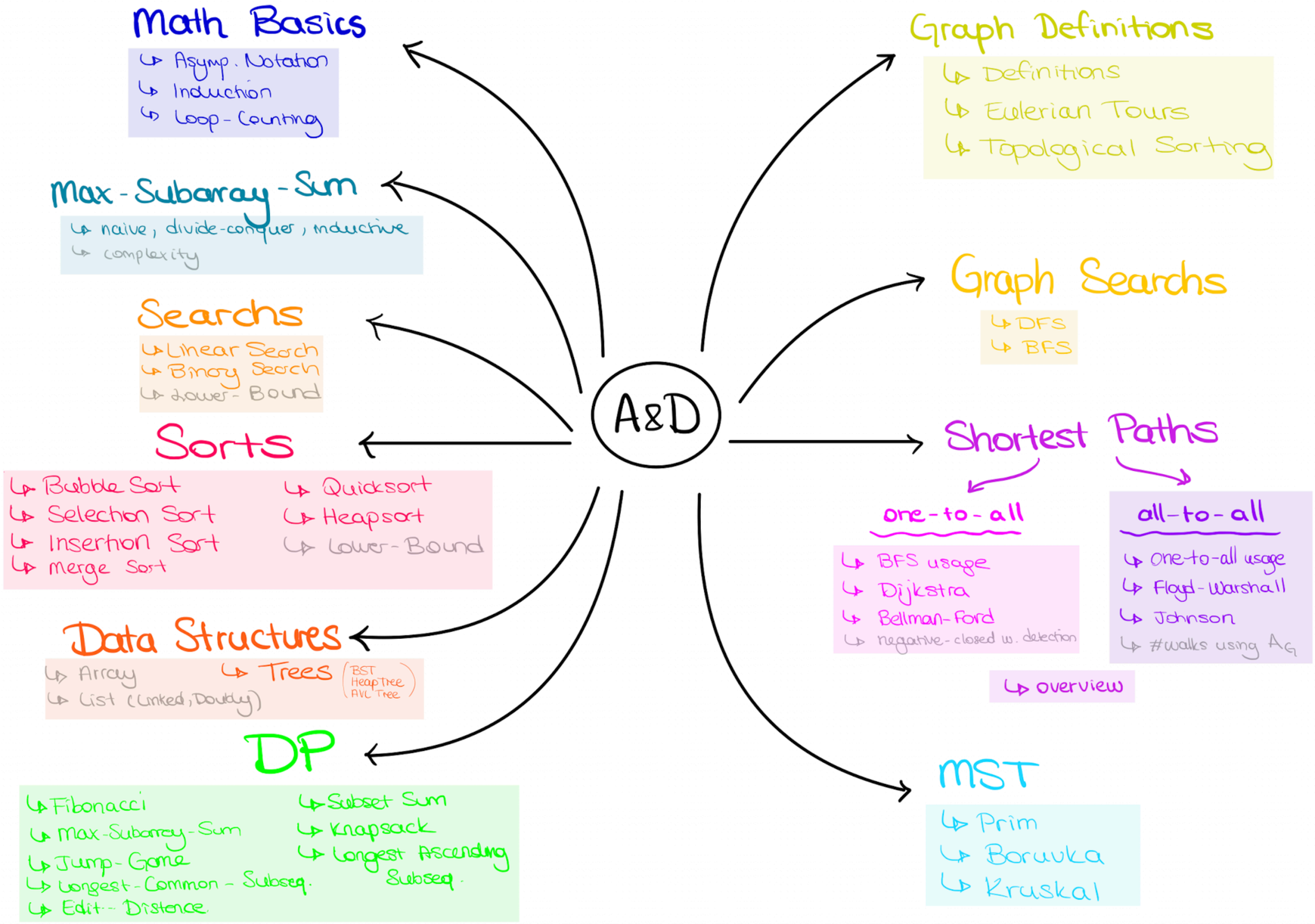
A&D

Exercise Session 4

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, HeapTree, 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
↳ Boruvka
↳ Kruskal

Outline

- Quiz

- Log Bases
- Exercise Sheet 2 Bonus Feedback
- Exercise Sheet 3 - non Bonus

- Search
- Sorts I

- Next week

Quiz

Why do the bases of log don’t matter for the asymptotic growth?

- The general idea is that different bases for the logarithm change the function only by a constant
  - The log_a(n) and log_b(n) are within constant factors of each other
  - So they don’t change the asymptotic behavior
    - If they’re not used as an exponent or so (But there ^3 and ^4 are also different)
- Why exactly ?
  - Changing the base

$$\log_b(x) = \frac{\log_a(x)}{\log_a(b)}$$

Exercise Sheet 2

Bonus Feedback

- Follow the task description.
  - If it says use theorem 1, use theorem 1!

- Pay attention to the little notes.
  - Exam grading won’t be like this

- Keep up the good work !

Exercise Sheet 3

Non Bonus

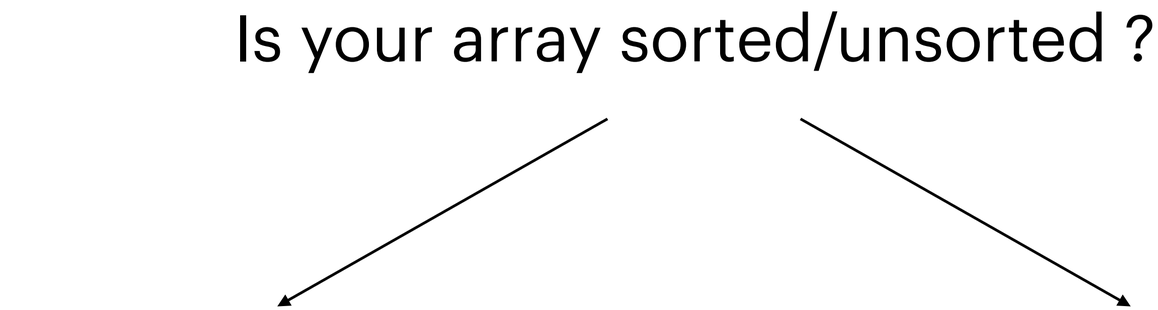
- First experience with algorithm construction (no points)
  - Pseudocode
  - Correctness
  - Runtime
  - Making it “besser”
- Next time , you’re graded ! (Exercise Sheet 4)

- Hope you solved it ! If not, you can still send it via email for this week !
  - Look at the notes ! Ask questions !

Searchs/Sorts I

Searchs

Searchs



Linear Search

Binary Search

Linear Search

Input : unsorted array , searched value

Output : the index of the search value (-1 if not found)

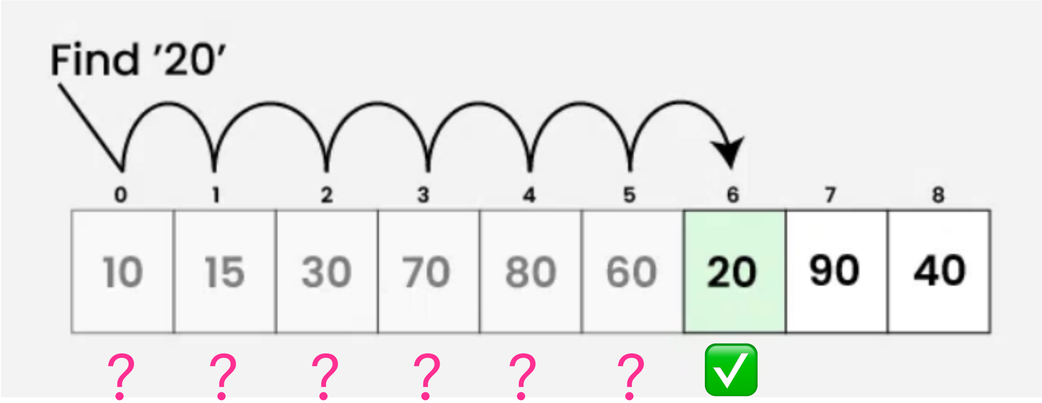
Runtime : O(n)

Pseudocode :

```
LINEAR-SEARCH(A[1..n], b)
1 for i ← 1, 2, ..., n do
2     if A[i] = b then return i
3 return "nicht gefunden"
```

Illustration

Code Example



Binary Search

Input : sorted array , searched value

Output : the index of the search value (-1 if not found)

Runtime : O(logn) $T(n) \leq T(n/2) + d \leq d \log_2(n) + c.$

Pseudocode :

BINÄRE SUCHE

BINARY-SEARCH$(A[1..n], b)$

```
1 ℓ ← 1; r ← n                          ▷ Initialer Suchbereich
2 while ℓ ≤ r do
3     m ← ⌊(ℓ + r)/2⌋
4     if A[m] = b then return m          ▷ Element gefunden
5     else if A[m] > b then r ← m − 1    ▷ Suche links weiter
6     else ℓ ← m + 1                     ▷ Suche rechts weiter
7 return "Nicht vorhanden"
```

Illustration

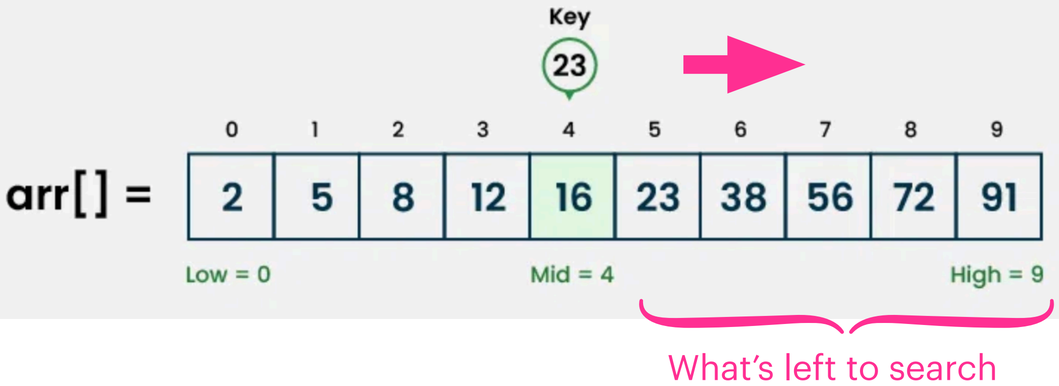
Code Example

Binary Search

Illustration

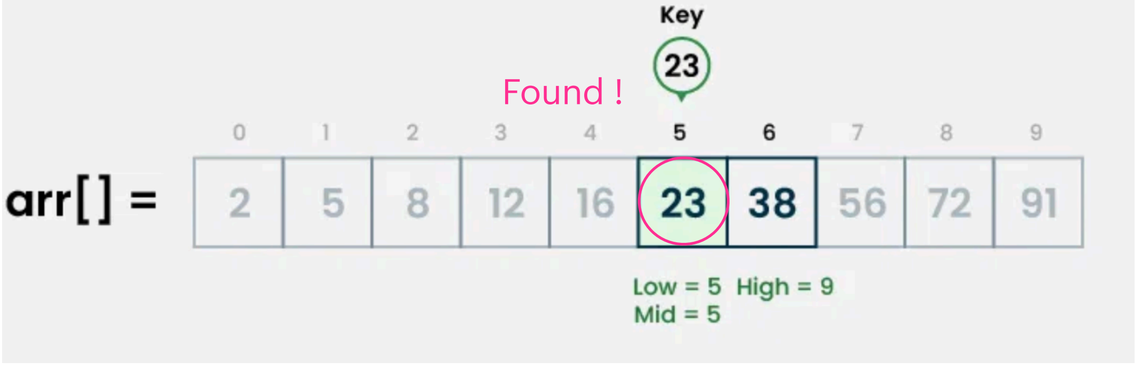
Initially | Find Key = 23 using Binary Search

Step 1 | Key (i.e., 23) is greater than current mid element (i.e., 16). The search space moves to the right.



Step 2 | Key is less than the current mid 56. The search space moves to the left.

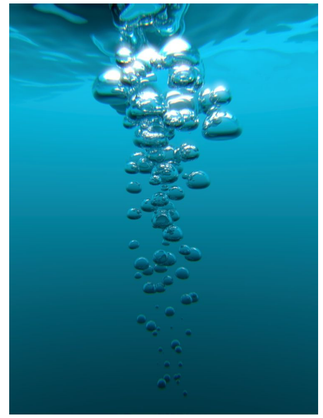
Step 3 | If the key matches the value of the mid element, the element is found and stop search.



Sorts I

Bubble Sort

Idea : bubbles going up to the surface



Input : unsorted array

Output : sorted array

Runtime: $O(n^2)$

#Comparisons: $O(n^2)$

#Swaps: $O(n^2)$

Pseudocode :

Algorithm 1 Bubble Sort (input: array $A[1 \dots n]$).

```
for j = 1, ..., n do
    for i = 1, ..., n − 1 do
        if A[i] > A[i + 1] then
            Swap A[i] and A[i + 1]
```

Illustration

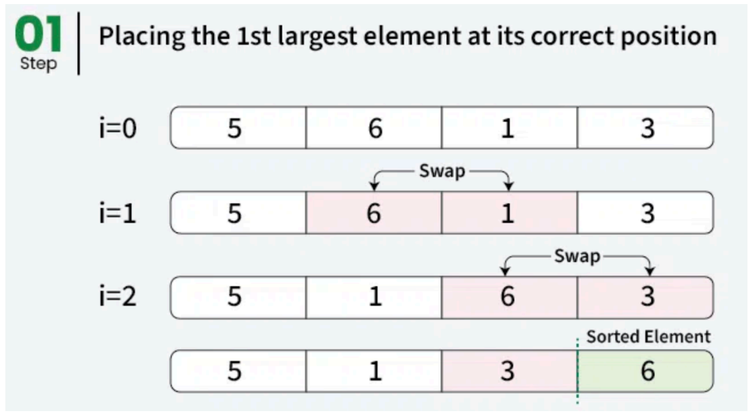
Code Example

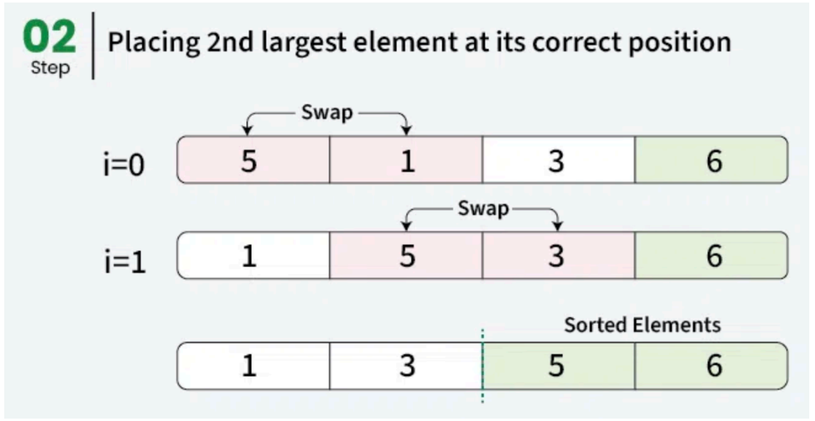
Bubble Sort

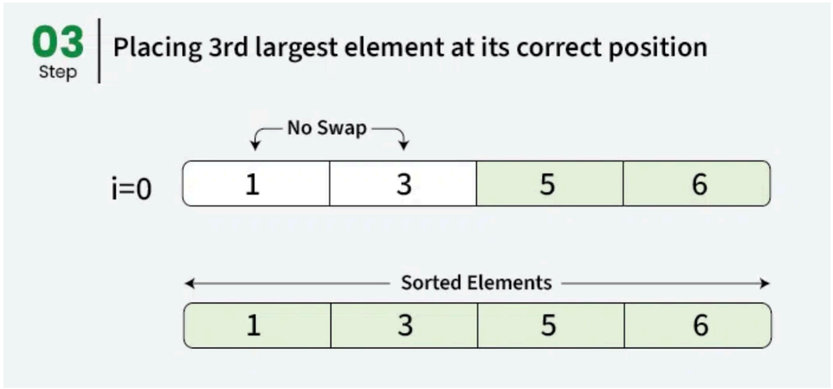
Illustration

Algorithm 1 Bubble Sort (input: array $A[1 \ldots n]$).

```
for j = 1, ..., n do
    for i = 1, ..., n − 1 do
        if A[i] > A[i + 1] then
            Swap A[i] and A[i + 1]
```







Sorted part

Selection Sort

 Idea : select the largest, put to the pos

Input : unsorted array

Output : sorted array

Runtime: $O(n^2)$

#Comparisons: $O(n^2)$

#Swaps: $O(n)$

Pseudocode :

SELECTION-SORT($A[1..n]$)

```
1 for j ← n, n − 1, . . . , 1 do
2     k ← Index des Maximums in A[1, . . . , j]
3     tausche A[k] und A[j]
```

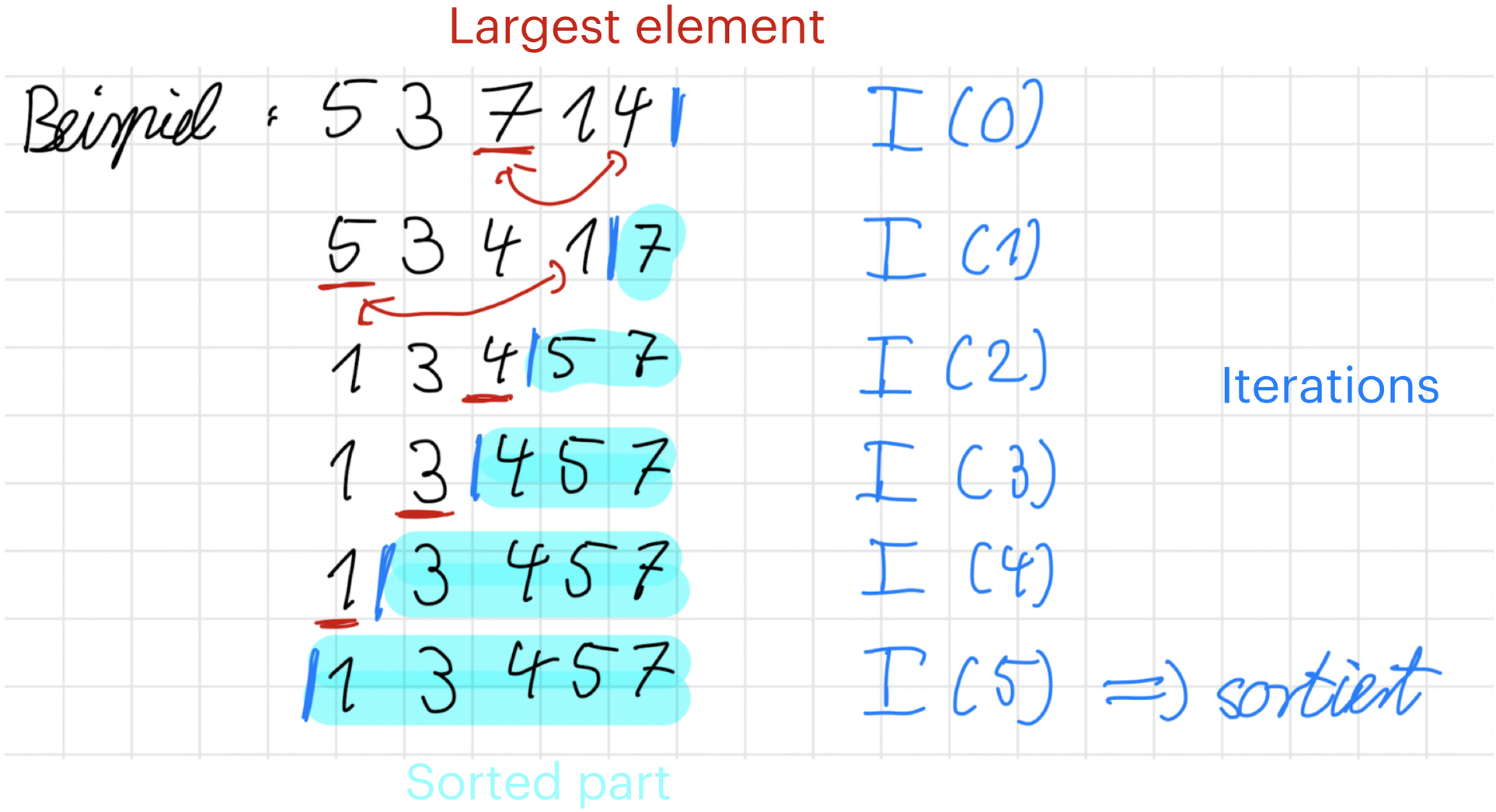
Illustration

Code Example

Selection Sort

Illustration

```
SELECTION-SORT(A[1..n])
1 for j ← n, n − 1, . . . , 1 do
2     k ← Index des Maximums in A[1, . . . , j]
3     tausche A[k] und A[j]
```



Insertion Sort

Idea : insert curr to the correct position , “verschiebe”, shift rest

Input : unsorted array

Output : sorted array

Runtime: $O(n^2)$

#Comparisons: O(nlogn)

#Swaps: $O(n^2)$

Pseudocode :

INSERTION-SORT($A[1..n]$)

```
1  for j ← 2, 3, ..., n do
2      k ← kleinster Index in {1, ..., j − 1} mit A[j] ≤ A[k]
                                          ▷ A[j] gehört an diese Stelle k
3      x ← A[j]                           ▷ merke A[j]
4      verschiebe A[k, ..., j − 1] nach A[k + 1, ..., j]
5      A[k] ← x
```

Illustration

Code Example

Insertion Sort

Idea : insert curr to the correct position , “verschiebe”, shift rest

Input : unsorted array

Output : sorted array

Runtime: $O(n^2)$

#Comparisons: O(nlogn)

#Swaps: $O(n^2)$

K can be found with binary search

Pseudocode :

INSERTION-SORT($A[1..n]$)

```
1  for j ← 2, 3, ..., n do
2      k ← kleinster Index in {1, ..., j − 1} mit A[j] ≤ A[k]
                                          ▷ A[j] gehört an diese Stelle k
3      x ← A[j]                           ▷ merke A[j]
4      verschiebe A[k, ..., j − 1] nach A[k + 1, ..., j]
5      A[k] ← x
```

Illustration

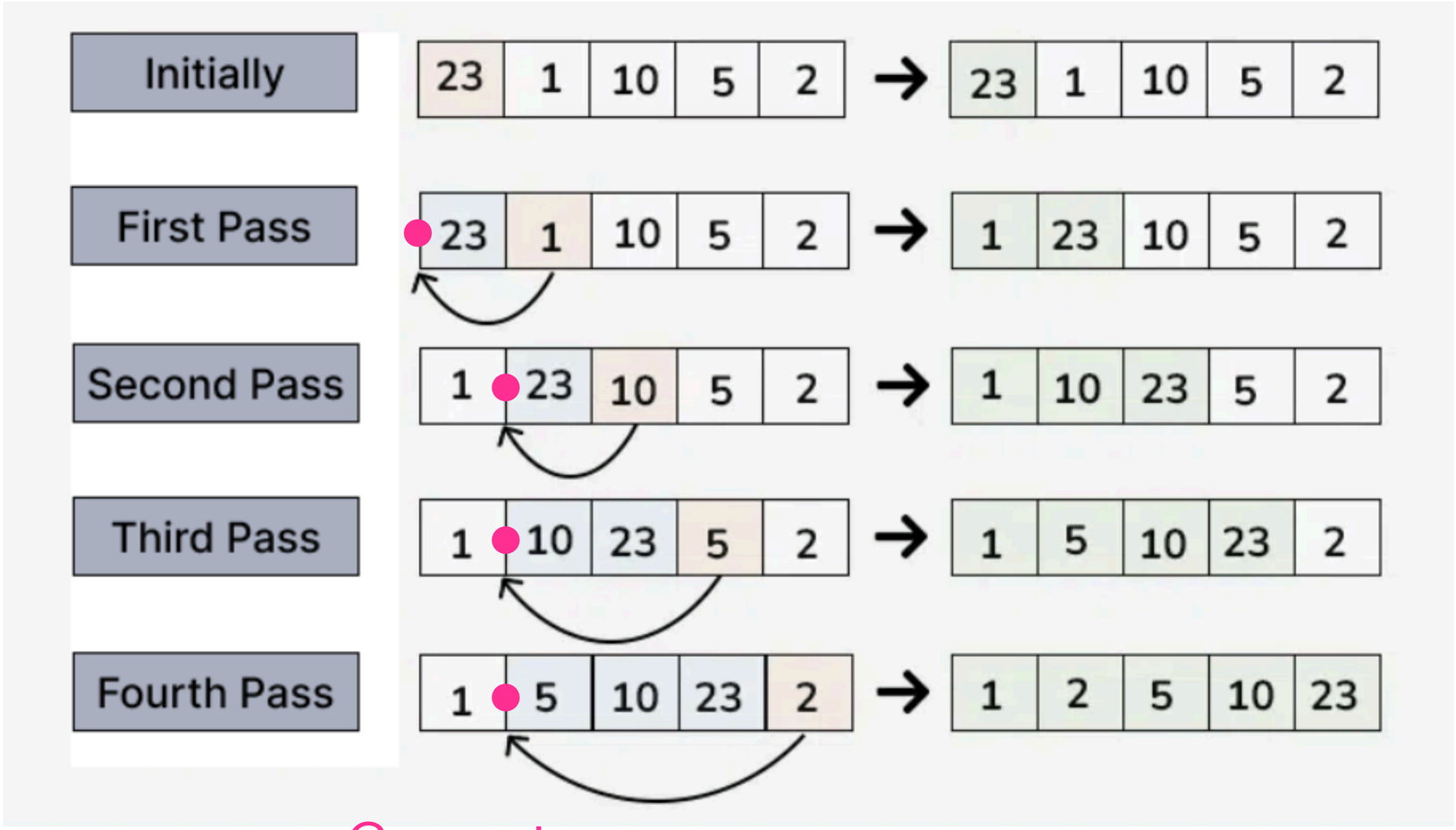
Code Example

Insertion Sort

Illustration

Algorithm 1 Insertion Sort (input: array $A[1 \ldots n]$).

```
for j = 1, ..., n do
    if j > 1 then
        for i = j − 1, ..., 1 do
            if A[i + 1] < A[i] then
                Swap A[i + 1] and A[i]
```



Merge Sort

 Idea : Divide and Conquer

Input : unsorted array

Output : sorted array

Runtime: O(nlogn)

#Comparisons: O(nlogn)

#Swaps: O(nlogn)

Pseudocode :

MERGE$(A[1..n], l, m, r)$

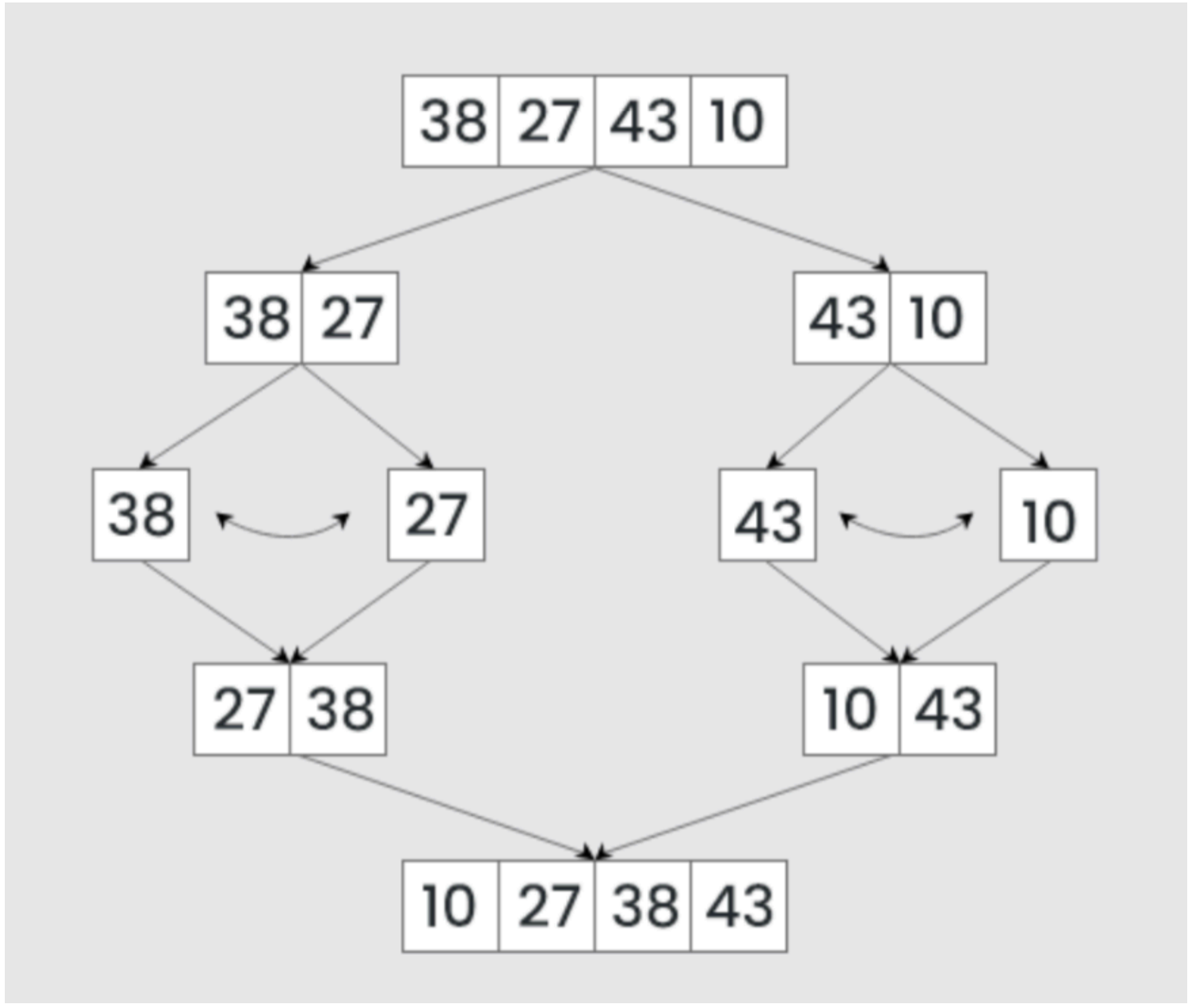
```
1  B ← new Array with r − l + 1 cells          ▷ so gross wie A[l, ..., r]
2  i ← l                                       ▷ erstes unbenutztes Element in linker Hälfte
3  j ← m + 1                                   ▷ erstes unbenutztes Element in rechter Hälfte
4  k ← 1                                       ▷ nächste Position in B
5  while i ≤ m and j ≤ r do                    ▷ beide Hälften noch nicht ausgeschöpft
6      if A[i] < A[j] then
7          B[k] ← A[i]
8          i ← i + 1
9          k ← k + 1
10     else
11         B[k] ← A[j]
12         j ← j + 1
13         k ← k + 1
14 übernimm Rest links bzw. rechts             ▷ wenn die andere Hälfte ausgeschöpft ist
15 kopiere B nach A[l, ..., r]
```

Illustration

Code Example

Merge Sort

Illustration



Costs Overview

Algorithmus	Vergleiche	Bewegungen	Extr. Platz	Lokalität
Bubble-Sort	$O(n^2)$	$O(n^2)$	$O(1)$	gut
Selection-Sort	$O(n^2)$	$O(n)$	$O(1)$	gut
Insertion-Sort	$O(n \log n)$	$O(n^2)$	$O(1)$	gut
Mergesort	$O(n \log n)$	$O(n \log n)$	$O(n)$	gut

	bubblesort		**insertion sort**		**selection sort**		**quicksort**	
	b.c.	w.c.	b.c.	w.c.	b.c.	w.c.	b.c.	w.c.
# comparisons	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
# permutations	0	$\Theta(n^2)$	0	$\Theta(n^2)$	0	$\Theta(n)$	$\Theta(n)$	$\Theta(n \log n)$
corresponding order	A	B	A	B	A	C	C	C

- A = already ordered
- B = inverse order
- C = special order

Let’s take a break

Searchs/Sorts 1

Exam Question (FS20)

/ 2 P

e) *Sorting algorithms:*

Below you see four sequences of snapshots, each obtained during the execution of one of the following five algorithms: `InsertionSort`, `SelectionSort`, ~~`QuickSort`~~, `MergeSort`, and `BubbleSort`. For each sequence, write down the corresponding algorithm.

8	6	4	2	5	1	3	7
6	4	2	5	1	3	7	8
4	2	5	1	3	6	7	8

Algorithm:

8	6	4	2	5	1	3	7
1	6	4	2	5	8	3	7
1	2	4	6	5	8	3	7

Algorithm:

8	6	4	2	5	1	3	7
6	8	2	4	1	5	3	7
2	4	6	8	1	3	5	7

Algorithm:

8	6	4	2	5	1	3	7
6	8	4	2	5	1	3	7
4	6	8	2	5	1	3	7

Algorithm:

Searchs/Sorts 1

Exam Question (FS23)

/ 4 P

g) *Sorting algorithms quiz:* For each of the following claims, state whether it is true or false. You get 1P for a correct answer, -1P for a wrong answer, 0P for a missing answer. You get at least 0 points in total.

Claim	**true**	**false**
There exist arrays of length n which can be sorted with `BubbleSort` after $\Theta(n)$ swaps.	☐	☐
There exist arrays of length n for which the runtime of `InsertionSort` is $\Theta(n)$.	☐	☐
Consider a sequence of n numbers $\{x_1, ..., x_n\}$, where $0 \leq x_i \leq 1000$, $\forall i = 1, ..., n$, is given as input. There exists an algorithm with runtime $O(n)$ which sorts any such sequence.	☐	☐
There exist a comparison-based sorting algorithm that can sort any array of length n in runtime $O(n)$.	☐	☐

Searchs/Sorts 1

Exam Question (HS20)

/ 3 P

g) *Sorting algorithms:*

i) Consider the sequence $6, 5, 4, 1, 2, 3$. How many swaps does `Bubble Sort` perform to sort this sequence? *Give the exact number of swaps required.*

ii) Consider the sequence $6, 5, 4, 1, 2, 3$. How many swaps does `Selection Sort` perform to sort this sequence? *Give the exact number of swaps required.*

iii) Let $n \in \mathbb{N}$ be an even number and consider the sequence with the following structure:

$$2, 1, 4, 3, 6, 5, \ldots, n, n-1.$$

How many swaps does `Insertion Sort` perform to sort this sequence? *Give the exact number, not just the asymptotics.*

Searchs/Sorts 1

Exam Tipps

- Whenever you have to learn a new algorithm :
  - Know the pseudocode
  - Be able to do an example by hand
  - Know the runtime ! (Here also #comparisons, #swaps)
  - (Know how to implement it)
- Types of questions in the exam
  - T/F (Sorting algorithms quiz) (most recent exams, until FS21)
  - #comparisons, #swaps
  - Counting swaps for a particular example (HS20)
  - Invariant proving (FS21 , HS20)

Formal proof of correctness

Bubble Sort

Exercise Sheet 4

Algorithm Construction expectations

```
SMALLESTINTEGER ( n , l, r)
    m ← ⌊(l+r)/2⌋ ,  f_m ← f(m)
    if  l>=r && f_m <= n  then      // bounds meet , f(T) ≥ N
        return  m                   //  return m (T)
    else if  f_m > n                // if value > N , reduce upperB
        return  SMALLESTINT (n, l, m)   // recursively call with r = m
    else if  f_m < n                // if value < N , increase lowerB
        return  SMALLESTINT (n, m+1, r)  // recursively call with l = m+1

upperB ← UPPERBOUND ( N, 1]     // upperBound calculation
SMALLESTINT (N, 0, upperB)      // call with l=0 , r= upperB

Correctness: Follows directly from explanation with comments.
             A value is only returned if bounds have
             met and desired T is found.

Runtime: T(T) = 2 · T(n/2) + 3·C   ≤ O(log n)
```

```
UPPERBOUND (N, T)        // We start by calling algo for (N, 1)
    if (f(T) ≥ N) then   // desired T found
        return T         // return T
    else
        return UPPERBOUND (N, 2T)  // T is less than N still
                                   // until it's ≥ N double T , repeat

Runtime  T(T) = T(T/2) + C  --MT a=0, b=0-->  O(T^(log_2 1) log T) = O(log T)

Correctness is proven by the descriptions of algo.
```

- Proper Pseudocode
- Correctness proof easier with comments
- Runtime

Next week ...

Prize ?

Questions

Feedbacks , Recommendations

Nil Ozer