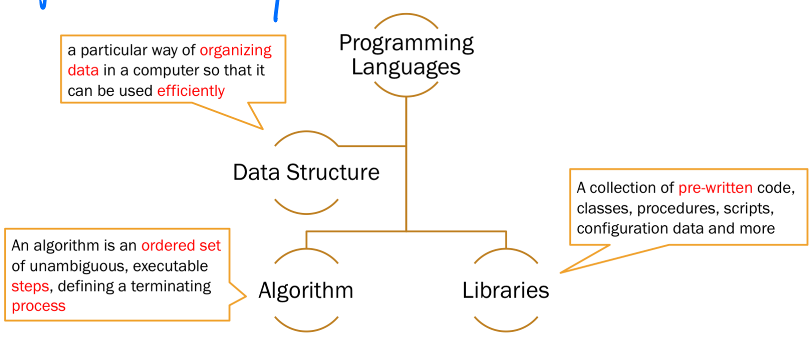
Lecture 8 Introduction to Data Structure and Algorithm

§1 Data Structure and Algorithm

1. Software development skill tree



2. Data structure and algorithm

1° A data structure is a systematic way of organizing and accessing data.

2° An algorithm is a step-by-step procedure for peforming some task in a finite amount of time.

3. Why study data structure and algorithm

- Important for all other branches of computer science
- Plays a key role in modern technological innovation
  - Moore's law: density of transistors in integrated circuits would continue to double every 1 to 2 years
  - However, in many areas, performance gains due to the improvements in algorithms have greatly exceeded even the dramatic performance gains due to increased processor speed
- Provide novel "lens" on processes outside of computer science and technology, such as physics, economic markets, evolution
- Challenging (good for your brain!!) and funny

§2 Primary Analysis of Algorithm

1. The primary analysis of algorithm

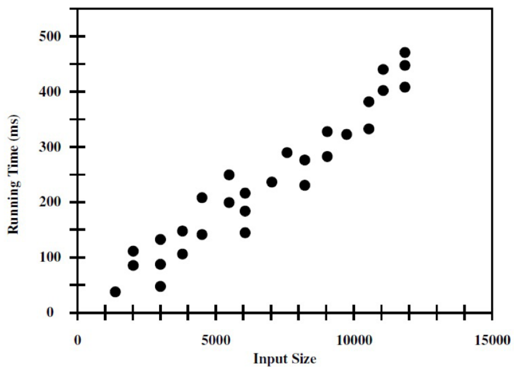
1° Running time

2° Space usage

2. Visualize the running time

1° Running time and space usage are dependent on the size of the input

2° Perform independent experiments on many different test inputs of various sizes.



3. Challenges of experimental analysis

- Experimental running times of two algorithms are difficult to directly compare
  - unless the experiments are performed in the same hardware and software environments
- Experiments can be done only on a limited set of test inputs
  - they leave out the running times of inputs not included in the experiment
  - and these inputs may be important
- An algorithm must be fully implemented in order to execute it to study its running time experimentally

§3 Principle of Algorithm Analysis

1. Principle 1: counting primitive (原始的) operations

We define a set of primitive operations such as the following:

1° Assigning an variable to an object

2° Determing the object associated with an variable

3° Performing an arithmetic operation

4° Compare two numbers

5° Accessing a single element of a Python list by index

6° Calling a function

7° Returning from a function

The running time of different primitive operations will be fairly similar.

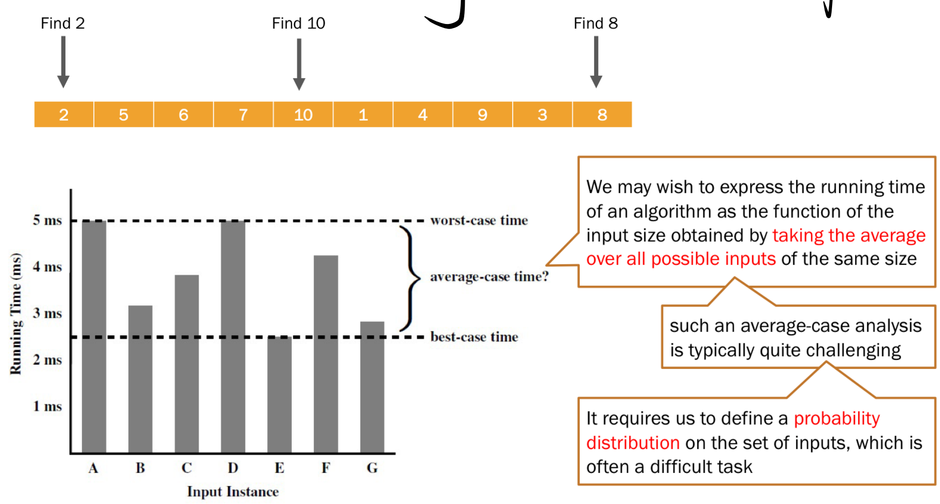
2. Principle 2: measuring operations as a function of input size

We associate a function $f(n)$ that characterizes the number of primitive operations that are performed as a function of the input size n.

3. Principle 3: Focus on the worst-case input

An algorithm may run faster on some inputs than it does on others of the same size.

We will characterize running time in terms of the worst case.



§4 Asymptotic analysis

1. Asymptotic analysis

In algorithm analysis, we focus on the growth rate (增长率) of the running time as a function of the input size n, taking a "big picture" approach

2. The big O notation

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers.

We say that $f(n)$ is $O(g(n))$ if there is a real constant $c>0$ and an integer constant $n_0 \geq 1$ such that

$$f(n) \leq c\,g(n), \text{ for } n \geq n_0$$

e.g. The function $8n+5$ is $O(n)$

The function $7n^5+6n^3+5n$ is $O(n^5)$

The function 2^n+3n^2+n is $O(2^n)$

The big O notation allows us to ignore constant factors and lower-order terms and focus on the main components of the function that affect its growth.

* In general, we should use the big O notation to characterize a function as closely as possible.

§5. Comparative analysis

- Use the big O notation to order classes of functions by asymptotic growth rate
- We may use the following 7 functions to measure the time complexity of an algorithm: constant, logarithm, linear, N-log-N, quadratic, cubic, exponential
- The 7 functions are ordered by increasing growth rate in the following sequence

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}
128	7	128	896	16,384	2,097,152	3.40×10^{38}
256	8	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	512	4,608	262,144	134,217,728	1.34×10^{154}

- 7 functions to measure the time complexity of an algorithm: constant, logarithm, linear, N-log-N, quadratic, cubic, exponential