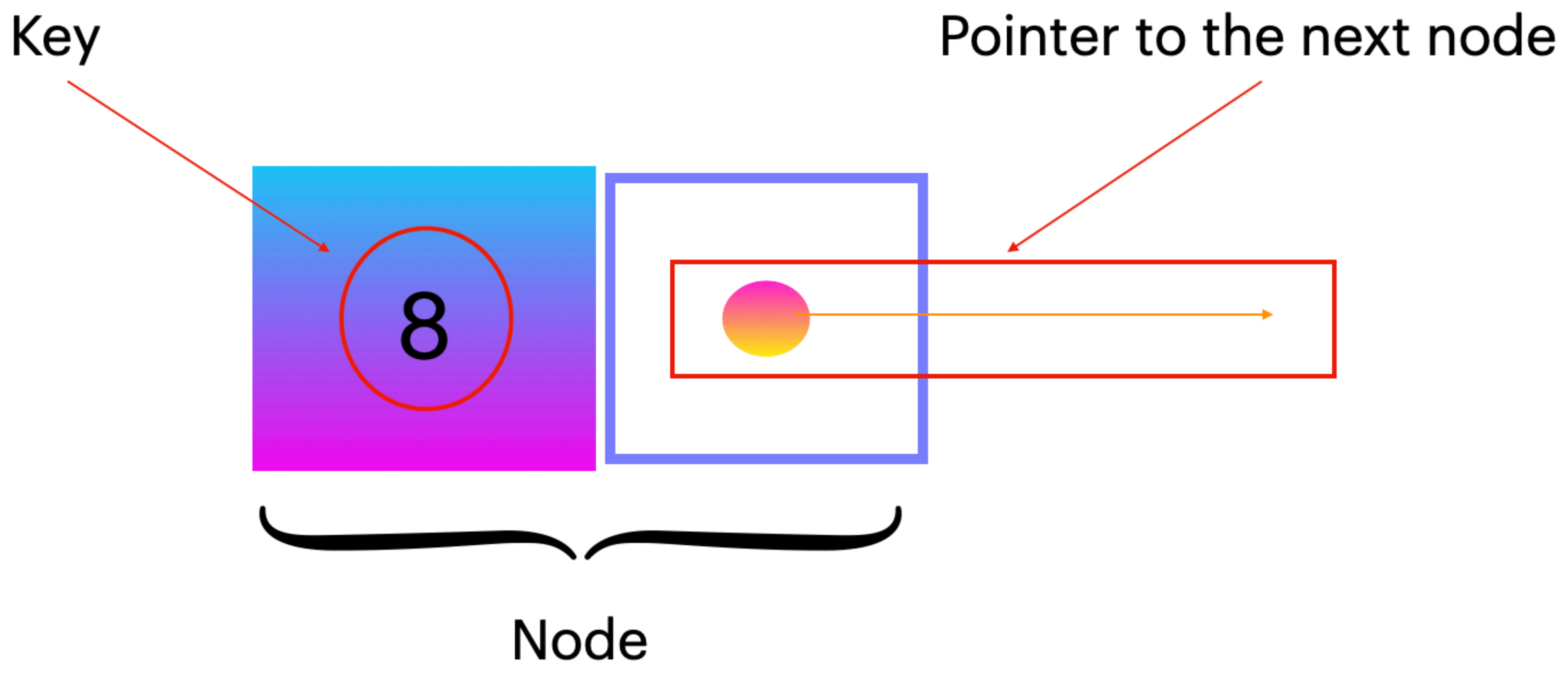
- Array
- Linked List
- Doubly Linked List
- Heaps
- ...

Linked List



Linked List

```
class LinkedList

      Node start ;


 class Node

      int key ;

      Node next ;
```



Linked List

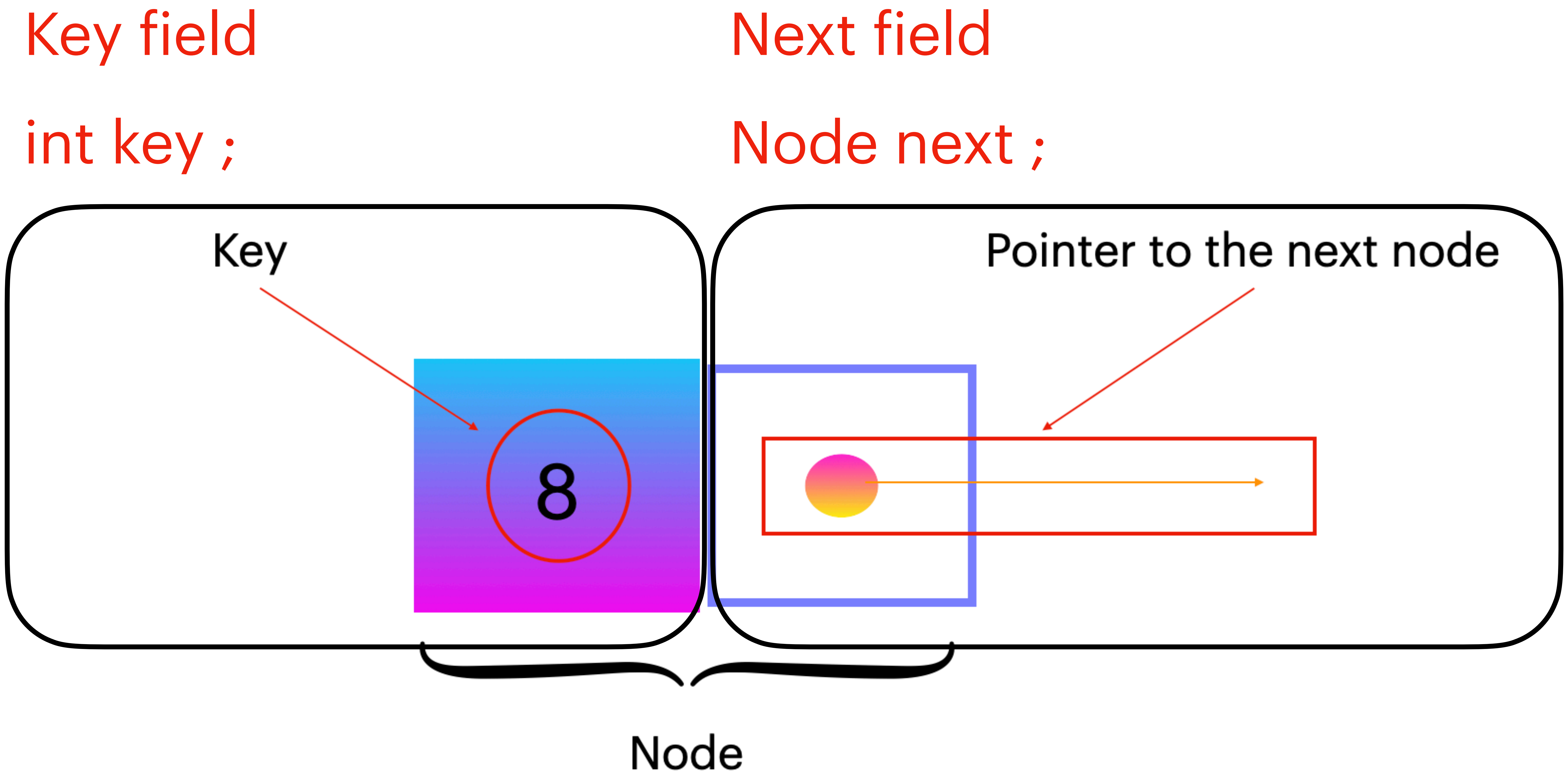
class LinkedList

    Node start ;

class Node

    int key ;

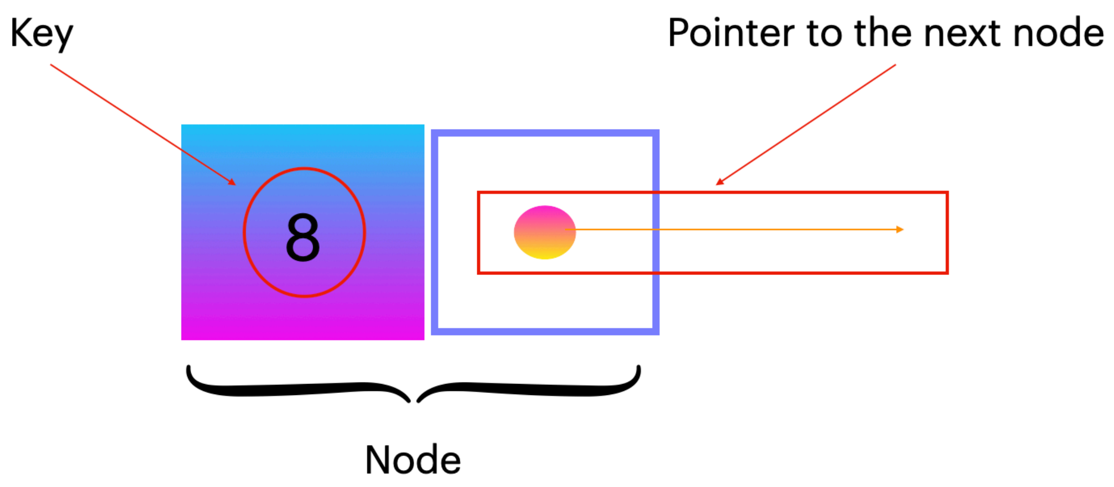
    Node next ;

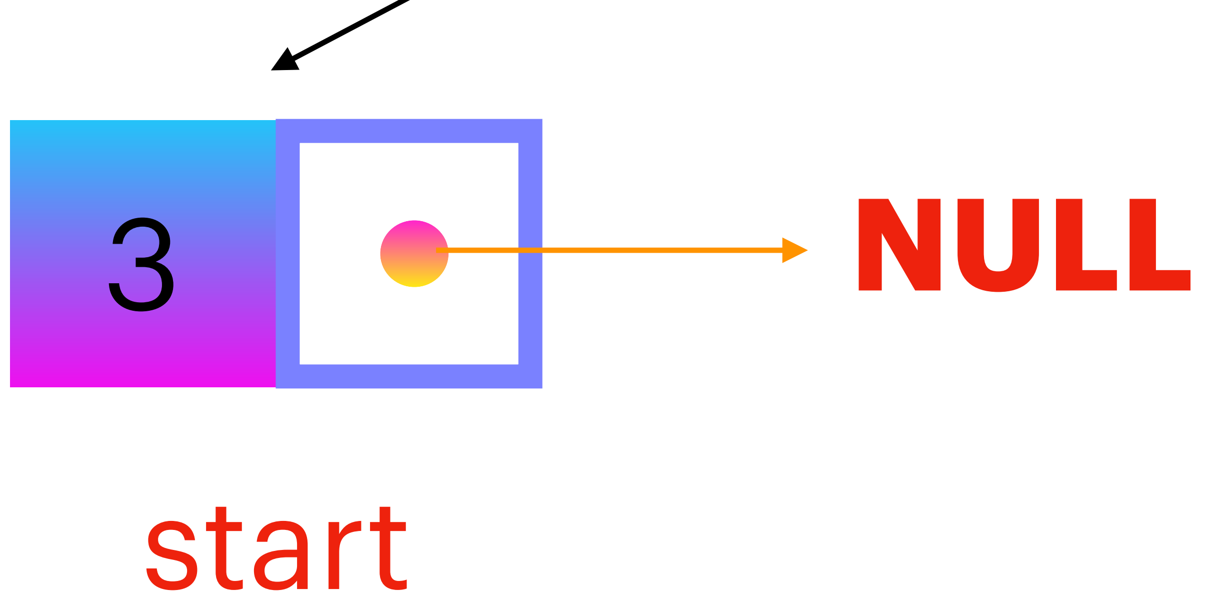


Linked List

```
class LinkedList

        Node start ;
```

```
class Node

    int key ;
    Node next ;
```

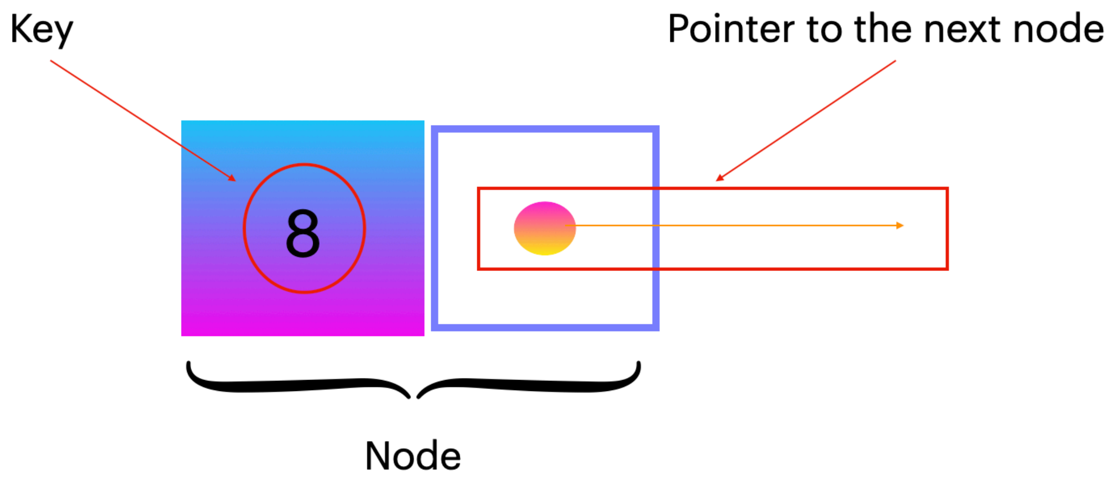


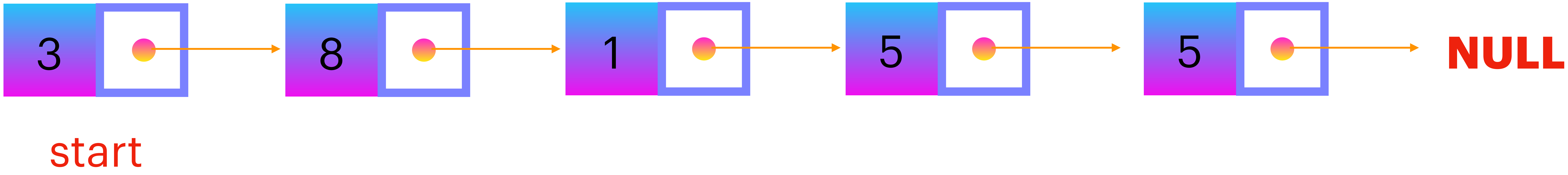


Linked List

```
class LinkedList
    Node start ;
```

```
class Node
    int key ;
    Node next ;
```





Doubly Linked List

class DoublyLinkedList

Node start ;

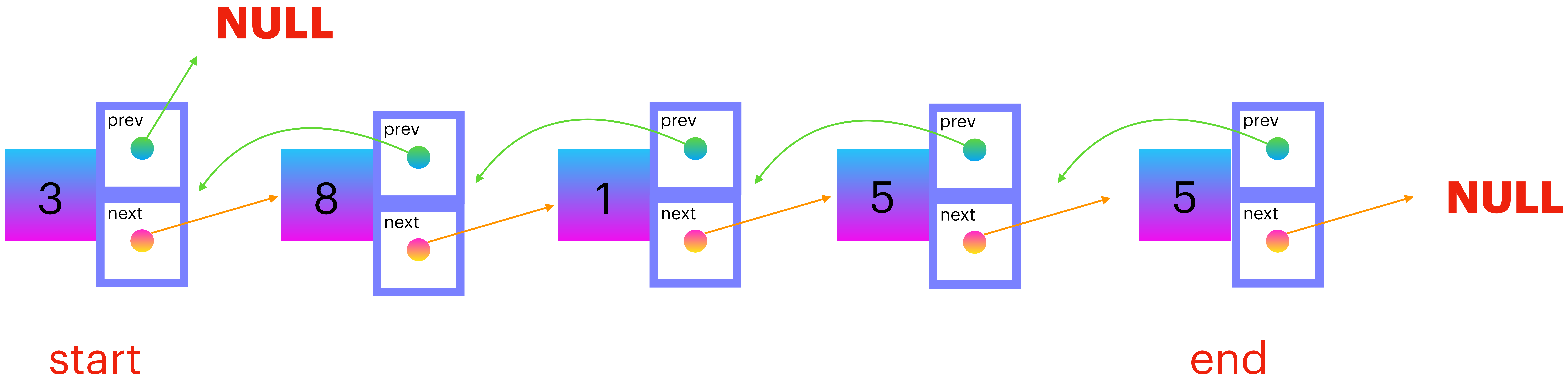
Node end ;

class Node

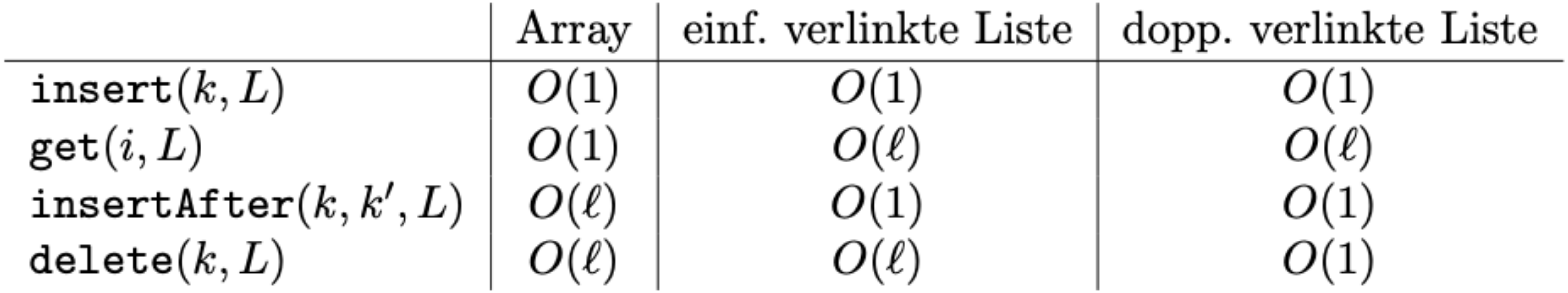
int key ;

Node next ;

Node prev ;



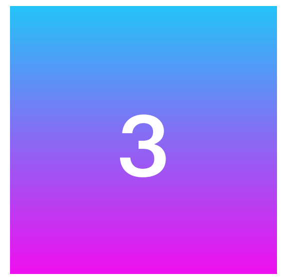
Runtimes



	Array	einf. verlinkte Liste	dopp. verlinkte Liste
`insert`(k, L)	$O(1)$	$O(1)$	$O(1)$
`get`(i, L)	$O(1)$	$O(\ell)$	$O(\ell)$
`insertAfter`(k, k', L)	$O(\ell)$	$O(1)$	$O(1)$
`delete`(k, L)	$O(\ell)$	$O(\ell)$	$O(1)$

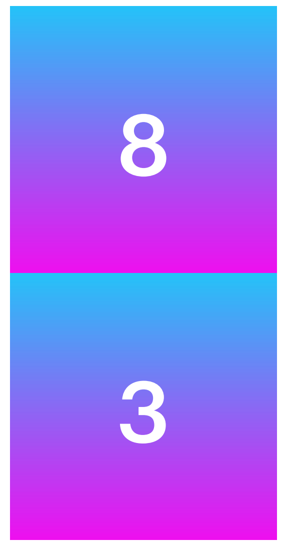
Stack

push(3)



Stack

push(8)

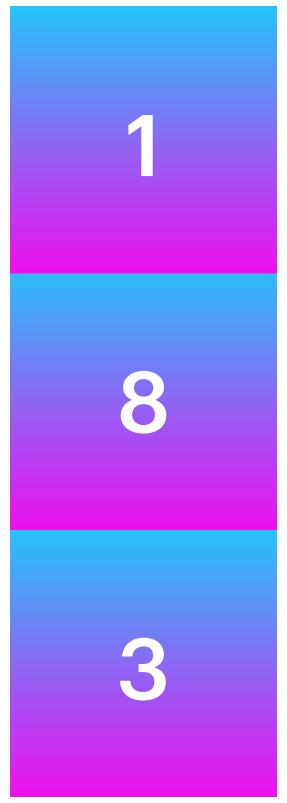


8

3

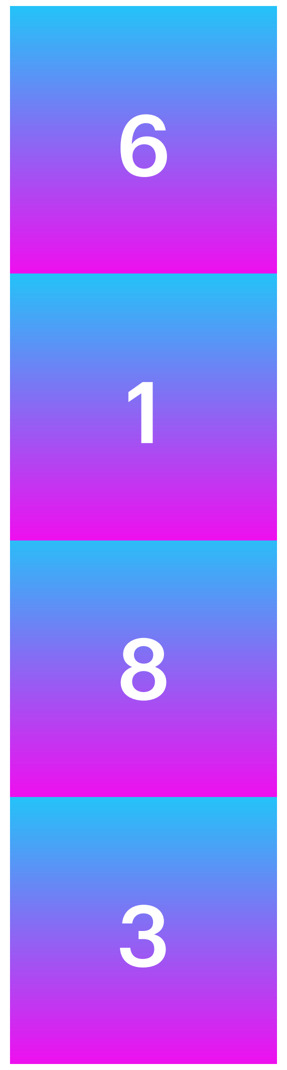
Stack

push(1)



Stack

push(6)



6

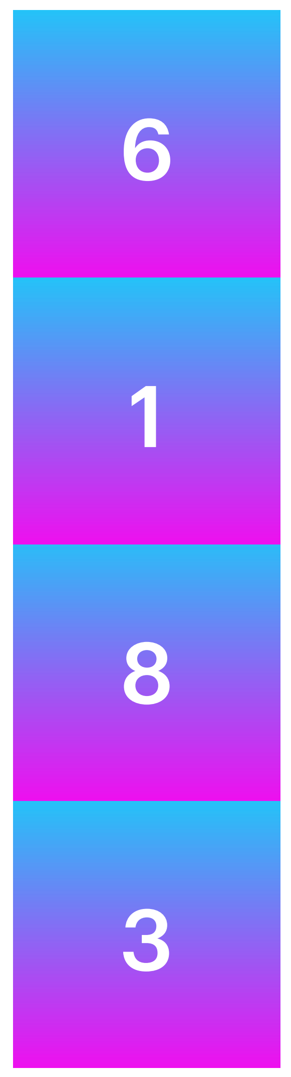
1

8

3

Stack

pop( )



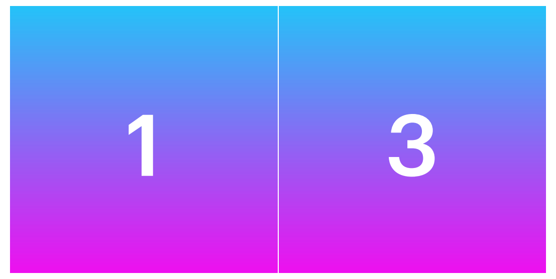
Queue

enqueue(3)

3

Queue

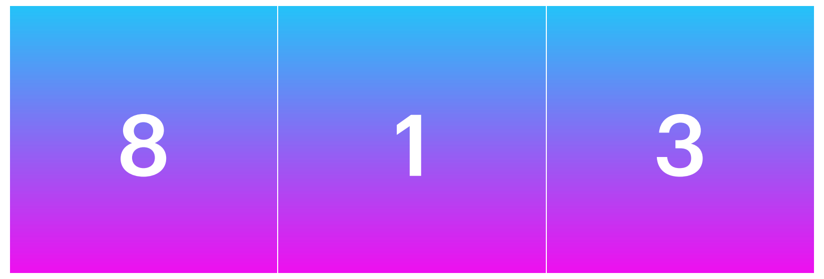
enqueue(1)



1	3

Queue

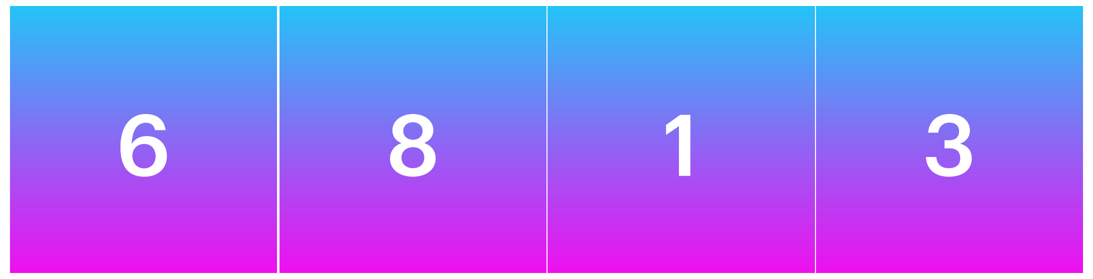
enqueue(8)



8	1	3

Queue

dequeue( )



6	8	1	3

Questions

Feedbacks , Recommendations

Nil Ozer