Introduction to the Course

311116030 DSA, Sichuan University

Hao Wang

Course overview

- What is this course all about?
- What are data structures?
- What is an algorithm?
- Example: Peak Finding

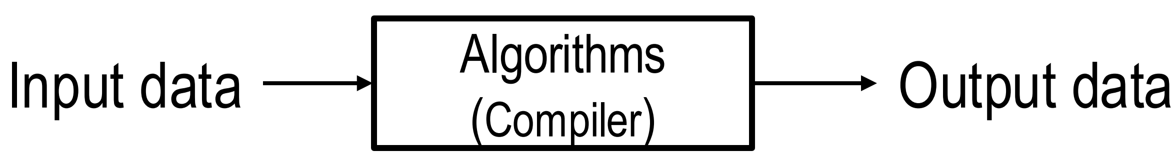
About this course

- What are programs made of?

Programs = Data Structures + Algorithms

About this course

- This course covers:
  - Data structures for efficiently storing, accessing, and modifying data
  - Algorithms for solving problems efficiently.
- In a nutshell,
  - Binary relation from **problem inputs** to **correct outputs**



About this course

Input data → Algorithms (Compiler) → Output data

- Inputs
  - Not general: small input instance
    - E.g., In this room, is there a pair of students with same birthday?
  - General: arbitrarily large inputs
    - E.g., Given any set of n, is there a pair of students with same birthday?
- Outputs
  - Usually don't specify every correct output for all inputs (too many!)
  - Provide a verifiable **predicate** (a property) that correct outputs must satisfy ('= same').

About this course

- Many approaches and technologies, how do we choose between them.
  - To design an algorithm that is easy to understand, code and debug. (No!)
  - To design an algorithm that makes ***efficient*** use of the computers. (Yes!)

- We mostly talk about the second realm in this course.

About this course

- A solution is said to be **efficient** if it solves the problem within its resource constraints.
  - Space
  - Time
- The cost of a solution is the amount of resources that the solution consumes.
- The different choices can have huge differences in running cost.
  - sequential search (~s) vs. binary search (~d).

Data Structures

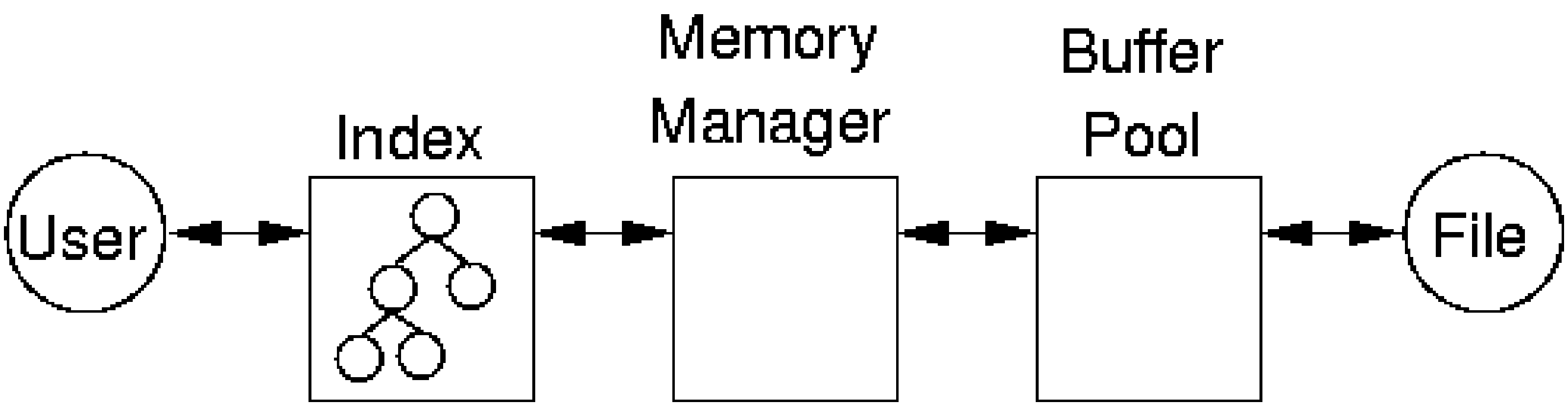
- Data structures organize data
  - to support and ground more efficient programs.
- A data structure is an implementation for an Abstract Data Type (**ADT**) .
  - An ADT is the realization of a data type, that supports a set of operations.
  - A collection of operations is called an **interface**
    - Sequence: Extrinsic order to items (first, last, nth)
    - Set: Intrinsic order to items (queries based on item keys)

Abstract Data Type

- Each ADT operation is defined by its inputs and outputs.
  - Encapsulation: hidden from the user of the ADT.
- An ADT handle complexity through the use of **abstraction**: metaphor.
  - Hierararchy of labels
  - E.g., hard_drive -> CPU -> computer.
- In a program, implement an ADT, then think only about the ADT, not its implementation.

Example 1.8: a simple database system

- A typical database-style project would have a lot of interactive and recursive parts.



A program such as this:

- too complex for human programmer to handle all at once.
- implemented through use of **<u>abstraction and metaphors</u>**.

Data Structures Philosophy

- Each data structure has <u>costs and benefits</u>.
- It is hardly ever true that one data structure is better than another for use in all situations. (No Free Lunch for both costs and benefits)
- A data structure requires:
  - space for each data item it stores,
  - time to perform each basic operation,
  - programming effort.

Data Structures Philosophy (cont'd)

- Each problem has constraints on available space and time.
- Only after a thorough analysis of problem characteristics can we determine the best data structure for the task.
- Bank **example**:
  - Start account: a few minutes
  - Transactions: a few seconds
  - Close account: overnight

Data Structures Philosophy (cont'd)

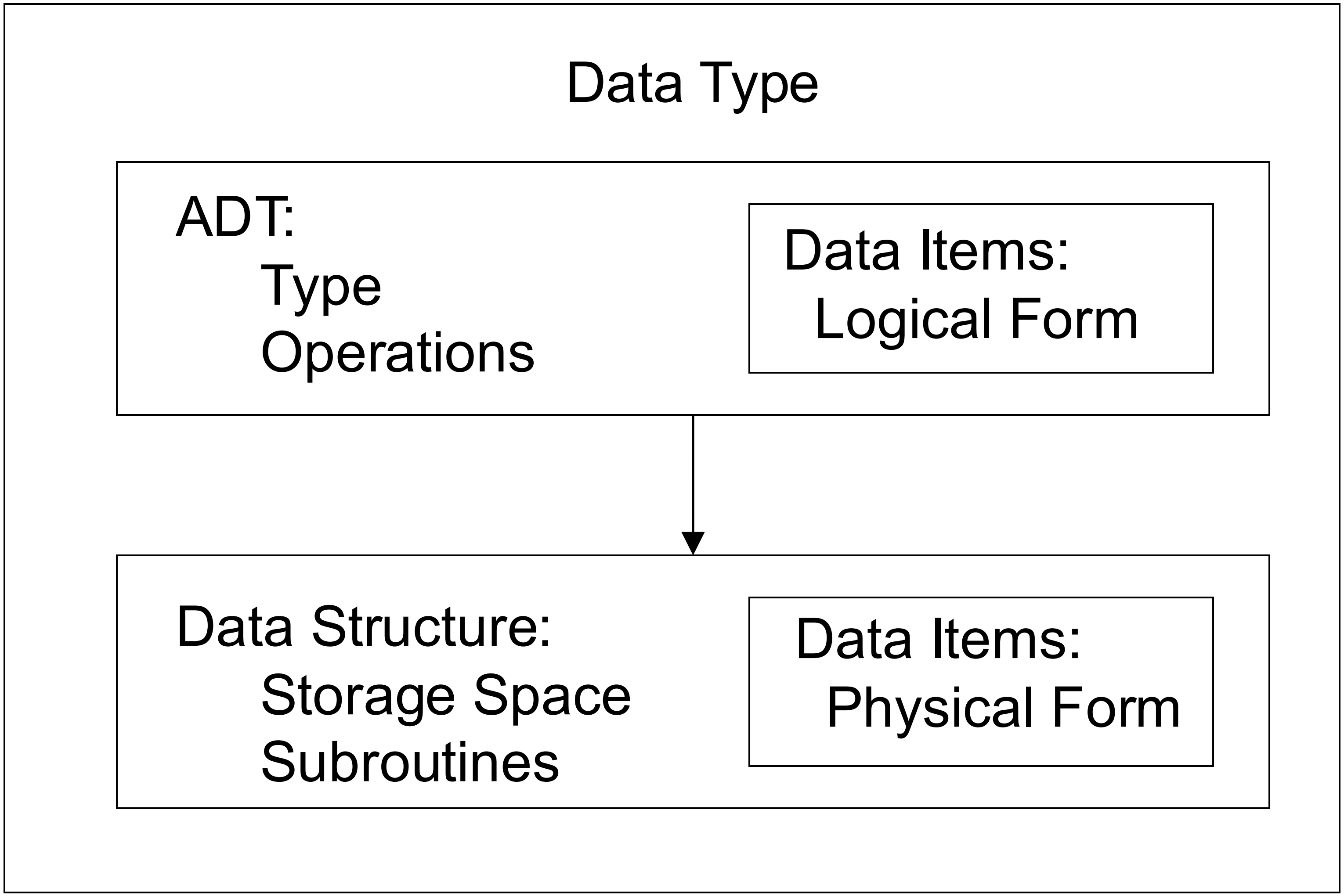
- Data structures may implement the same interface with different performance
  - e.g., an interface for items to order
    - can use a stack, queue, circular array, etc.
- Data structure (DS) vs. File structure (FS)
  - DS usually refers to an organization for data in main memory.
  - FS is an organization for data on peripheral storage, such as a disk drive.

Data Structures -- Logical vs. Physical

- Data items have both a **logical** and a **physical** form.

- Logical form: definition of the data item within an ADT.
  - E.g., Integers in mathematical sense: +, -
- Physical form: implementation of the data item within a data structure.
  - E.g., 16/32 bit integers, overflow.

The relationship



To selecting a data structure

A **three-step** approach:

1. Analyze the problem to determine the basic operations that must be supported.
2. Quantify the resource constraints for each operation.
3. Select the data structure that best meets these requirements. ("simplest")

When choosing a data structure

Ask yourself **three questions**:

1. Are all data items inserted into the data structure at the beginning, or are insertions interspersed with other operations?
2. Can data items be deleted?
3. Are all items processed in some well-defined order, or is search for specific data items allowed?

Birthday matching

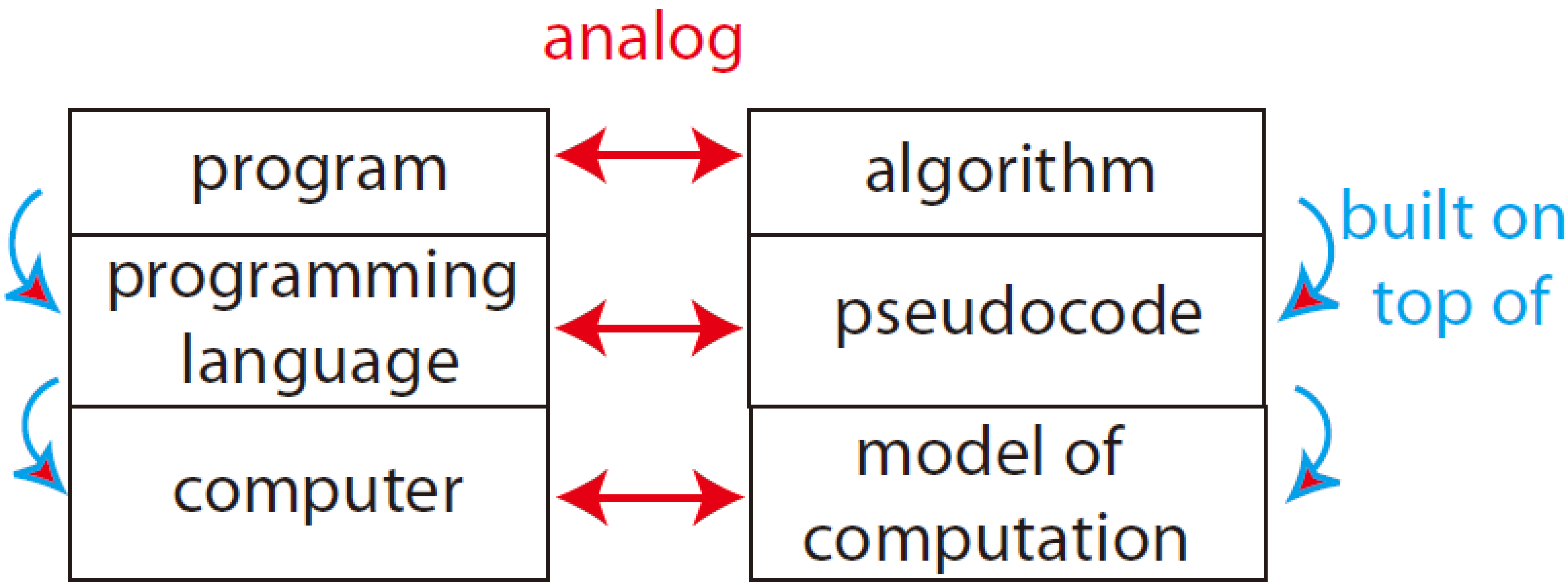
```
/* use array */
int birthdays[50]



/* Class encapsulation in C++ */
class student{
public:
    int birthdays[50];
}
```

Algorithms

- Al-Khwārizmī "al-kha-raz-mi" (c. 780-850)
  - "father of algebra" with his book "The Compendious Book on Calculation by Completion & Balancing"
  - linear & quadratic equation solving: some of **the first algorithms**.

Algorithms

- What is an Algorithm?
  - Mathematical abstraction of computer program
  - Computational procedure to solve a problem



An example algorithm

- An algorithm to solve birthday matching
  - Maintain a record of names and birthdays (initially empty)
  - Interview each student in some order
    - If birthday exists in record, return found pair!
    - Else add name and birthday to record
  - Return None if last student interviewed without success

Birthday matching – a case

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int birthdays[50]  // array
    bool matched = False;
    for(int i=1; i<50-1; i++)
    {
       for(int j=i+1; j<50; j++)
          if(birthdays[i] == birthdays[j])
             matched = True;
             count << "True!" << endl;
             return;
    }
}
```

Algorithms -- correctness

- Programs/algorithms have fixed size, so how to prove correct?
- For small inputs, can use case analysis
- For arbitrarily large inputs, algorithms must be **recursive** or loop in some way
  - use **induction** (why recursion is such a key concept in computer science)

Proof of correctness of birthday matching algorithm

- Induct on k: the number of students in record
- **Hypothesis**: if first k contain match, returns match before interviewing student $k+1$
- **Base case**: $k=0$, first k contains no match
- Assume for induction hypothesis holds for $k=k'$, and consider $k=k'+1$
- If first k' contains a match, already returned a match by induction
- Else first k' do not have match, so if first $k'+1$ has match, match contains $k'+1$
- Then algorithm checks directly weather birthday of student $k'+1$ exists in first k' $\square$

Algorithms - efficiency

- How fast does an algorithm?
  - Could measure time
  - Idea! Count number of fixed-time operations algorithms takes to return
  - Expect to depend on size of input
  - Size of input is often called 'n', but not always!
  - Efficient if return in **polynomial time** w.r.t. input
  - Sometimes no efficient algorithm exists for a problem!

Algorithms – efficiency (cont'd)

- Asymptotic Notation: ignore constant factors and low order terms
  - Upper bounds (O) $\in$, =, is, order

constant	logarithmic	linear	log-linear	quadratic	polynomial	exponential
$\Theta(1)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^c)$	$2^{\Theta(n^c)}$

- Running time analysis: birthday matching
  - Two loops: outer $k \in \{0, \ldots, n-1\}$, inner is $i \in \{0, \ldots, k\}$
  - Running time is $O(n) + \sum_{k=0}^{n-1}\big(O(1) + k \cdot O(1)\big) = O(n^2)$
  - Quadratic in n is **polynomial**. Could be more efficient?

To solving an algorithms problem

A **two-step** approach

1. Reduce to a problem you already know (use data structure or algorithm)
   - Search, sort, shortest path algorithms

2. Design your own (recursive) algorithm
   - Brute Force
   - Decrease and Conquer
   - Divide and Conquer
   - Dynamic Programming
   - Greedy / Incremental

Problems vs. algorithms vs. programs

- Problem: a task to be performed
  - Best though of as inputs and matching outputs
  - e.g., sort a set of numbers
  - Problem definition should include constraints on the resources that may be consumed by any acceptable solution

Problems (cont'd)

- Problems ⇔ Mathematical functions
  - A <u>function</u> is a matching between inputs (the <u>domain</u>) and outputs (the <u>range</u>)
  - An input to a function may be single number, or a collection of information.
  - The values making up an input are called the <u>parameters</u> of the function.
  - A particular input mush always result in the same output every time the function is computed.
- Math. functions is not exactly the same to computer programs.

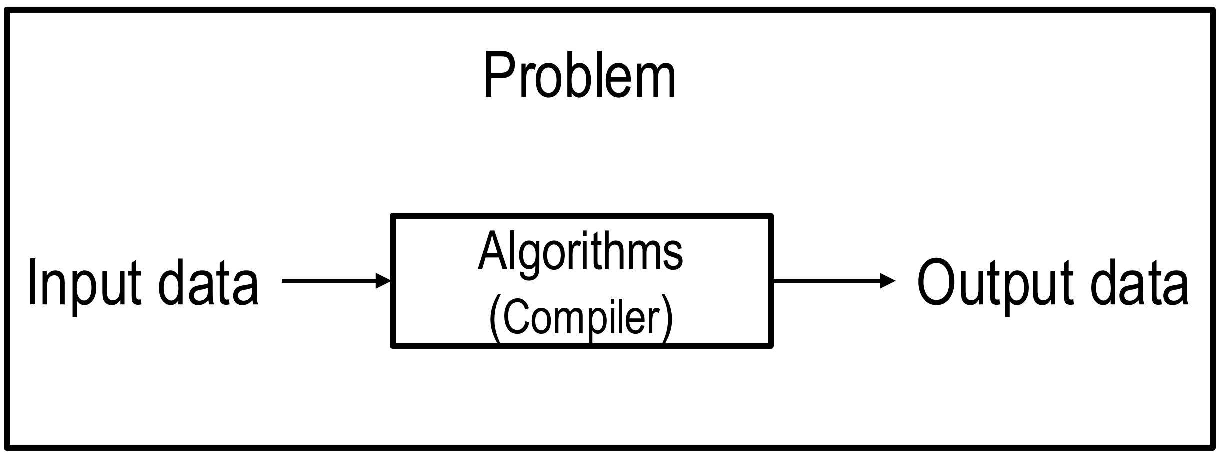
Algorithms and Programs

- Algorithm: a method or a process followed to solve a problem.
  - A recipe.
- An algorithm takes the input to a problem (function) and transforms it to the output.
  - A mapping of input to output.
- A problem can have many algorithms.

Algorithm Properties

- An algorithm has the following five properties:
  - It must be correct.
  - It must be composed of a series of concrete steps.
  - There can be no ambiguity as to which step will be performed next.
  - It must be composed a finite number of steps.
  - It must terminate.
- A computer program is an instance, or concrete representation, for an algorithm in some programming language.

Venn diagram



Conclusion

- Course overview
  - Syllabus
  - Abstract data type, Data structures
  - Problems, Algorithms, Programs
- An example: birthday matching
  - Think and solve birthday matching problem with more efficient algorithms
- Take-home-messages
  - Philosophy of abstraction
  - Simple but not simpler

Peak Finder

Position 2 is a peak if and only if $b \geq a$ and $b \geq c$. Position 9 is a peak if $i \geq h$.

	1	2	3	4	5	6	7	8	9
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**
A case	**6**	**7**	**4**	**3**	**2**	**2**	**0**	**4**	**5**

Problem: Find a peak if it exists (Doe it always exists?)

1	2	…	…	n/2	…	…	n-1	n

Next week

- Math prelims (Chapter I. 2)
  - Set
  - Relations
  - Functions