1.Algorithms and Data Structures

A data structure is a systematic way of organizing and accessing data. 数据结构：一种系统地储存和组织数据的方式。

An algorithm is a **sequence of steps** for performing a **task** in a **finite amount of time**. 算法：在有限时间内完成一项任务的一系列步骤（what and how）。例如：

- Navigate to a place：Root-finding algorithm
- Transfer Audio/Video format：Compression algorithm
- Control solar panels：Optimization algorithm
- Generate an image from 2D/3D model：Rendering algorithm

设计算法时要注意：

- What makes a good algorithm?
- How do you measure efficiency?

Our fundamental interest is to design “good” algorithms which **utilize efficient data structures**. In this context “good” means fast in a way to be clarified（既包括时间也包括空间）.

2.Algorithm Analysis

The analysis of algorithm is the theoretical study of computer program performance and resourse usage：

- Primary interest: Running time (time-complexity) of the algorithm and the operations on data structures.（soft 时间复杂度）
- Secondary interest: Space (or “memory”) usage (space-complexity).（hardware 空间复杂度）

This course focuses on the **mathematical** design and analysis of algorithms and their associated data structures, utilizing relevant mathematical tools to do so.

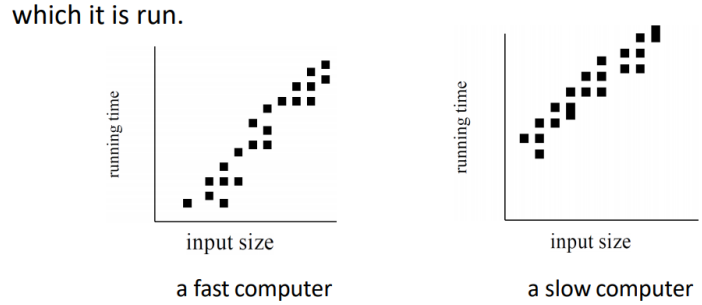
Experimental Analysis of Algorithms（算法的实验性分析）：

运行所需的时间以及空间与输入的大小相关。

大小（size）可以包括：图的边（edge）以及节点（vertices），编码/解码的信息的长度，数字的长度（内存中所占的bit大小）。为了分析算法经常会进行实验（experiments）。

In general, the running time of an algorithm or data structure method increases with the input size.

Running-time depends on both the *size* and *instance* of input and the algorithm used, as well as the software and hardware environment on which it is run.



这种分析办法的局限：

1. 由于需要做实验，而且测试输入的数量有限。

2. 需要所有的测试在同样的硬件和软件上进行。
3. 需要算法实际去实现和执行。

Theoretical Analysis（理论分析）：

实验性分析的好处：

1. 能够把所有可能的输入值考虑进去。
2. 能够在不考虑软硬件环境的前提下比较多个算法的效率。
3. 包括算法的高级描述（pseudo code，伪码）。

When we analyze an algorithm in this fashion, we aim to associate a function f (n) to each algorithm, where f (n) characterizes the running–time in terms of some measure of the input–size, n. 根据输入的大小n来判断运行时间。

Typical functions include: n, log n, n2 , n log n, 2n ,...

‘Algorithm A runs in time proportional to n’ 运行时间与A成正比。

实验分析的需求：

- A language for describing algorithms.
- Computational model in which algorithms are executed.
- Metric for measuring performance.
- A way of characterizing performance.

Pseudo–code：

算法的高级描述语言，提供更结构化的算法描述，允许对算法进行高级分析以确定算法的运行时间以及内存需求。

伪码是高级编程语言和自然语言的混合，包括：

expressions, declarations, variable initialization and assignments, conditionals, loops, arrays, method calls, etc.

We define a set of high–level primitive operations（原始操作） that are largely independent from the programming language used.

Primitive operations include the following:

- Assigning a value to a variable
- Calling a method
- Performing an arithmetic operation (for example, adding two numbers)
- Comparing two numbers
- Indexing into an array
- Following an object reference
- Returning from a method

算法的运行时间即计算上述操作进行的次数。

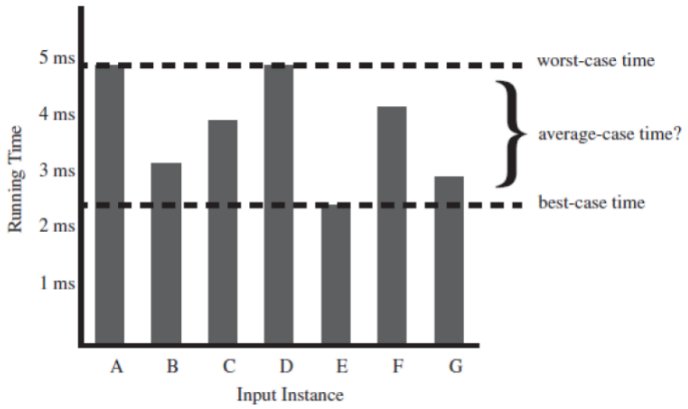
Random Access Machine（随机存储器）：

这种简单计算原始操作的方法产生了一个称为随机存储器的计算模型，CPU连接到一组存储单元，每个记忆单元可以存储一个数字、字符或地址。

假定：所有原始操作需要的时间相同。

输入值不同，运行时间也可能不同：

- Average-case complexity refers to running time as an average taken over all inputs of the same size.
- Worst-case complexity refers to running time as the maximum taken over all inputs of the same size.



An average-case analysis also typically requires that we calculate expected running times based on a given input distribution.

例如：

```
Algorithm arrayMax(A,n):
input: array A
output: maximum element
currentMax
1    currentMax ← A[0]
2    for j ← 1 to n-1 do
3            if currentMax<A[j]
4                then currentMax ← A[j]
5    return currentMax
```

How many primitive operations?

第一行：2 indexing+assignment

第二行：initializing j：1 verifying j<n n

第五行：returing：1

第三行和第四行：2（n−1） array indexing+comparing/assignment

每次循环里的值进行一次后，要对计数器的值累加并赋值给计数器，此时也需要 2（n−1)次。

最佳情况：即 第四行永远不需要运行，可以理解为第一个数即为最大值，因此只需要执行比较操作，此种情况的原始操作次数为5n。

最差情况：即 最后一个数才是最大值，因此需要把上述所有值相加，此种情况的原始操作次数为7n−2。