JICSCI803
Algorithms and Data Structures
September 2020 to January 2021

Before the Subject

Who have taken this subject in your undergraduate study?

Why this subject is important?

Highlights of Lecture 01

Information on this subject
Importance of this subject
Concept and examples of algorithms
Concept and examples of data structures

Questions about the Subject

What are algorithms?

How to evaluate and compare algorithms?

What is data structures?

What is the role of algorithms and data structures in computer science?

Where the data structures and algorithms come from?

What is the learning outcome of this subject?

What is the ways to learn them?

What is the importance of this subject?

Why Study this Course?

- Donald E. Knuth stated “Computer Science is the study of algorithms”
  - Cornerstone of computer science. Programs will not exist without algorithms.
- Closely related to our lives
- Help to guide how others analyze and solve problems
- Help to develop the ability of analyzing and solving problems via computers
- Very interesting if you can concentrate on this course
- Difference from the college version

Subject Outcomes

On successfully completing this subject students should be able to:

1. compare the complexities of algorithms;
2. choose and use appropriate data structures and algorithms for a wide class of problems;
3. make effective use of abstract data types as a design technique and implement abstract data types using C++ classes or C modules or JAVA or Python;
4. develop and use modules that implement algorithms in a generic manner and which can be reused in different applications

CSCI803 Grading Scheme

ASSESSMENT ITEMS & FORMAT	% OF FINAL MARK	GROUP/ INDIVIDUAL	DUE DATE	SUBJECT LEARNING OUTCOMES
Class participation	10			
Assignments	30	Individual	To be advised by the lecturer	1-4
Final examination	60	Individual	Examination Period	1-4

- Bonus is possible for those positive and active students
- NOTES: written submissions in LaTex (pdf)

Marks

- Participation and Assignments (40%)
  - Participation an 4 Assignments
  - To be eligible for a Pass in this subject a student must achieve a mark of at least 40% (i.e. 16/40).
- Final Exam (60%)
  - To be eligible for a Pass in this subject a student must achieve a mark of at least 40% in the Exam (i.e. 24/60).

Assignment

- Assignment 1 is purely written
- Assignments 2,3 &4 are programming assignments
  - NB: 0 marks if it doesn't compile in the labs
  - Very few marks if it doesn't work to spec
  - Some of the marks are for suitable choice of data structures and algorithms.
  - Deductions for poor clarity/style.

Caution

- Please finish your assignments and projects independently.
  - If you really meet problems, you can seek help from tutor, me or other students, but remember not copying, CHEATING!
- Please sign attendance when needed, but remember not signing for others, CHEATING!
- Power off (or ring off) your mobile phone

- *Feedback (positive or negative) is encouraged and welcome, and IMPORTANT. (ask questions!)*
- *Thinking, before, in and after class is encouraged and IMPORTANT.*

Course Prerequisite

- Discrete Mathematics
- C, Java or other programming languages
- Advanced Mathematics
- Ability to form problems from a wide range of tasks.

Pedagogy

What I hear, I forget.
What I see, I remember.
What I do, I understand.

What does this mean?

Explanation of Algorithms and Data Structures

- Algorithms
  - What is an algorithm?
  - Where do they come from?
  - Why do we study algorithms?
- Data structures?
  - More than just data?
  - An organised collection of data

Explanation of Algorithms and Data Structures

- What is an algorithm?
  - A set of rules for carrying out some calculation/procedure
  - An algorithm should be *correct*
  - An algorithm should *terminate*
  - An algorithm should be *efficient*

- What are the desired characteristics of an algorithm in Computer Science?
  - Efficiency, Fast, Less Resource, Robust, Simplicity, General, Easy-to-implement (Implementy)….

Explanation of Algorithms & Data Structures

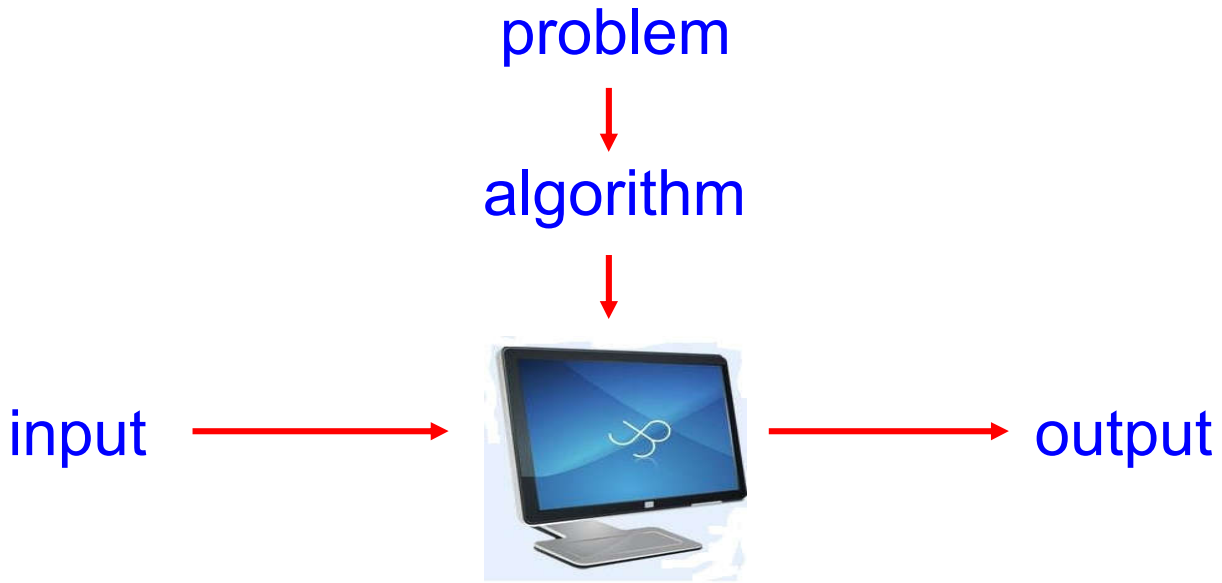
- Where do algorithms come from?
  - Named for the ninth-century Persian mathematician al-Khowârizmî
  - Derived from the need to solve a specific problem
  - Algorithms do not simply come out of thin air (or out of a textbook)

(image of Al Khwarizmi from http://jeff560.tripod.com/)

Explanation of Algorithms and Data Structures

Notion: Algorithms

– An algorithm is a sequence of unambiguous instructions for solving a computational problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.



Explanation of Algorithms and Data Structures

Notion: Algorithms

- More precisely, an algorithm is a method or process to solve a problem satisfying the following properties:
  - Finiteness
    - terminates after a finite number of steps
  - Definiteness
    - Each step must be rigorously and unambiguously specified.

      -e.g., ”stir until lumpy”
  - Input
    - Valid inputs must be clearly specified.
  - Output
    - can be proved to produce the correct output given a valid can be proved to produce the correct output given a valid input.
  - Effectiveness
    - Steps must be sufficiently simple and basic.

      -e.g., check if 2 is the largest integer n for which there is a solution to the equation $x^n + y^n = z^n$ in positive integers x, y, and z

Explanation of Algorithms and Data Structures

Algorithm: A method or process followed to solve a problem.
In other words, an algorithm is a finite set of instructions.

In addition, all algorithms must satisfy the following criteria:

- *It must be correct.*
- *It must be composed of series of concrete steps.*
- *There can be no ambiguity as to which step will be performed next.*
- *It must be composed of a* finite *number of steps.*
- *It must terminate.*
- *Input/Output:* There must be a specified number of input values, and one or more result values.

Explanation of Algorithms and Data Structures

How to describe the *algorithm?*

- *Natural language.*
- *Graphic.*
- *Program.*
- *Formal language.*

Explanation of Algorithms and Data Structures

- Why do we study algorithms?
  - To understand how they work
  - To be able to know when to use them
  - To be able to adapt them to new problems
  - To be able to create new algorithms that are better than the existing ones (not all of us)
    - Better? In what way?

» Faster
» Less memory
» More general
» Simpler description
» ……

Explanation of Algorithms and Data Structures

An example of problem and *algorithm*

The problem of **finding the maximum:**

Find the maximum from *n numbers*.

Select a data structure to store n numbers:

list [n], linked list; stack.

Design an algorithm to solve it:

list [n], linked list; stack.

The following algorithm is to find the maximum.

```
max = list [0];
for (i=1; i<n; i++)
     {
        if  list[i] > max   then max = list[i];
      }
```

Explanation of Algorithms and Data Structures

The *program*

program: A computer program is viewed as an instance or concrete representation of an algorithm in some programming language.

The **distinguishes** **between an algorithm and a program:**

program– presented in a programming language

Algorithm--may was natural language、graphic、a programming Language

A problem is a function or a mapping of inputs to outputs.

An algorithm is a recipe for solving a problem.

A program is an instantiation of an algorithm in a computer programming language.

Algorithm Efficiency: What does it mean?

Many approaches can solve a problem.

How to choose between them?

At the heart of program design are goals:

1) To design an algorithm that is easy to understand, code, and debug.
2) To design an algorithm that makes efficient use of the computer's resource.

The efficiency measurement will be discussed in coming lecture.

Examples

- Is the following a legitimate algorithm?

```
i ←1
While (i <= 10) do
    a ← i + 1
    Print the value of a
End of loop
Stop
```

Examples of Algorithms – Computing the Greatest Common Divisor of Two Integers

- gcd(m, n): the largest integer that divides both m and n.
- First try -- **Euclid's alg:** gcd(m, n) = gcd(n, m mod n)
  - Step1: If n = 0, return the value of m as the answer and stop; otherwise, proceed to Step 2.
  - Step2: Divide m by n and assign the value of the remainder to r.
  - Step 3: Assign the value of n to m and the value of r to n. Go to Step 1.

Methods of Specifying an Algorithm

- Natural language
  - Ambiguous

    "Mike ate the sandwich on a bed."
- Pseudocode
  - A mixture of a natural language and programming language-like structures
  - Precise and succinct.
  - Pseudocode in this course
    - omits declarations of variables
    - use indentation to show the scope of such statements as for, if, and while.
    - use ← for assignment

Pseudocode of Euclid's Algorithm

Algorithm *Euclid(m, n)*

//Computes gcd(m, n) by Euclid's algorithm

//Input: Two nonnegative, not-both-zero integers m and n

//Output: Greatest common divisor of m and n

while n ≠ 0 do

    r ← m mod n

    m ← n

    n ← r

return m

- Questions:
  - Finiteness: how do we know that Euclid's algorithm actually comes to a stop?
  - Definiteness: non ambiguity
  - Effectiveness: effectively computable.

Second Try for gcd(m, n)

- **Consecutive Integer Algorithm**
  - Step1: Assign the value of min{m, n} to t.
  - Step2: Divide m by t. If the remainder of this division is 0, go to Step3;otherwise, go to Step 4.
  - Step3: Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop; otherwise, proceed to Step4.
  - Step4: Decrease the value of t by 1. Go to Step2.
- Questions
  - Finiteness
  - Definiteness
  - Effectiveness
  - Which algorithm is faster, the Euclid's or this one?

Second Try for gcd(m, n)

- **Consecutive Integer Algorithm**

  Input m and n two integers; output gcd(m, n).

  - Step1: m ← max{m, n} ; t ← min{m, n}.
  - Step2: Divide m by t. If the remainder of this division is 0, go to Step3;otherwise, go to Step 4.
  - Step3: Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop; otherwise, proceed to Step4.
  - Step4: Decrease the value of t by 1. Go to Step2.

- Questions
  - Finiteness
  - Definiteness
  - Effectiveness
  - Which algorithm is faster, the Euclid's or this one?

Third try for gcd(m, n)

- **Middle-school procedure**
  - Step1: Find the prime factors of m.
  - Step2: Find the prime factors of n.
  - Step3: Identify all the common factors in the two prime expansions found in Step1 and Step2. (If p is a common factor occurring Pm and Pn times in m and n, respectively, it should be repeated in min{Pm, Pn} times.)
  - Step4: Compute the product of all the common factors and return it as the gcd of the numbers given.
- Question
  - Is this a legitimate algorithm?
  - How to find the prime factors?

Sieve of Eratosthenes

Input: Integer $n \geq 2$

Output: List of primes less than or equal to n

for $p \leftarrow 2$ to n do

    $A[p] \leftarrow p$

for $p \leftarrow 2$ to $\lfloor \text{sqrt}(n) \rfloor$ do

    if $A[p] \neq 0$ //p hasn't been previously eliminated from the list

        $j \leftarrow p * p$

        while $j \leq n$ do

            $A[j] \leftarrow 0$ //mark element as eliminated

            $j \leftarrow j + p$

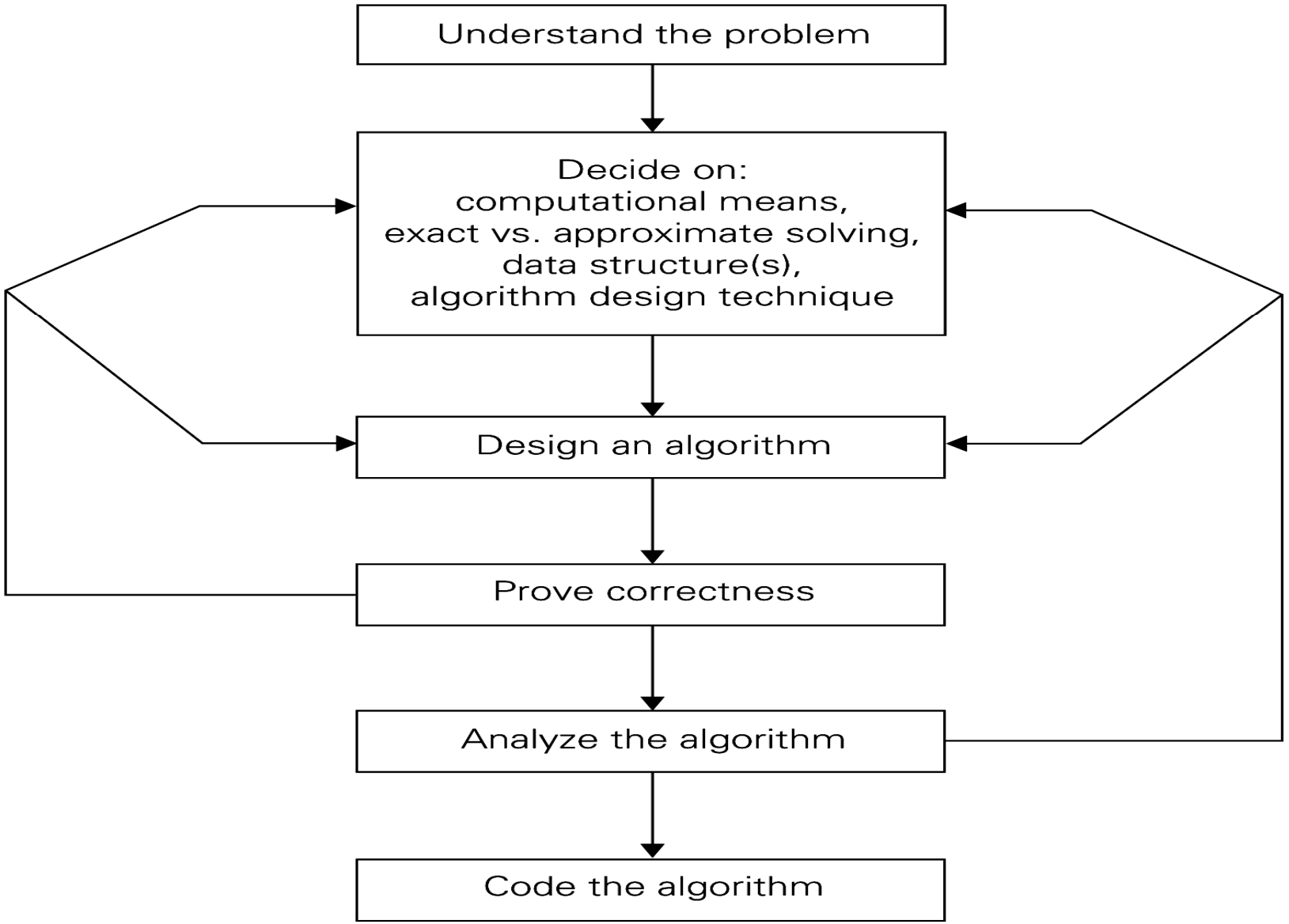
Example: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

What can we learn from the previous 3 examples?

- Each step of an algorithm must be unambiguous.
- The same algorithm can be represented in several different ways. (different pseudocodes)
- There might exists more than one algorithm for a certain problem.
- Algorithms for the same problem can be based on very different ideas and can solve the problem with dramatically different speeds.

Fundamentals of Algorithmic Problem Solving

- Understanding the problem
  - Asking questions, do a few examples by hand, think about special cases, etc.
- Deciding on
  - Exact vs. approximate problem solving
  - Appropriate data structure
- Design an algorithm
- Proving correctness
- Analyzing an algorithm
  - Time efficiency : how fast the algorithm runs
  - Space efficiency: how much extra memory the algorithm needs.
- Coding an algorithm



Algorithm Design and Analysis Process

Two main issues related to algorithms

- How to design algorithms

- How to analyze algorithm efficiency

Algorithm Design Techniques/Strategies

- **Brute force**
- **Divide and conquer**
- **Decrease and conquer**
- **Transform and conquer**
- **Space and time tradeoffs**

- **Greedy approach**
- **Dynamic programming**
- **Backtracking**
- **Branch and bound**
- **Deep Learning**

Analysis of Algorithms

- How good is the algorithm?
  - time efficiency
  - space efficiency
- Does there exist a better algorithm?
  - lower bounds
  - optimality

Example: Fibonacci Number

(image of Leonardo Fibonacci from http://www.math.ethz.ch/fibonacci)

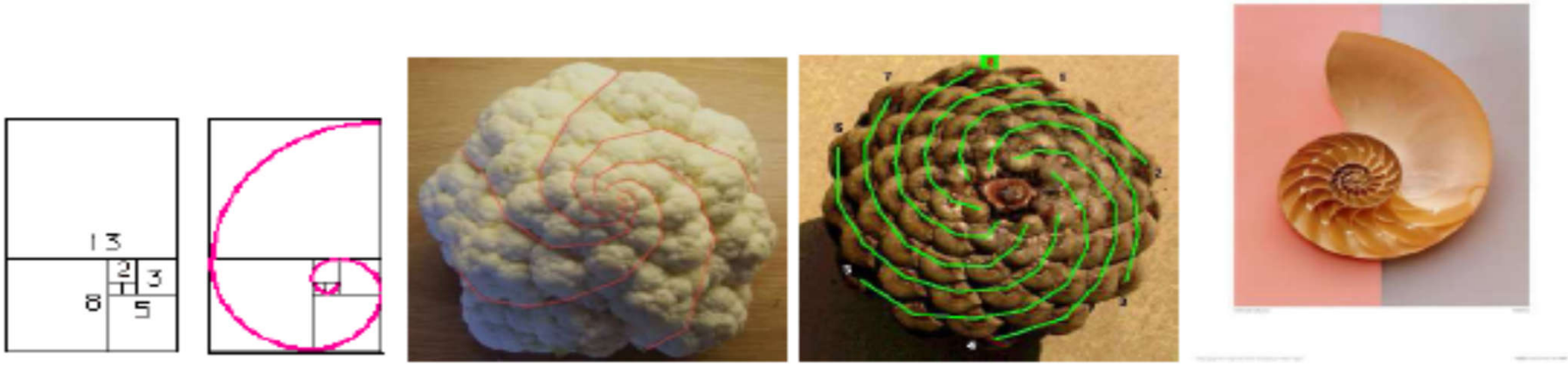
- Fibonacci's original question:
  - Suppose that you are given a newly-born pair of rabbits, one male, one female.
  - Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits.
  - Suppose that our rabbits never die.
  - Suppose that the female always produces one new pair (one male, one female) every month.

- **Question**: How many pairs will there be in one year?
  - 1. In the beginning: (1 pair)
  - 2. End of month 1: (1 pair) Rabbits are ready to mate.
  - 3. End of month 2: (2 pairs) A new pair of rabbits are born.
  - 4. End of month 3: (3 pairs) A new pair and two old pairs.
  - 5. End of month 4: (5 pairs) ...
  - 6. End of month 5: (8 pairs) ...
  - 7. after 12 months, there will be 233 rabits

Recurrence Relation of Fibonacci Number fib(*n*):

$$\text{fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1 \end{cases}$$

{0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …}



(images from *http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fib.html*)

Algorithm: Fibonacci

Problem: What is fib(200)? What about fib(n), where n is any positive integer?

Algorithm 1 fib(n)
if n = 0 **then**
    return (0)
if n = 1**then**
    return (1)
return (fib(n − 1) + fib(n − 2))

Questions that we should ask ourselves.
1. Is the algorithm correct?
2. What is the running time of our algorithm?
3. Can we do better?

Answer of Questions

- Is the algorithm correct?
  - Yes, we simply follow the definition of Fibonacci numbers
- How fast is the algorithm?
  - If we let the run time of fib(n) be T(n), then we can formulate
    $T(n) = T(n-1) + T(n-2) + 3 \approx 1.6^n$
  - $T(200) \geq 2^{139}$
  - The world fastest computer BlueGene/L, which can run 2^{48} instructions per second, will take 2^{91} seconds to compute in 2017. (2^{91} seconds = 7.85×10^{19} years)
  - Can Moose's law, which predicts that CPU get 1.6 times faster each year, solve our problem?
  - No, because the time needed to compute fib(n) also have the same "growth" rate
    - if we can compute fib(100) in exactly a year,
    - then in the next year, spend a year to compute fib(101)
    - if compute fib(200) within a year, wait for 100 years.

Improvement

- Can we do better?
  - Yes, because many computations in the previous algorithm are repeated.

Algorithm 2: fib(n)

comment: Initially we create an array A[0: n]

A[0] ← 0,A[1] ← 1

for i = 2 to n do

    A[i] = A[i − 1] + A[i − 2]

return (A[n])

Important problem types

- sorting
- searching
- string processing
- graph problems
- combinatorial problems
- geometric problems
- numerical problems

Example of computational problem: sorting

- Statement of problem:
  - *Input:* A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
  - *Output:* A reordering of the input sequence $\langle a'_1, a'_2, \ldots, a'_n \rangle$ so that $a'_i \leq a'_j$ whenever $i < j$
- Instance: The sequence <5, 3, 2, 8, 3>
- Algorithms:
  - Selection sort
  - Insertion sort
  - Merge sort
  - (many others)

Selection Sort

- Input: array $a[1], \cdots, a[n]$
- Output: array a sorted in non-decreasing order

Algorithm: for i=1 to n

swap $a[i]$ with smallest of $a[i], \cdots a[n]$

- see also pseudocode, section 3.1

Some Well-known Computational Problems

- Sorting
- Searching
- Shortest paths in a graph
- Minimum spanning tree
- Primality testing
- Traveling salesman problem
- Knapsack problem
- Chess
- Towers of Hanoi
- Program termination

Basic Issues Related to Algorithms

- How to design algorithms
- How to express algorithms
- Proving correctness
- Efficiency
  - Theoretical analysis
  - Empirical analysis
- Optimality

Algorithm design strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer

- **Greedy approach**
- **Dynamic programming**
- **Backtracking and Branch and bound**
- **Space and time tradeoffs**

Analysis of Algorithms

- How good is the algorithm?
  - Correctness
  - Time efficiency
  - Space efficiency
- Does there exist a better algorithm?
  - Lower bounds
  - Optimality

Why study algorithms?

- Theoretical importance
  - the core of computer science
- Practical importance
  - A practitioner's toolkit of known algorithms
  - Framework for designing and analyzing algorithms for new problems

Efficiency

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.

Explanation of Algorithms and Data Structures

- Data structures – more than just data?
  - Organised collection of data
  - Represents an abstract data type in an efficient (hopefully) way

– Efficient? In what way?

» Allows for efficient access by the algorithm

» Uses a minimum of storage

Definition: A data structure is any data representation and its associated operation

(1) data representation

More typically, a data structure is meant to be an organization or structuring for a collection of data items.

(2) associated operation

Such as: search, print, modify, sort, etc.

e.g. for integer, the operations include +,-,*,/,%,...

Three Steps to select a data structure

- Analyze problem, determine the resource constraints
- Determine the basic operations that must be supported and quantify the resource constraints for each operation
- Select the data structure that best meets these requirements

This three-step approach to selecting a data structure operationalizes a data-centered view to design

cost and benefits

Each data structure has associated cost and benefits.

cost: A data structure requires a certain amount of space to store each data item, a certain amount of time to perform a single basic operation, and a certain amount of programming effort.

benefit: To solve a problem efficiently. A solution is said to be efficient if it solves the problem within the required resource constraints.

Abstract Data Type (ADT, class)

ADT: Defines a data type solely in terms of a set of values and a set of operations on that data type.

Each ADT operation is defined by its inputs and outputs.

Encapsulation : Hide implementation details.

An ADT Example

ADT Complex_Number {

Objects: $\{a_1+a_2i \mid a_1, a_2 \in R\}$

where R is the set of real

Operations: ***let*** $x= x_1+x_2i$, $y= y_1+y_2i$

real realpart(x): ***get the real part of x***

real imagpart(x): ***get the imaginary part of x***

complex_number add(x,y): ***return x+y,***

that is $(x_1+y_1)+(x_2+y_2)i$

complex_number subtract(x,y): ***return x-y,***

that is $(x_1-y_1)+(x_2-y_2)i$

complex_number multiplay(x,y): ***return x*y,***

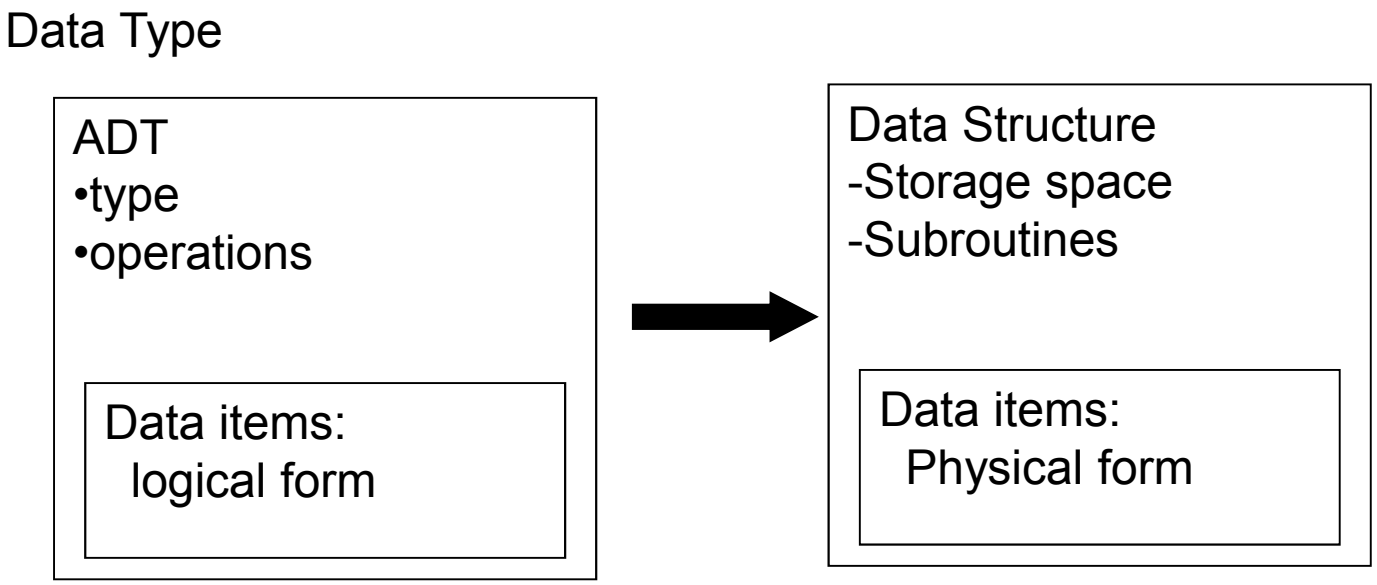
that is $(x_{1*}y_1-x_2{*}y_2)+(x_1{*}y_2+x_2{*}y_1)i$

}

The Data Structure (DS)

The physical implementation of an ADT

the data items have both a logical and a physical form. An ADT is logical form. The implementation of the data item within a data structure is its physical form



Fundamental data structures

- list
  - array
  - linked list
  - string
- stack
- queue
- priority queue

- graph
- tree
- set and dictionary

0 Related Concepts

1. Data

Number

Text: names, articles, books

Images

Videos

Figures

2. Functions (methods)

Input => output

1 A Philosophy of Data Structure

The Need for Data Structure

Data structure + Algorithm = Programming

The system development process:

(1) the purpose of the project (input, output)
(2) break the problem down into management pieces
(3) choose concrete representation for data objects and write algorithm for each operations on them
(4) developing correctness for programs

Data structures organize data➔ more efficient programes

More powerfull Computers➔ more complex applications

More complex applications demand more calculations.

Complex computing tasks are unlike our everyday experience.

Organizing Data

Any organization for a collection of records can be searched. Processed in any order or modified.

The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.

Problems, Algorithms, and Programs

Problem: a problem is a task be performed. It is best thought of in terms of inputs and matching outputs.

- **A problem definition should not include any constraints on how the problem is to be solved.**
- **A problem definition should include constraints on the resources that may be consumed by any acceptable solution.**

Problem can be viewed as function in mathematical sense

A function is a matching between inputs (the domain) and outputs (the range).

- Different inputs are permitted to give the same output.
- Any particular input must always result in the same output every time the function is computed using that input.

Discussion Problems

- Discuss why algorithms are important.
- What is the general process of designing an algorithm?
- What is your experience of designing an algorithm OR designing a program.
- What percentage of published papers in computer science are about bettering algorithms?


Thank you