

Datastructures and Algorithms

Containers, auto, Templates, Induction

Adel Gavranović — ETH Zürich — 2025

Overview

Learning Objectives
C++ Container Library
Templates Recap
Auto vs Templates
Repetition theory: Induction
Subarray Sum Problem
Code Example
Programming Exercise
Past Exam Questions
Tips for **code expert**



n.ethz.ch/~agavranovic

- [Material](#)
- [Webpage](#)
- [Mail](#)

1. Follow-up

Follow-up from last session

Random Access and Pointer Machine Models

- Nothing that's very relevant for the exam, so no need to worry about it too much...

Follow-up from last session

Regarding the "one week Deadline for the first"

- I was wrong! because I didn't verify the (provided) information...

Follow-up from last session

Regarding the "one week Deadline for the first"

- I was wrong! because I didn't verify the (provided) information...
- The deadline for the **code expert** Exercise 1 (Asymptotic Running Times, Prefix Sums) is

Follow-up from last session

Regarding the "one week Deadline for the first"

- I was wrong! because I didn't verify the (provided) information...
- The deadline for the **code expert** Exercise 1 (Asymptotic Running Times, Prefix Sums) is

Thu, 27.02.2025 at 23:59

which can be seen on the **code expert** page

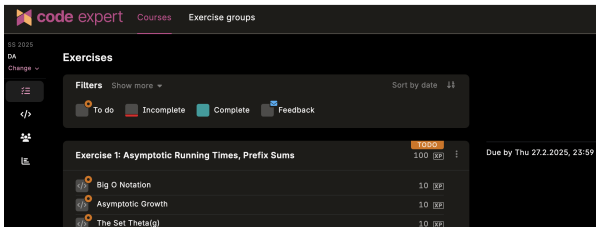
Follow-up from last session

Regarding the "one week Deadline for the first"

- I was wrong! because I didn't verify the (provided) information...
- The deadline for the **code expert** Exercise 1 (Asymptotic Running Times, Prefix Sums) is

Thu, 27.02.2025 at 23:59

which can be seen on the **code expert** page




The screenshot shows the 'code expert' interface. At the top, there are navigation links for 'Courses' and 'Exercise groups'. Below this, the page is titled 'Exercises'. There are filter buttons for 'To do', 'Incomplete', 'Complete', and 'Feedback'. A 'Sort by date' dropdown is also visible. The main content area displays 'Exercise 1: Asymptotic Running Times, Prefix Sums' with a 'TODO' status and a score of 100. To the right of this exercise, the deadline is listed as 'Due by Thu 27.2.2025, 23:59'. Below the exercise title, there is a list of sub-exercises: 'Big O Notation' (10 XP), 'Asymptotic Growth' (10 XP), and 'The Set Theta(g)' (10 XP).

2. Feedback regarding **code expert**

General things regarding **code expert**

General things regarding **code expert**

- *Please* learn some \LaTeX and markdown for the submissions
- Amazing tool for finding the right commands:  Detexify
- Also: read the descriptions carefully!

Any questions regarding **code expert** on your part?

3. Learning Objectives

Learning Objectives

- Understand what Containers are and what benefits they bring
- Understand what Templates are and what benefits they bring
- Understand how to do Induction Proofs in this course
- Be prepared to solve the next **code expert** exercises

4. Summary

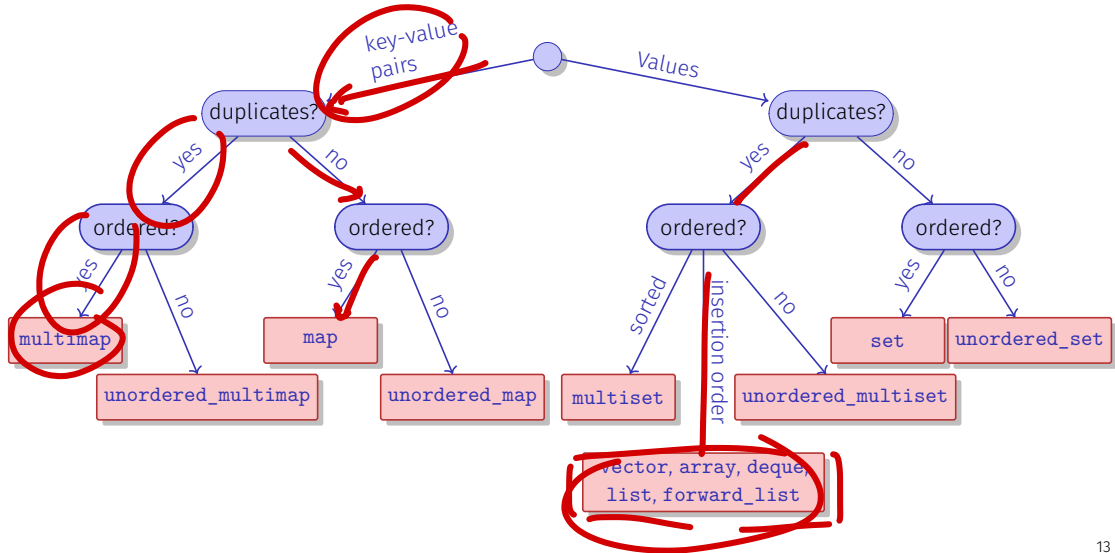
Getting on the same page

Getting on the same page

- What was covered this week and what would you like to revisit?

5. C++ Container Library

C++ Containers



Sequence-Container

<code>vector</code>	<code>array</code>	<code>deque</code>	<code>list</code>	<code>forward_list</code>
contiguous dynamic memory	contiguous static memory	Non-contig. dynamic memory	Non-contig. dynamic memory	Non-contig. dynamic memory
random access	random access	random access		
fast push/pop back		fast push/pop front/back	fast push/pop front/back	fast push/pop front
bidirectional iteration	bidirectional iteration	bidirectional iteration	bidirectional iteration	forward iteration

dynamic: size can change during runtime, **static:** size fixed at compile-time, **random access:** direct, immediate access to any element by its *index* (e.g. `vec[42]`), **bidirectional:** backward and forwards iterable

Sets and Multisets

- `std::set<E>` contains unique elements
- `std::multiset<E>` allows duplicate elements
 - Iteration yields all elements in ascending order (in non-deterministic order if `unordered_multiset`)
 - `std::multiset<E>::count(elem)` returns the number of occurrences of a given element

Example of `std::multiset`

```
Content: { Xanten Xenon Xenon Xenon Xerografie Xerophil Xylose }  
count("Xenon") = 3  
count("Xylose") = 1
```

Maps and Multimaps

- `std::map<K,V>` contains pairs (key, value), where a key maps to at most one value
- `std::multimap<K,V>` allows duplicate pairs
 - Iteration yields all pairs in ascending key order (in non-deterministic order, if `unordered_multimap`)
 - `std::multimap<K,V>::count(key)` returns the number of occurrences of a given key
 - `std::multimap<K,V>::equal_range(key)` returns all values (in non-det. order) for a given key

-
?
MP.count("19-...") = 1
MP.equal_range("Miller") = { Jan 1

Maps and Multimaps

- `std::map<K,V>` contains pairs (key, value), where a key maps to at most one value
- `std::multimap<K,V>` allows duplicate pairs
 - Iteration yields all pairs in ascending key order (in non-deterministic order, if `unordered_multimap`)
 - `std::multimap<K,V>::count(key)` returns the number of occurrences of a given key
 - `std::multimap<K,V>::equal_range(key)` returns all values (in non-det. order) for a given key

Example of `std::multimap<K,V>`

```
Content: {2, er} {2, du} {2, es} {3, Axt} {3, sie} {4, Igel}
```

```
count(2) = 3
```

```
Values for key 2: er du es
```


6. Templates Recap

Motivational Example

From the
Lecture

Example goal: generic class and functionality for matrices (and vectors), without duplicating code!

```
class Matrix { ... };  
auto m1 = Matrix<int>(5,3);  
auto m2 = Vector<std::string> {"Zurich", "Locarno"};  
auto m3 = Matrix<Complex>(...);  
  
m1(1,2) = 10;  
auto sum = m1 + m2 + m3;  
std::cout << m3.max();
```

Types as template parameters

1. In the concrete implementation of a class replace the type that should become generic (in our example: `int`) by a representative element, e.g. `T`.
2. Put in front of the class the construct `template<typename T>` (Replace `T` by the representative name).

The construct `template<typename T>` can be understood as **“for all types T”**.

Integer Matrix

```
class Matrix {  
    unsigned sizeR, sizeC;  
    std::vector<int> data;  
public:  
    Matrix(unsigned R, unsigned C): sizeR(R), sizeC(C), data(R*C) {}  
  
    int& operator() (unsigned r, unsigned c){  
        assert ( r < sizeR && c < sizeC);  
        return data[r*sizeC + c];  
    }  
  
    const int& operator() (unsigned r, unsigned c) const { .. }  
    unsigned rows() const { return sizeR; }  
  
    ...  
};
```

$m(0,0) = 1$

Generic Matrix

```
template <typename T>
class Matrix {
    unsigned sizeR, sizeC;
    std::vector<T> data;
public:
    Matrix(unsigned R, unsigned C): sizeR(R), sizeC(C), data(R*C) {}

    T& operator() (unsigned r, unsigned c){
        assert ((r < sizeR)&&(c < sizeC))
        return data[r*sizeC + c];
    }
    const T& operator() (unsigned r, unsigned c) const { .. }
    unsigned rows() const { return sizeR; }
    ...
};
```



Algorithms and functions can also be parameterised with a type:

Function Templates

1. To make a concrete implementation generic, replace the specific type (e.g. `int`) with a name, e.g. `T`,
2. Put in front of the function the construct `template<typename T>`
(Replace `T` by the chosen name)

Examples

■ For free functions

```
template <typename T>
void swap(T& x, T& y) {
    T temp = x;
    x = y;
    y = temp;
}
```

```
template <typename Iter>
void is_sorted(Iter begin, Iter end){
    ... {
        begin++      end--;
    }
```

■ For operators

```
template <typename T>
ostream& operator<<(ostream& out, const Node<T> root) {
    ...
}
```

Semantics (Code-Generation)

For each template instance, the compiler creates a corresponding instantiated class (or function) → static code generation

```
Matrix<int> m1 = ...;  
Matrix<std::string> m2 = ...;  
Matrix<Student> m3 = ...;
```

m1

m2

m3

```
class Matrix_int {  
    ...  
    std::vector<int> data  
    ...  
};
```

```
class Matrix_string {  
    ...  
    std::vector<std::string>  
    data  
    ...  
};
```

```
class Matrix_student {  
    ...  
    std::vector<Student> data  
    ...  
};
```


Semantics (Code-Generation)

For each template instance, the compiler creates a corresponding instantiated class (or function) → static code generation

Question: what does this imply for separate compilation?

- Should templates be split up into .h (declarations) and .cpp (definitions) files?
- Is it possible to ship the compiled implementation (binary file compiled from .cpp) alongside the header file?

No Separate Compilation

- Code is usually separated into `.h` (header) and `.cpp` (source) files for modularity and faster re-compilation. The header contains declarations, while the source file holds definitions/implementations, enabling better code organization and independent compilation.
- Templates can **not** be split into `.h` and `.cpp` files because the entire definition (not just the declaration) must be visible to the compiler.
- It is not possible to ship the compiled binary from the `.cpp` file alongside the header because templates are instantiated at compile time for the specific types used. The compiler doesn't know the types in advance and cannot pre-compile all in a binary file.

Integer Matrix

```
class Int_Matrix {  
    ...  
    int& operator() (unsigned r, unsigned c);  
};
```

matrix.h

```
int& Int_Matrix::operator() (unsigned r, unsigned c){  
    return data[r*sizeC + c];  
}
```

matrix.cpp

```
#include <matrix.h>  
...  
Int_Matrix m(10,10);  
m(3,3) = 5; // ok
```

main.cpp

Generic Matrix?

```
template <typename T>
class Matrix {
    ...
    T& operator() (unsigned r, unsigned c);
};
```

matrix.h

```
template <typename T>
T& Matrix<T>::operator() (unsigned r, unsigned c){
    return data[r*sizeC + c];
}
```

matrix.cpp

```
#include <matrix.h>
...
Matrix<int> m(10,10);
m(3,3) = 5; // error: undefined reference
```

main.cpp

Generalizing Code using Templates

```
class Vector {  
public:  
    Vector() {...}  
    float& operator [] (int i) { return data[i]; }  
private:  
    float data[3];  
};  
  
float scalar_product(Vector a, Vector b) {  
    float result = 0;  
    for (int i=0; i<3; ++i)  
        result += a[i] * b[i];  
    return result;  
}
```

Type testing



- Templates: syntactic checks
- Instances: checks as usual

```
template <typename T>
T abs(T v) {
    return 0 <= v ? v : -v;
}
// main
abs(8); // OK
```

```
template <typename T>
void swap(T& x, T& y) {
    ...
}
// main
double a = 1.0;
double b = 7;
swap(a, b); // OK
```

Wouldn't even compile!

```
template <typename T>
T abs(T v) {
    return 0 <= v ? v : -v; // Error
}
// main
abs("hi"); // Error
```

```
template <typename T>
void swap(T& x, T& y) {
    ...
}
// main
double a = 1.0;
string b = "seven";
swap(a, b); // Error
```

Other Languages

All languages try to foster code reuse but chose different solutions.

- C++, Rust:
 - static code generation
 - no runtime overhead
 - difficult to integrate into OOP
- C#, Scala (, Java)
 - type parameters are turned into runtime values
 - well-suited for OOP
 - minor runtime overhead
- Python, JavaScript:
 - dynamic typing (duck typing)
 - no syntactic overhead
 - potentially significant runtime overhead

7. Auto vs Templates

auto

■ Placeholder type specifier

- Must be uniquely determined by direct context: initialiser code, or returns
- User could write type themselves, but leave it to the compiler

```
std::vector<int> vec = ...;  
auto it = vec.cbegin();  
// placeholder for std::vector<int>::const_iterator
```

■ Failing examples:

 `auto x;` // x has no initializer

`x = 0.0;`

```
auto first_or_else(std::vector<int> data, unsigned int or_else) {  
    if (data.size() == 0) return or_else;  
    else return data[0];  
}
```

Handwritten red code: `auto it = vec.begin();`



Templates

- Parameters are unknown until instantiated

```
template <typename N>
char sign(N v) {
    if (0 <= v) return '+';
    else return '-';
}
```

sum<int, int>

sum

```
template <typename T1, typename T2>
struct Pair {
    T1 fst;
    T2 snd;
};
```

- Instantiation may happen anywhere

```
Pair<int, double> p1 = Pair{1, 0.1};
← auto p2 = Pair<std::string, bool>{"Brazil", true};
```

Combining templates and auto

auto inside template must be determined after instantiation

```
template <typename C>
void print(C container) {
    for (auto& e : container)
        std::cout << e << ' ';
}
```

← ranged based for-loop

```
std::vector<int> numbers = {1, 2, 3};
print(numbers); // now auto can be determined
```

```
std::vector<std::string> airports = {"LAX", "LDN", "ZHR"};
print(airports); // now auto can be determined
```

Combining templates and auto

auto inside template must be determined after instantiation

```
template <typename C>
void print(C container) {
    for (auto& e : container)
        std::cout << e << ' ';
}
```

Question: Is it possible to not use auto here?

Combining templates and auto

auto inside template must be determined after instantiation

```
template <typename C>
void print(C container) {
    for (auto& e : container)
        std::cout << e << ' ';
}
```

Question: Is it possible to not use auto here?

Answer: Yes, for example by replacing auto with an additional template parameter E

From auto to templates

- Before C++20 auto function parameters are forbidden

```
void print(auto x) {...} // Compiler error
```

From auto to templates

- Before C++20 auto function parameters are forbidden

```
void print(auto x) {...} // Compiler error
```

Question: Why do you think that is?

From auto to templates

- Before C++20 auto function parameters are forbidden

```
void print(auto x) {...} // Compiler error
```

Question: Why do you think that is?

Answer: Cannot determine type from context

From auto to templates

- Before C++20 auto function parameters are forbidden

```
void print(auto x) {...} // Compiler error
```

Question: Why do you think that is?

Answer: Cannot determine type from context

- Since C++20 auto function parameters are allowed

```
void print(auto x) {...} // ok
```

Clearly, it is still not possible to determine what auto stands for.

From auto to templates

- Before C++20 auto function parameters are forbidden

```
void print(auto x) {...} // Compiler error
```

Question: Why do you think that is?

Answer: Cannot determine type from context

- Since C++20 auto function parameters are allowed

```
void print(auto x) {...} // ok
```

Clearly, it is still not possible to determine what auto stands for.

Question: What could be the meaning of auto in this case?

From auto to templates

- Before C++20 auto function parameters are forbidden

```
void print(auto x) {...} // Compiler error
```

Question: Why do you think that is?

Answer: Cannot determine type from context

- Since C++20 auto function parameters are allowed

```
void print(auto x) {...} // ok
```

Clearly, it is still not possible to determine what auto stands for.

Question: What could be the meaning of auto in this case?

Answer: It is a shorthand for a template parameter!

```
template <typename T>  
void Print(T x){ ... }
```

8. Repetition theory: Induction

Induction: what is required?

- Prove statements, for example $\sum_{i=1}^n i = \frac{n(n+1)}{2}$.


Induction: what is required?

- Prove statements, for example $\sum_{i=1}^n i = \frac{n(n+1)}{2}$.
- Base clause:
 - The given (in)equality holds for one or more base cases.
 - e.g. $\sum_{i=1}^1 i = 1 = \frac{1(1+1)}{2}$.

Induction: what is required?

- Prove statements, for example $\sum_{i=1}^n i = \frac{n(n+1)}{2}$.
- Base clause:
 - The given (in)equality holds for one or more base cases.
 - e.g. $\sum_{i=1}^1 i = 1 = \frac{1(1+1)}{2}$.
- Induction hypothesis: we assume that the statement holds for some n

Induction: what is required?

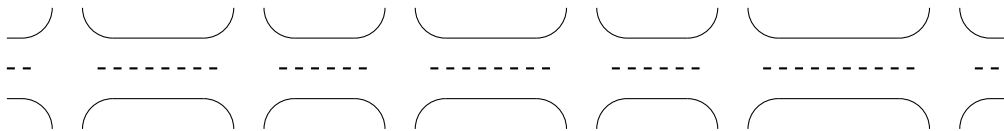
- Prove statements, for example $\sum_{i=1}^n i = \frac{n(n+1)}{2}$.
- Base clause:
 - The given (in)equality holds for one or more base cases.
 - e.g. $\sum_{i=1}^1 i = 1 = \frac{1(1+1)}{2}$.
- Induction hypothesis: we assume that the statement holds for some n
- Induction step ($n \rightarrow n + 1$):  $\leftarrow n \rightarrow \frac{n}{2}$
 - From the validity of the statement for n (induction hypothesis) it follows the one for $n + 1$.
 - e.g.: $\sum_{i=1}^{n+1} i = n + 1 + \sum_{i=1}^n i = n + 1 + \frac{n(n+1)}{2} = \frac{(n+2)(n+1)}{2}$.

9. Subarray Sum Problem

Naive Solution, prefix sums, binary search, Sliding Window

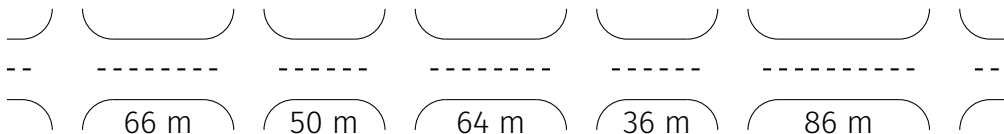
Street section of a given length

Street section of a given length



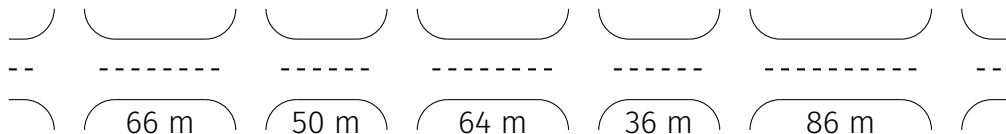
Street section of a given length

Given: distances between all crossroads on a street



Street section of a given length

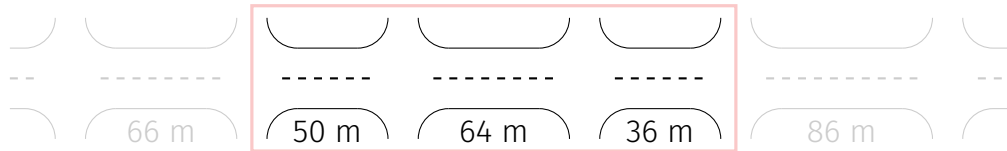
Given: distances between all crossroads on a street



Wanted: street section of length 150 meters between crossroads

Street section of a given length

Given: distances between all crossroads on a street



Wanted: street section of length 150 meters between crossroads

Subarray Sum Problem

Given: an array $A = (A[1], \dots, A[n])$ of non-negative integers

Subarray Sum Problem

Given: an array $A = (A[1], \dots, A[n])$ of non-negative integers

Wanted: a subarray with sum k :

pair (l, r) with $1 \leq l \leq r \leq n$ such that $\sum_{i=l}^{r-1} A[i] = k$

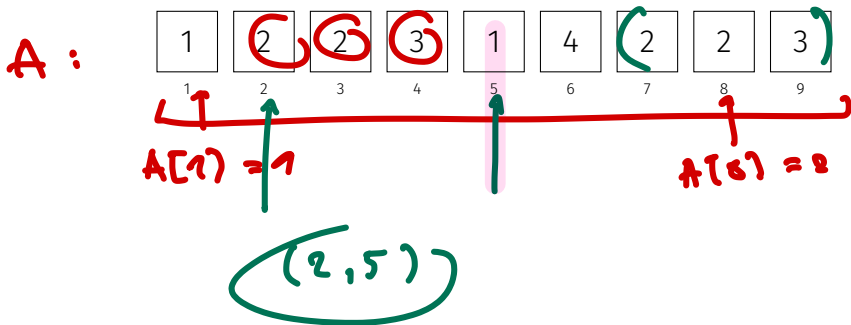
Subarray Sum Problem

Given: an array $A = (A[1], \dots, A[n])$ of non-negative integers

→ **Wanted:** a subarray with sum k :

pair (l, r) with $1 \leq l \leq r \leq n$ such that $\sum_{i=l}^{r-1} A[i] = k$

Example: $n = 9, k = 7$



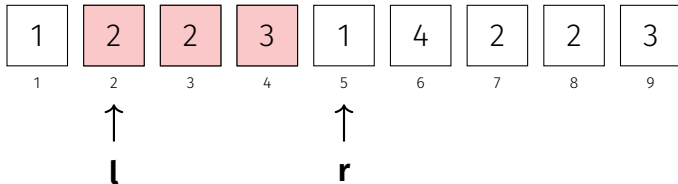
Subarray Sum Problem

Given: an array $A = (A[1], \dots, A[n])$ of non-negative integers

Wanted: a subarray with sum k :

pair (l, r) with $1 \leq l \leq r \leq n$ such that $\sum_{i=l}^{r-1} A[i] = k$

Example: $n = 9, k = 7$ **Solution:** $l = 2, r = 5$.



Strategies?

Given: an array $A = (A[1], \dots, A[n])$ of non-negative integers

Wanted: a subarray with sum k :

pair (l, r) with $1 \leq l \leq r \leq n$ such that $\sum_{i=l}^{r-1} A[i] = k$

Strategies

$\Theta(n^3)$	Three loops
$\Theta(n^2)$?
$\Theta(n \log n)$?
$\Theta(n)$?

Strategies?

Given: an array $A = (A[1], \dots, A[n])$ of non-negative integers

Wanted: a subarray with sum k :

pair (l, r) with $1 \leq l \leq r \leq n$ such that $\sum_{i=l}^{r-1} A[i] = k$

Strategies

$\Theta(n^3)$	Three loops
$\Theta(n^2)$	Prefix Sums
$\Theta(n \log n)$?
$\Theta(n)$?

Strategies?

Given: an array $A = (A[1], \dots, A[n])$ of non-negative integers

Wanted: a subarray with sum k :

pair (l, r) with $1 \leq l \leq r \leq n$ such that $\sum_{i=l}^{r-1} A[i] = k$

Strategies

$\Theta(n^3)$	Three loops
$\Theta(n^2)$	Prefix Sums
$\Theta(n \log n)$	Binary Search
$\Theta(n)$?

Strategies?

Given: an array $A = (A[1], \dots, A[n])$ of non-negative integers

Wanted: a subarray with sum k :

pair (l, r) with $1 \leq l \leq r \leq n$ such that $\sum_{i=l}^{r-1} A[i] = k$

Strategies

$\Theta(n^3)$	Three loops
$\Theta(n^2)$	Prefix Sums
$\Theta(n \log n)$	Binary Search
$\Theta(n)$	Sliding Window

Subarray Sum Problem: Sliding Window

Sliding Window Idea

Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1

Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:

Subarray Sum Problem: Sliding Window

Sliding Window Idea

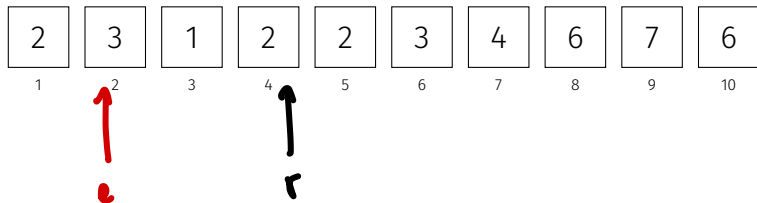
- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

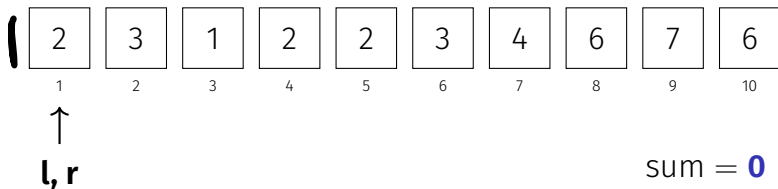


Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

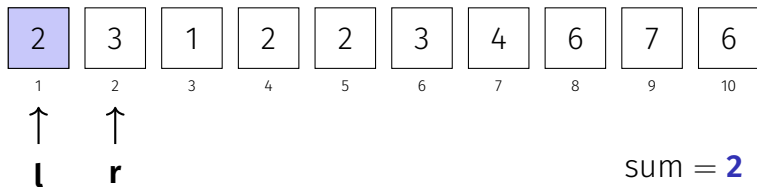


Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

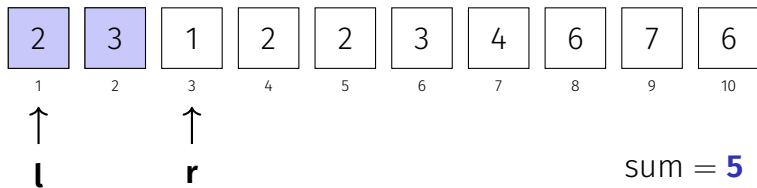


Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

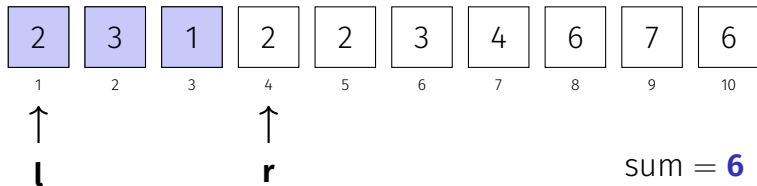


Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

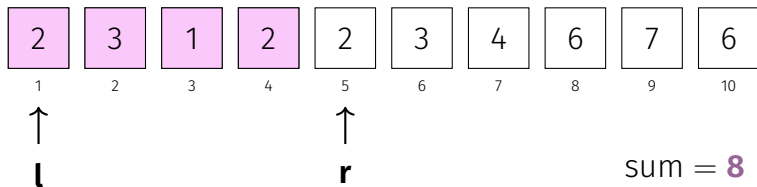


Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

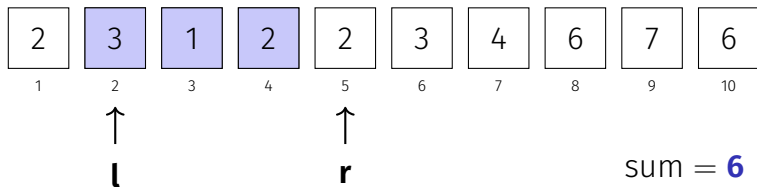


Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

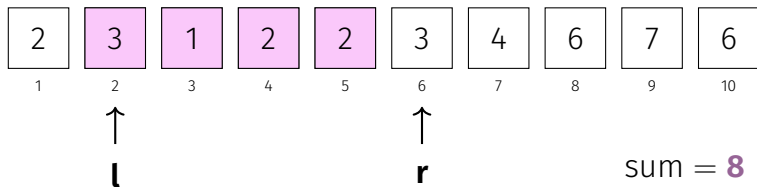


Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

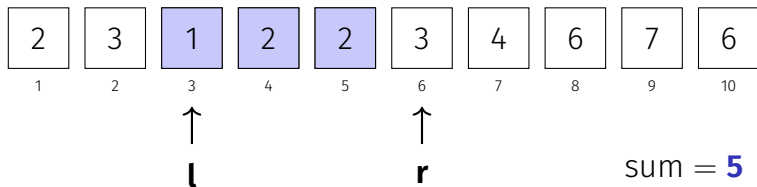


Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

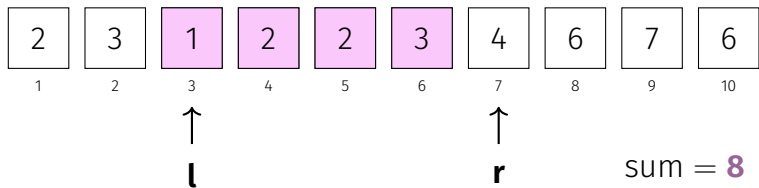


Subarray Sum Problem: Sliding Window

Sliding Window Idea

- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$

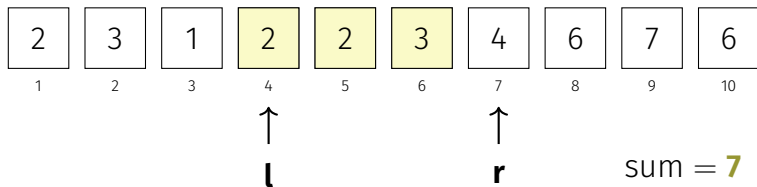


Subarray Sum Problem: Sliding Window

Sliding Window Idea

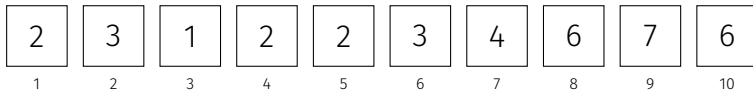
- start with left and right pointer at 1
- repeat until the end of the sequence:
 - window **too small** ($\text{sum} < k$) \Rightarrow increment right pointer
 - window **too large** ($\text{sum} > k$) \Rightarrow increment left pointer
 - window **as desired** ($\text{sum} = k$) \Rightarrow done!

Example: $k = 7$



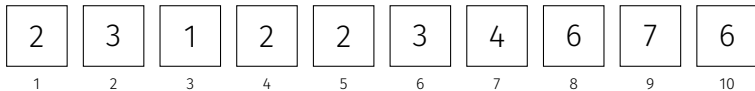
Subarray Sum Problem: Sliding Window Analysis

Subarray Sum Problem: Sliding Window Analysis



Subarray Sum Problem: Sliding Window Analysis

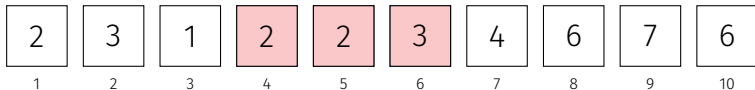
- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps



Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

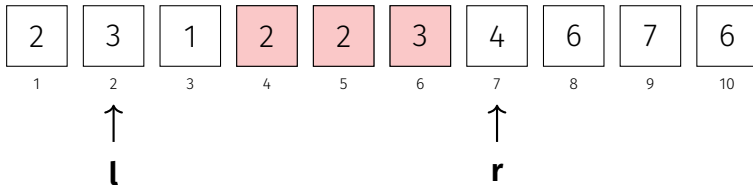


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start

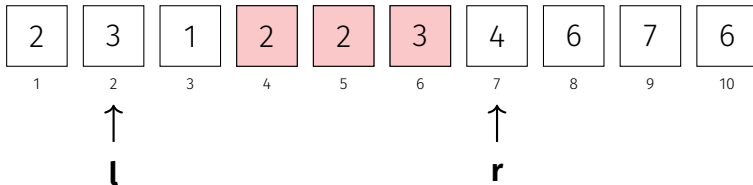


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start
⇒ sum too large

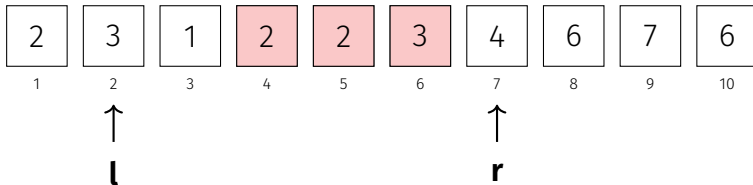


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start
⇒ sum too large ⇒ l is increased until it reaches the start of the window

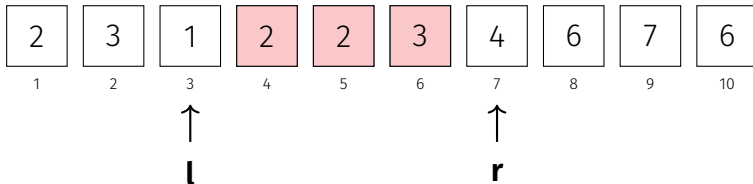


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start
⇒ sum too large ⇒ l is increased until it reaches the start of the window

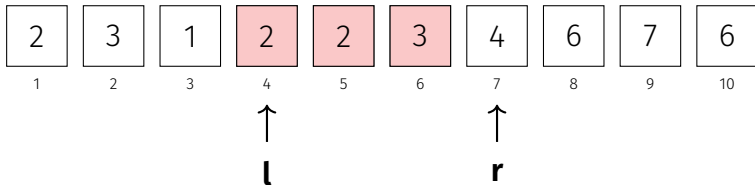


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start
⇒ sum too large ⇒ l is increased until it reaches the start of the window

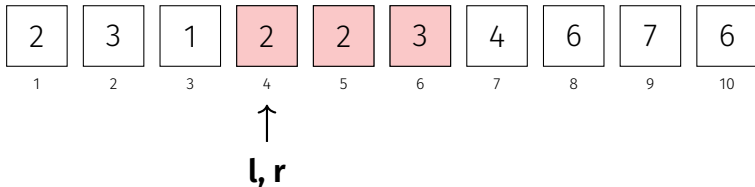


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start
⇒ sum too large ⇒ l is increased until it reaches the start of the window
- if l reaches the start before r reaches the end

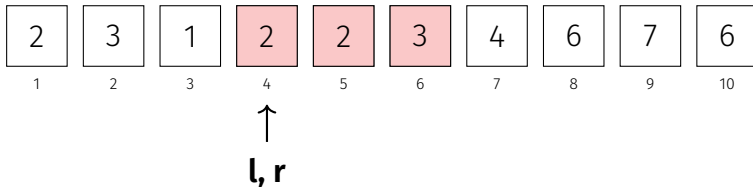


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start
⇒ sum too large ⇒ l is increased until it reaches the start of the window
- if l reaches the start before r reaches the end
⇒ sum too small

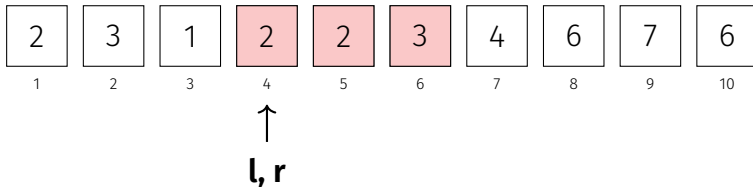


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start
⇒ sum too large ⇒ l is increased until it reaches the start of the window
- if l reaches the start before r reaches the end
⇒ sum too small ⇒ r is increased until it reaches the end of the window

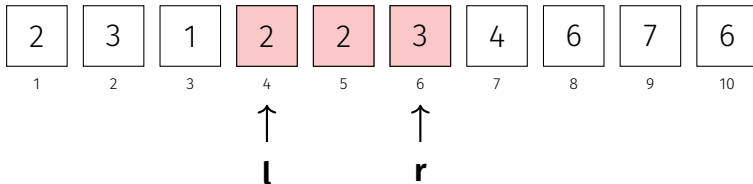


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start
⇒ sum too large ⇒ l is increased until it reaches the start of the window
- if l reaches the start before r reaches the end
⇒ sum too small ⇒ r is increased until it reaches the end of the window

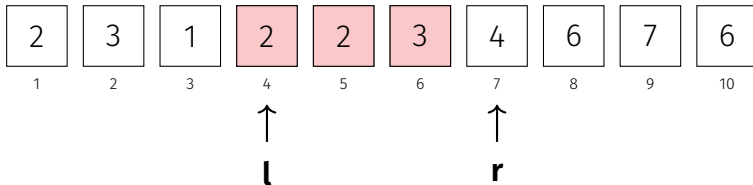


Subarray Sum Problem: Sliding Window Analysis

- in each step: either l or r is increased
⇒ algorithm terminates after a maximum of $2n$ steps

target window: lexicographically smallest (left-most) window with sum k

- if r reaches the end before l reaches the start
⇒ sum too large ⇒ l is increased until it reaches the start of the window
- if l reaches the start before r reaches the end
⇒ sum too small ⇒ r is increased until it reaches the end of the window



Analysis

We consider the lexicographically smallest (left-most) window with sum k , called *target window*

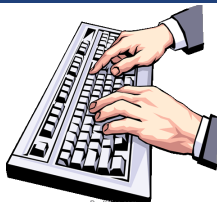
- In each step of the algorithm either l or r is increased. The algorithm terminates after a maximum of $2n$ steps.
- Assume r reaches the end of the target window before l reaches the start of the target window, then l keeps increasing until it reaches the start of the window.
- Assume l reaches the start of the target window before r reaches the end of the target window, then r keeps increasing until it reaches the end of the window.

Exercise: window with sum closest to k

10. Code Example

10. Code Example

Subarray Sum Problem → CodeExpert



11. Programming Exercise

Preparing remarks for the homework (Prefix Sum in 2D)

Sum in Subarray (naive algorithm)

Input: A sequence of n numbers (a_1, \dots, a_n) and a sub-interval $I = [l, r]$

Output: $\sum_{i=l}^r a_i$.

$S \leftarrow 0$

for $i \in \{l, \dots, r\}$ **do**

$S \leftarrow S + a_i$

return S

Sum in Subarray (naive algorithm)

Input: A sequence of n numbers (a_1, \dots, a_n) and a sub-interval $I = [l, r]$

Output: $\sum_{i=l}^r a_i$.

$S \leftarrow 0$

for $i \in \{l, \dots, r\}$ **do**

$S \leftarrow S + a_i$

return S

Idea of the exercise

- Use the prefix sum to compute the sum of arbitrary sub-intervals with constant running time
- **Generalize** to two dimensions.

12. Past Exam Questions

Aufgabe 2: Asymptotik (16P)

- (a) Geben Sie für die untenstehenden Funktionen eine Reihenfolge an, so dass folgendes gilt: Wenn eine Funktion f links von einer Funktion g steht, dann gilt $f \in \mathcal{O}(g)$.
Beispiel: die drei Funktionen n^3 , n^5 und n^7 sind bereits in der entsprechenden Reihenfolge, da $n^3 \in \mathcal{O}(n^5)$ und $n^5 \in \mathcal{O}(n^7)$.

Provide an order for the following functions such that the following holds: If a function f is left of a function g then it holds that $f \in \mathcal{O}(g)$.

Example: the functions n^3 , n^5 and n^7 are already in the respective order because $n^3 \in \mathcal{O}(n^5)$ and $n^5 \in \mathcal{O}(n^7)$.

/3P

$$\log \sqrt{n}, 2^{\log_4 n}, \sqrt{\log n}, n!, \sum_{i=1}^n 2^i, n^{1/3}, n\sqrt{n}$$

--	--	--	--	--	--	--

b

Aufgabe 2: Asymptotik (16P)

- (a) Geben Sie für die untenstehenden Funktionen eine Reihenfolge an, so dass folgendes gilt: Wenn eine Funktion f links von einer Funktion g steht, dann gilt $f \in \mathcal{O}(g)$.
Beispiel: die drei Funktionen n^3 , n^5 und n^7 sind bereits in der entsprechenden Reihenfolge, da $n^3 \in \mathcal{O}(n^5)$ und $n^5 \in \mathcal{O}(n^7)$.

Provide an order for the following functions such that the following holds: If a function f is left of a function g then it holds that $f \in \mathcal{O}(g)$.

Example: the functions n^3 , n^5 and n^7 are already in the respective order because $n^3 \in \mathcal{O}(n^5)$ and $n^5 \in \mathcal{O}(n^7)$.

/3P

$$\log \sqrt{n}, 2^{\log_4 n}, \sqrt{\log n}, n!, \sum_{i=1}^n 2^i, n^{1/3}, n\sqrt{n}$$

$\sqrt{\log n}$

$\log \sqrt{n}$

$n^{1/3}$

$2^{\log_4 n}$

$n\sqrt{n}$

$\sum_{i=1}^n 2^i$

$n!$

Altklausur 2020: Aufgabe 2b), 2c)

(c)

```
void g(int n){  
    for (int i = 0; i < n; ++i){  
        for (int j = i; j < n; ++j){  
            f();  
        }  
    }  
}
```

/1P

Anzahl Aufrufe von f / *Number of calls of f*

(d)

```
void g(int n){  
    f();  
    if (n>1){  
        f();g(n/2);  
    }  
    if (n>2) {  
        f();g(n/2);  
    }  
}
```

/3P

Anzahl Aufrufe von f / *Number of calls of f*

Altklausur 2020: Aufgabe 2b), 2c) – Solution

(c)

```
void g(int n){  
    for (int i = 0; i < n; ++i){  
        for (int j = i; j < n; ++j){  
            f();  
        }  
    }  
}
```

/1P

Anzahl Aufrufe von f / Number of calls of f

$\Theta(n^2)$

(d)

```
void g(int n){  
    f();  
    if (n>1){  
        f();g(n/2);  
    }  
    if (n>2) {  
        f();g(n/2);  
    }  
}
```

/3P

Anzahl Aufrufe von f / Number of calls of f

$\Theta(n)$

13. Tips for **code** expert

Tips for **code expert** Exercise 2

Task "Prefix Sum in 2D"

¹There's an implementation in the code examples on **code expert**

Tips for **code expert** Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D¹ well and go from there
- Make sketches!

¹There's an implementation in the code examples on **code expert**

Tips for **code expert** Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D¹ well and go from there
- Make sketches!

Task "Sliding Window"

¹There's an implementation in the code examples on **code expert**

Tips for **code expert** Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D¹ well and go from there
- Make sketches!

Task "Sliding Window"

- Sketches!

¹There's an implementation in the code examples on **code expert**

Tips for `code expert` Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D¹ well and go from there
- Make sketches!

Task "Sliding Window"

- Sketches!

Task "Proofs by Induction"

¹There's an implementation in the code examples on `code expert`

Tips for `code expert` Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D¹ well and go from there
- Make sketches!

Task "Sliding Window"

- Sketches!

Task "Proofs by Induction"

- The binomial formula will be useful for the second one
- Please format it well or just scan a PDF and upload it

¹There's an implementation in the code examples on `code expert`

Tips for `code expert` Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D¹ well and go from there
- Make sketches!

Task "Sliding Window"

- Sketches!

Task "Proofs by Induction"

- The binomial formula will be useful for the second one
- Please format it well or just scan a PDF and upload it

Task "Karatsuba Ofman"

¹There's an implementation in the code examples on `code expert`

Tips for **code expert** Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D¹ well and go from there
- Make sketches!

Task "Sliding Window"

- Sketches!

Task "Proofs by Induction"

- The binomial formula will be useful for the second one
- Please format it well or just scan a PDF and upload it

Task "Karatsuba Ofman"

- Just translate the math into code

¹There's an implementation in the code examples on **code expert**

14. Outro

General Questions?



See you next time!

Have a nice week!