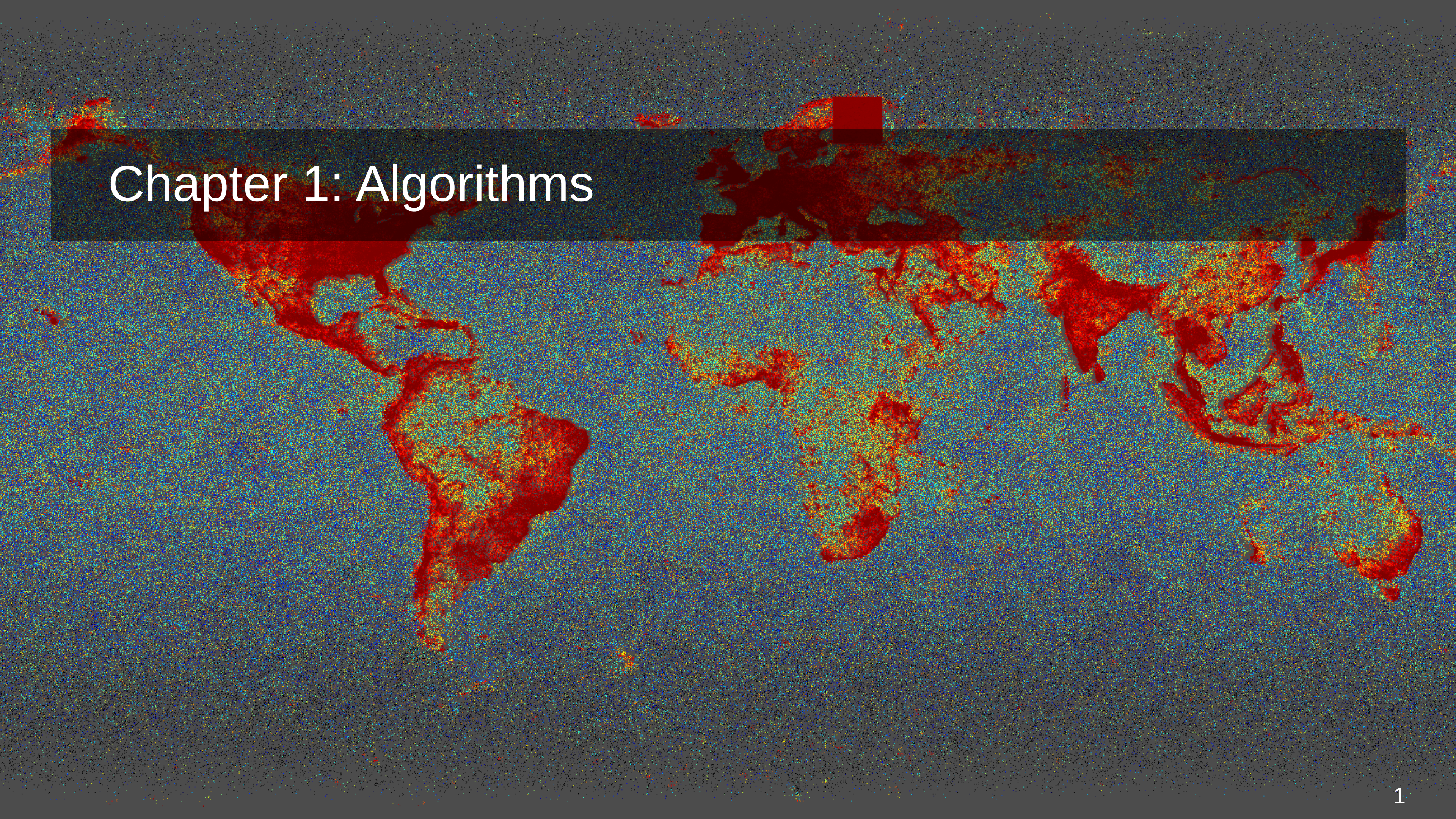
Chapter 1: Algorithms



Free-form Algorithm Introduction

Algorithm 1. *One draws a circle with the same radius r around A and B. If the radius is large enough (greater than half the distance), we are left with two intersection points of the circles which together define a line. This line is the bisector and the intersection of this respective line with the original line is the middle point p.*

Assume we have a device (or person, or computer, or . . .) that enables us to

- Draw a straight line between two points in space
- Observe intersections of pairs of lines or pairs of circles resolving them to points
- Draw a circle around any of the involved points such that the circle intersects an already existing point.

Introducing an (abstract) machine

Assume we have a device (or person, or computer, or . . .) that enables us to

- Draw a straight line between two points in space
- Observe intersections of pairs of lines or pairs of circles resolving them to points
- Draw a circle around any of the involved points such that the circle intersects an already existing point.

Implement in Terms of Machine Instruction

Algorithm 1: Given a line between two points A and B, we can draw a circle around A intersecting B and a circle around B intersecting A. These two circles will intersect in two points, say C and D. Now draw the line C D and observe the intersection with A B. This is the middle point.

Listing 1.2.1 A first program

```
Circle(A,B)
Circle(B,A)
```

Listing 1.2.2 Bisector of a Line in Euclidean Geometry

```
C1  :=  Circle(A,B)
C2  :=  Circle(B,A)
C,D :=  Intersect(C1,C2)
L1  :=  Line(A,B)
L2  :=  Line(C,D)
P  :=  Intersect(L1,L2)
```

This is called **imperative programming**

Compress Representation by Expressions

Note: now, expressions are just built from basic operators (+,*,…), precedence management (brackets) and function calls (hardware functionalities like Circle)

Listing 1.2.3 Bisector of a Line in Euclidean Geometry

```
C,D := Intersect(Circle(A,B),Circle(B,A))
P := Intersect(Line(A,B),Line(C,D))
```

Advantage: only relevant intermediate results get names (C,D)

This compression is universal

Listing 1.2.4 Bisector of a Line in Euclidean Geometry

```
P:=Intersect(Line(A,B),
        Intersect(Circle(A,B),Circle(B,A)))
```

Each sensible program or algorithm that based on some input I creates an output O, more formally each machine implementation of a function or relation

$$O = f(I)$$

Can be given as the evaluation of an expression.

- ➔ λ-calculus
- ➔ Functional Programm
- ➔ R
- ➔ Scala

This is called **functional programming**

Another way of representing this algorithm

$$M = A + \tau(B - A)$$
$$M = C + \lambda(D - C)$$
$$\|C - A\| = \|A - B\|$$
$$\|C - B\| = \|A - B\|$$
$$\|D - A\| = \|A - B\|$$
$$\|D - B\| = \|A - B\|$$

Express the problem as a set of rules (e.g., equations in this case), maybe involving variables (τ, λ). Such programs are usually executed by a classical imperative implementation solving a certain family of problems. SQL is one important example of a declarative „programming“ language

Related to:

- Integer programming, Constraint solving
- Numeric optimization
- Physical Planning (the transformation of a SQL query to an executable sequence of DB operations)

This is called **declarative programming**

Chapter 1.2.1 Written Addition

Addition of Digits – Table Lookup

Define Addition of two one-digit numbers

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Note: In order to avoid having the

Algorithm 2. *Given two n-digit numbers* $a = (a_n, \ldots, a_1)$ *and* $b = (b_n, \ldots, b_1)$*, prepare a number as follows: Starting from* $i = 1$*, set the* $i - th$ *digit (from the end) of an output variable* $c = (c_{n+1}, \ldots, c_0)$ *to the sum of* a_i *and* b_i *modulo 10. If the number* $a_i + b_i$ *is larger than 10, remember this and add an additional 1 to the next digit. If the last input digit* c_n *does not lead to a carry, remove the slot* c_{n+1}

Adding Two n-digit numbers

Listing 1.2.5 Adding two n-digit numbers

```
#include<iostream>
#include<vector>
#include<cassert>
// compile: g++ -std=c++11 -o add add.cpp

std::vector<int> add (std::vector<int> &a, std::vector<int> &b)
{
    // simplifying assumption: same length
    assert(a.size() == b.size());
    std::vector<int> c(b.size()+1);
    int carry = 0;
    for (int i=a.size()-1; i >= 0; i--)
    {
        auto digit = a[i] + b[i] + carry;
        carry = (digit >= 10)?1:0;
        auto reduced_digit = digit % 10;
        std::cout << "At␣"<<i << ",␣we␣have␣a[i]="<< a[i]
                  << ",b[i]=="<<b[i]
                  << ",digit=="<<digit
                  << ",carry=="<<carry
                  << "c[i]␣=␣reduced_digit=="<<reduced_digit
```

Adding Two n-digit numbers

```
                        << std::endl;
            c[i+1] = reduced_digit;
        }
        c[0] = carry; // can be 0 or 1
        return c;

}

in
{




    auto output = add(a,b);
    std::cout << "Result:␣";
    for (const auto digit:output)
      std::cout << digit;
    std::cout << std::endl;
}
```

Listing 1.2.6 Output of adding two numbers program.

```
At 2, we have a[i]=3,b[i]==3,digit==6,carry==0,c[i] = reduced_digit==6
At 1, we have a[i]=2,b[i]==2,digit==4,carry==0,c[i] = reduced_digit==4
At 0, we have a[i]=5,b[i]==6,digit==11,carry==1,c[i] = reduced_digit==1
Result: 1146
```

Complexity (a Naive Version of Counting Lines)

```
    assert(a.size() == b.size());
    std::vector<int> c(b.size()+1);
    int carry = 0;
    for (int i=a.size()-1; i >= 0; i--)
    {
        auto digit = a[i] + b[i] + carry;
        carry = (digit >= 10)?1:0;
        auto reduced_digit = digit % 10;
        std::cout << "At␣"<<i << ",␣we␣have␣a[i]="<< a[i]
                  << ",b[i]=="<<b[i]
                  << ",digit=="<<digit
                  << ",carry=="<<carry
                  << "c[i]␣=␣reduced_digit=="<<reduced_d
                  << std::endl;
        c[i+1] = reduced_digit;
    }
    c[0] = carry; // can be 0 or 1
    return c;

}
```

3

5

2

N=a.size()

N+1 Times
A few instructions that don‘t depend on N

Landau-Symbols (Lower bounds)

We say, this algorithm has complexity O(N)

More formally

$$f \in o(g) :\Leftrightarrow \forall_{c>0} \exists_{x_0>0} \forall_{x>x_0} |f(x)| < c|g(x)|$$

Interpretation:

A function is $o(g)$ if for large enough arguments $\mathrm{f(x)}$ is bounded by a constant times $g(x)$

Less formally

„In the end (for $x \rightarrow \infty$), f grows slower than g"

More formally

$$f \in O(g) :\Leftrightarrow \forall_{c>0} \exists_{x_0>0} \forall_{x>x_0} |f(x)| \leq c|g(x)|$$

Same, but now:

„In the end (for $x \rightarrow \infty$), g grows slower or equally fast as compared to f"

Landau-Symbols (Upper Bound)

Conversely

$$f \in \Omega(g) :\Leftrightarrow \forall_{c>0} \exists_{x_0>0} \forall_{x>x_0} c|g(x)| \leq |f(x)|$$

Less formally

„In the end (for $x \to \infty$), g grows slower than f“

Finally (for us)

$$f \in \Theta(g) :\Leftrightarrow f \in O(g) \wedge f \in \Omega(g)$$

Same, but now:

„In the end (for $x \to \infty$), g grows slower or equally fast as compared to f“

Applied:

Let f denote the number of steps of our addition program on a certain computer. Then

$$f(x) = 4 + 5\,(N + 1) = 4 + 5\mathrm{N} + 5 = 5\mathrm{N} + 9$$

On a different computer, maybe the cout is not counted as a single instruction (as in our strange machine), but itself as 42 instructions for really preparing the output.

Then

$$f(x) = 4 + (42 + 4)\,(N + 1) = 46\mathrm{N} + 50$$

In both cases (that is kind-of independent from the details of the machine) both algorithms are

$$f \in O(N)$$

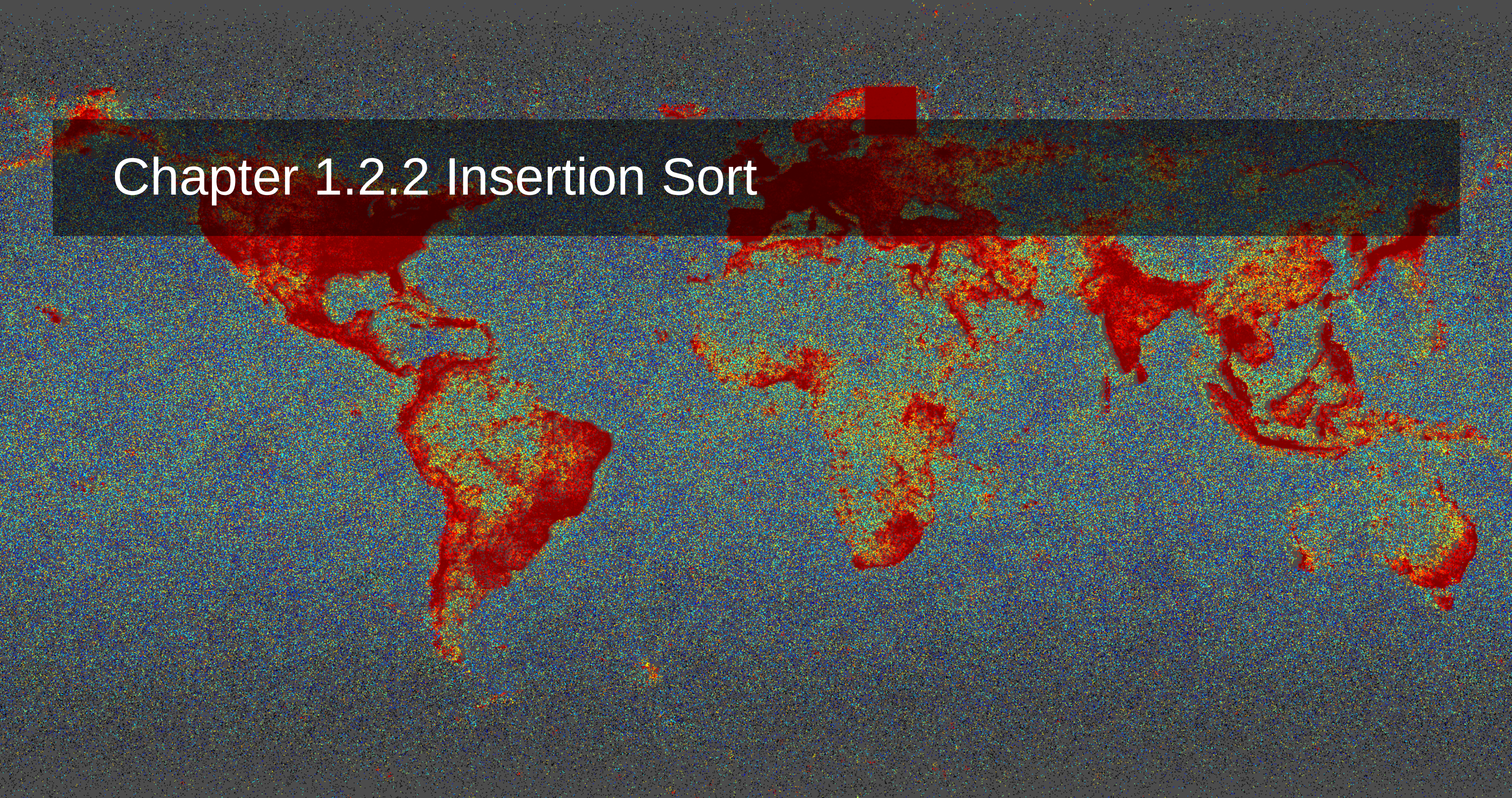
Conversely, we know that – independent of the machine – each digit must be written. That is, at least N instructions (assuming that a write is O(1)) are needed to create the output, hence,

$$f \in \Omega(N)$$

In summary, we conclude

$$f \in \Theta(N)$$

Chapter 1.2.2 Insertion Sort



Definition 1. *An algorithm $\mathcal{A}$ solves* **the search problem** *if given an array $A = (a_1, \ldots, a_n)$ it creates an array B with the same entries as A, but $b_i \leq bi + 1$ for all $i = 1 \ldots n - 1$. Such an algorithm*

Definition 2. *Such an algorithm $\mathcal{A}$ is called* **in-place** *if the input array A is used to hold the output (and typically only a small constant amount of additional memory is used).*

Insertion Sort

Listing 1.2.7 Insertion Sort

```
A = [1,2,4,3,5,2]
% Insertion Sort
for j = 2:length(A)
      in_hand = A(j);
      i = j-1;
      while (i > 0 && A(i) > in_hand)
            A(i+1) = A(i); %shift right, consider A[i] empty
            i = i -1;
      end
      A(i+1) = in_hand;
      disp(A)
end
```

Example Process Flow

A formal discussion of this algorithms follows when we look more closely into sorting.

Hand	A_0	A_1	A_2	A_3
ϵ	4	3	2	1
3	4	ϵ	2	1
3	ϵ	4	2	1
ϵ	3	4	2	1
2	3	4	ϵ	1
2	3	ϵ	4	1
2	ϵ	3	4	1
ϵ	2	3	4	1
1	2	3	4	ϵ
1	2	3	4	ϵ
1		3	ϵ	4
1	2	ϵ	3	4
1	ϵ	2	3	4
ϵ	1	2	3	4