Lecture 2
Algorithm Analysis

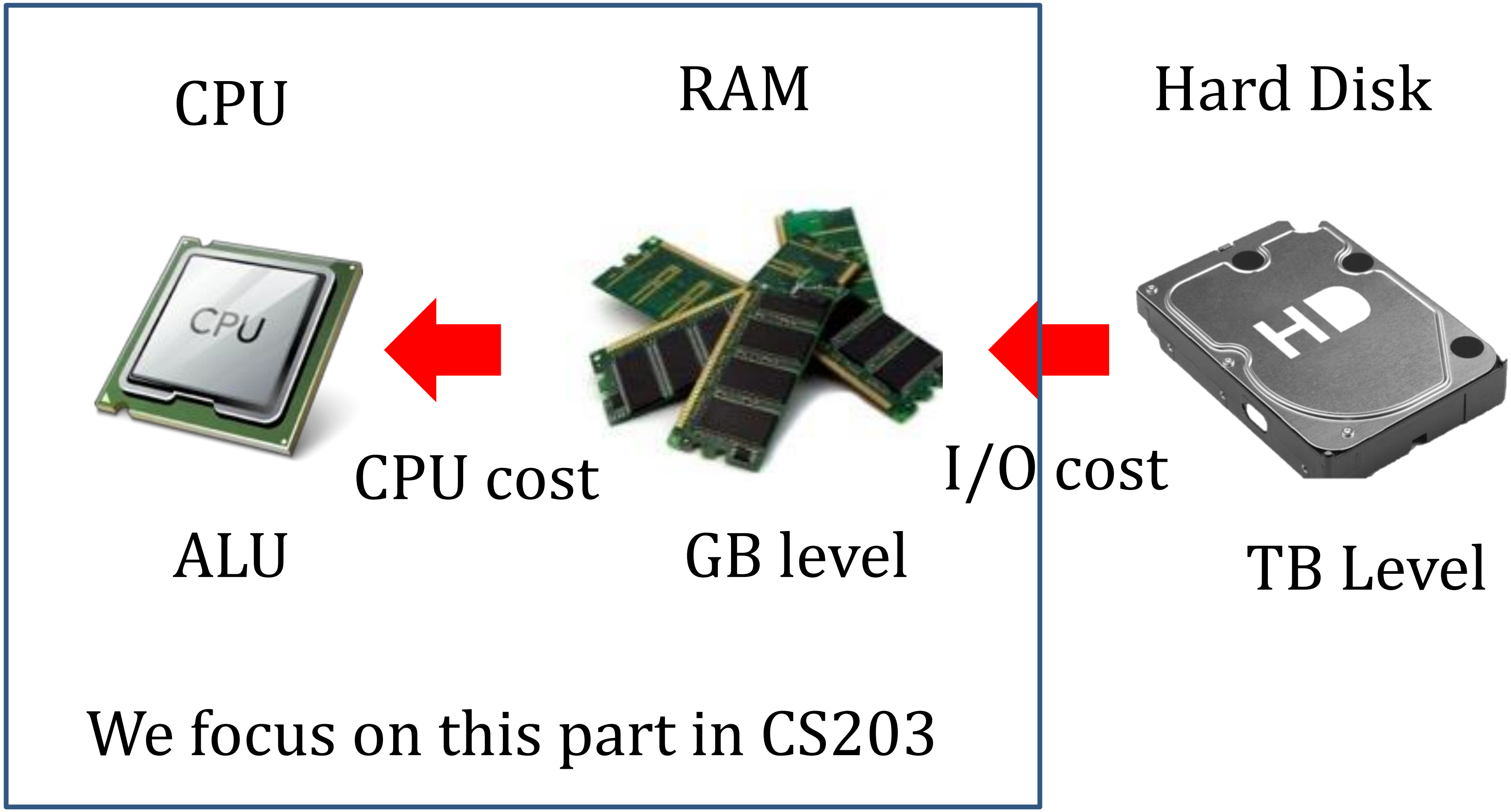
Bo Tang @ SUSTech, Fall 2022

Several pages are based on the notes by Dr. Ken Yiu (PolyU) and Dr. Yufei Tao(CUHK)

Our Roadmap

- RAM Computation Model
  - Memory, CPU, Algorithm
  - Algorithm, Pseudocode
- Worst Case Analysis
  - Binary Search Problem
  - Big O notation

RAM Computation Model



CPU

RAM

Hard Disk

CPU cost

I/O cost

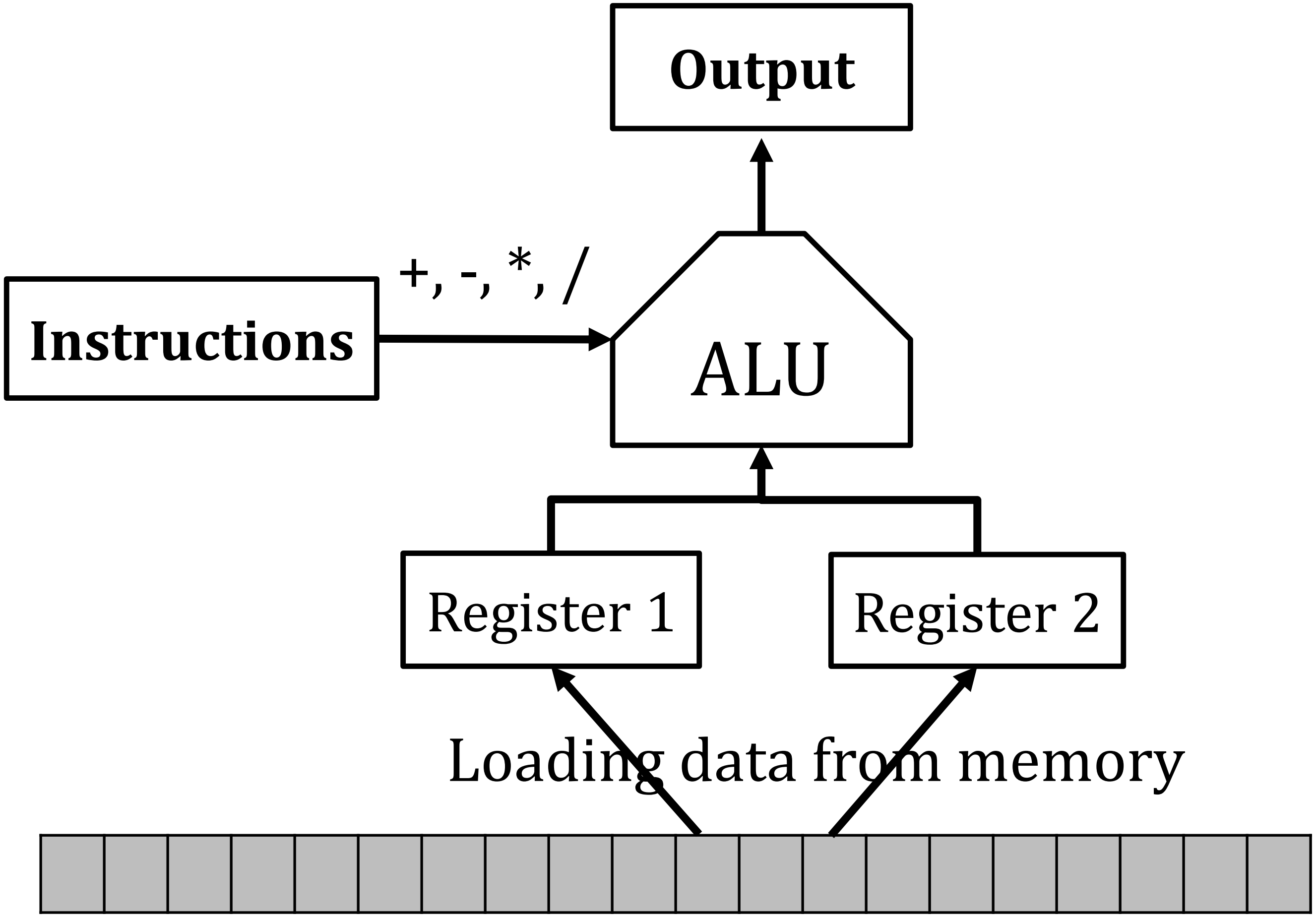
ALU

GB level

TB Level

We focus on this part in CS203

RAM Computation Model



Output

+, -, *, /

Instructions

ALU

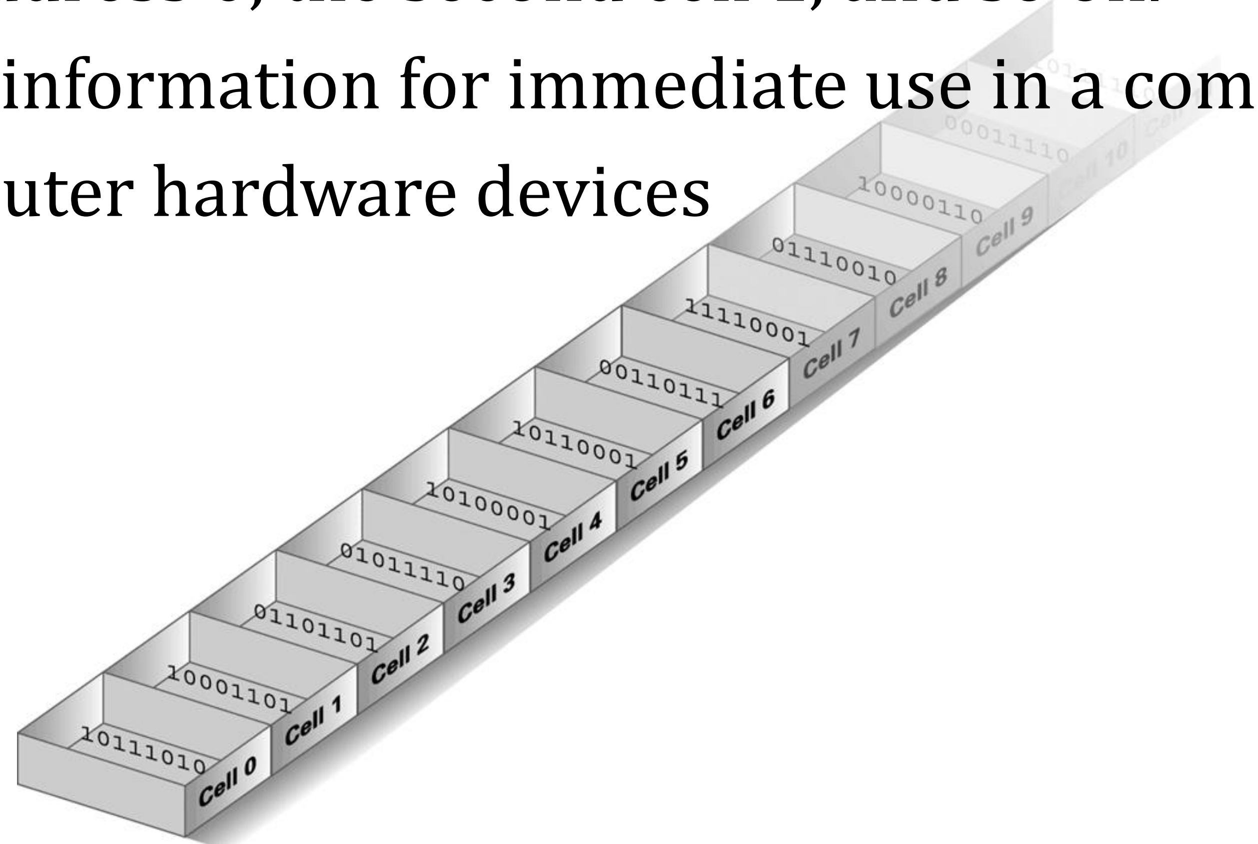
Register 1

Register 2

Loading data from memory

Memory

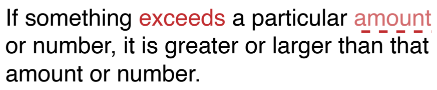
- A finite sequence of cells, each cell has the same number of bits.
- Every cell has an address: the first cell of memory has address 0, the second cell 1, and so on.
- Store information for immediate use in a computer
- Computer hardware devices



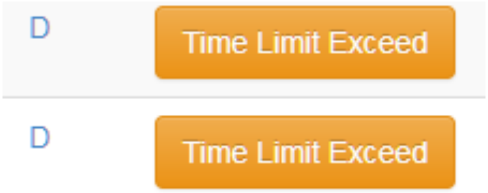
Center Process Unit (CPU)

- Contains a fixed number of registers
- Basic (atomic) operations
  - **Initialization**
    - Set a register to a fixed values (e.g., 100, 1000, etc.)
  - **Arithmetic** (ALU)
    - Take integers a, b stored in two registers, calculate one of {+, -, *, /} and store the result in a register
  - **Comparison / Branching**
    - Take integers a, b stored in two registers, compare them, and learn which of {a<b, a=b, a>b} is true.
  - **Memory Access**
    - Take a memory address A currently stored in a register, Do the READ (i.e., load data from memory) or WRITE (i.e., flush data to memory)operator

Algorithm Analysis



If something exceeds a particular amount or number, it is greater or larger than that amount or number.



D Time Limit Exceed

D Time Limit Exceed

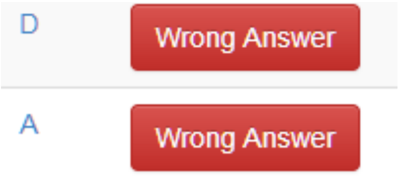
- **Algorithm**
  - A sequence of basic operations
- **Algorithm Analysis**
  - **Cost analysis**
    - Algorithm cost (running time) is the length of the sequences, i.e., the number of basic operations
    - My algorithm is correct, why my submission is **TLE**?
    - Is your algorithm fast?
      - Focus on the order of growth (how the running time grows for large n)

the length of the sequence

- Unless otherwise stated, we refer algorithm analysis as cost analysis in CS203

time & space complexity analysis

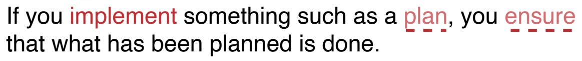
Algorithm Correctness Analysis


D Wrong Answer

A Wrong Answer

- **Correctness** analysis
  - I have passed all test cases, why is still **WA**?
  - It is not enough even if you have tested your algorithm on many instances
    - Will your algorithm fail on some other instances?
  - Proof your algorithm is correct
  - Guarantee your implementation is correct

{ proof your algorithm : correct
| garantee your implementation is correct : bug-free


If you implement something such as a plan, you ensure that what has been planned is done.

- Software testing is an individual course in other many Universities
  - We will not introduce software testing techniques in this course.

Example 1: Summation

- **Problem**: given integer n, calculate 1+2+3+…+n

Pseadocode: without ambiguity

```
① load n from memory to b
② register a ← 1. c ← 0
③ repeat
④     c ← c+a
⑤     a ← a+1
⑥ until a>b
⑦ return c
```

- **Algorithm**:
  - Initialize variable a to 1, b to n, c to 0
  - Repeat the following until a > b:
    - Calculate c plus a, and store the result to c.
    - Calculate a plus 1, and store the result to a.

Alg:

```
init a ← 1
     b ← n
     c ← 0
repeat the following until a>b
     c ← c+a
     a ← a+1
```

- **Cost** of the algorithm:
  - 3 + n + n + n = 3n + 3

cost analyse:

① : 1 memory access

② : 2 initialization

③-⑥ : n (1+1+1)

↓

interation

- Which atomic operations are performed?
- Algorithm is described by English words

Example I: Summation

- **Algorithm**:

```
load n from memory to register b
register a ← 1, c ← 0
repeat
        c ← c + a
        a ← a + 1
until a > b
return c
```

another Algorithm

~

$a \leftarrow 1$

$a \leftarrow a+b$

$a \leftarrow a \times b$

$a \leftarrow a/2$

Return a

cost analysis: 5

- The above is **pseudocode**, it serves the purpose of express (without **ambiguity**) how our algorithm runs.
- **Pseudocode** does not reply on any particular programming language

Example II: Summation

- **Problem**: given integer n, calculate 1+2+3+…+n
- **Cost** of the above algorithm: 3n + 3
- Can we make it faster?
- In our middle school math course:

  1+2+3+…+n = (1+n)*n / 2

Example II: Summation

- **Algorithm**:

```
load n from memory to register b
register a ← 1
a ← a + b
a ← a * b
a ← a / 2
return a
```

- **Cost of the algorithm = 5**
  - This is significantly faster than the previous algorithm
  - The time of the previous algorithm increases linearly with ***n***
  - The time of this algorithm remains constant with ***n***

Our Roadmap

- RAM Computation Model
  - Memory, CPU, Algorithm
  - Algorithm, Pseudocode

✓

- Worst Case Analysis
  - Binary Search Problem
  - Big O notation

Search Problem

- An array ***A*** of ***n*** integers have been sorted in ascending order. Design an algorithm to determine whether given value ***t*** exists in ***A***.

- Example

A	5	8	10	13	16	19	27	46	51	86

  - ***t = 16,*** the result is “TRUE”

  - ***t = 17,*** the result is “FALSE”

Search Problem

- The First Algorithm
  - Simply read the value of $\mathbf{A}[i]$ for each $i \in [1, n]$
  - If any of those cell equals to ***t***, return “TRUE”, otherwise return “FALSE”

- Pseudocode:

```
variable i ← 1
Repeat
    if A[i] = t then
        return “TRUE”
    i ← i + 1
until i > n
return “FALSE”
```

Running Time of the First Algorithm

A	5	8	10	13	16	19	27	46	51	86

- How much time does the algorithm require?
  - If ***t*** is 5, the algorithm has running time = 3
  - If ***t*** is 6, the algorithm has running time = 4***n*** + 1 = 41

在纠结为何为4n
其实也不必.
因为最后会化成 O(n).

- In computer science, it is an art to design algorithms with performance guarantees.

- What is the largest running time on the worst input with ***n*** integers?

Worst-Case Running Time

The worst-case running time (or worst case cost) of an algorithm under a problem size ***n***, is defined to be the largest running time of the algorithm on all the inputs of the same size ***n***.

Worst-Case Time of Search Problem

- Our algorithm has worst-case time

$$f(n) = 4n + 1$$

- In other words, the algorithm will terminates with a cost at most ***4n+1***.

- This is a performance guarantee on every ***n***

- Can we make it faster?
    - **Binary search algorithm**

Binary Search Algorithm

- We utilize the fact that array ***A*** has been sorted in ascending order.

- Let us compare t to the element ***x*** in the middle of ***A*** (i.e., **A[n/2]**)
  - If ***t = A[n/2],*** we have found ***t***, return “TRUE”, terminate
  - If ***t < A[n/2],*** we can ignore ***A[n/2+1] to A[n]***
  - If ***t > A[n/2],*** we can ignore ***A[0] to A[n/2]***

- In the 2nd and 3rd cases, we have at most ***n/2*** elements. Then repeat the above on these left elements.

Binary Search Algorithm

A	5	8	10	13	16	19	27	46	51	86	***t=27***
A	5	8	10	13	**16**	19	27	46	51	86	**< t**
A						19	27	**46**	51	86	**> t**
A						19	**27**	46			**= t**

Binary Search Algorithm

- Binary Search in Pseudocode

```
left ← 1, right ← n
repeat
  mid ← (left+right)/2
  if (t = A[mid]) then
        return TRUE
  else if (t < A[mid]) then
        right ← mid -1
  else
        left ← mid + 1
  until left > right
  return FLASE
```

for standard binary search which search for a certain value, sometimes the value is not included in the array. So the loop condition is left<=right, and the index are returned only when a[mid] == target. Although left==right, if the a[mid]!=target, we still return -1.

Worst-Case Time of Binary Search

- We call the elements from left to right as surviving elements

- Line 1: initialization: 2 basic operations
- Line 2 – 10: iteration, each iteration performs at most 9 basic operations
- Line 11: termination

- How many iterations in the algorithm?

Worst-Case Time of Binary Search

迭代

- How many iterations in the algorithm?
  - After the ***1st*** iteration, the number of surviving elements is at most **n/2**
  - After the ***2nd*** iteration, the number of surviving elements is at most **n/4**
  - In general, after ***i-th*** iteration, the number of surviving elements is at mots n / 2^i
  - Suppose that there are ***h*** iterations in total, it holds that ***h*** is the smallest integer satisfying (why?):

$$n / 2^h < 1$$

  - Then, $h > \log_2 n \rightarrow h = 1 + \log_2 n$
- Thus, the worst case time of binary search is at most:

$$g(n) = 2 + 9h = 2 + 9(1 + \log_2 n)$$

- This is a performance guarantee that holds on all values of ***n***.

Search Problem

- Running time of two algorithms, with input size *n*
  - Algorithm 1: $f(n) = 4n + 1$ (operations)
  - Algorithm 2: $g(n) = 9\log_2 n + 11$ (operations)

- Which algorithm is better?
  - Algorithm 2. Why?
  - We care about the running time at *large input size*
  - Constant factors do not affect *the order of growth*

Asymptotic Analysis

a straight line that is closely approached by a plane curve so that the perpendicular distance between them decreases to zero as the distance from the origin increases to infinity

- Running time of two algorithms, with input size *n*
  - Algorithm 1: f(n) = 4n +1 (operations)
  - Algorithm 2: g(n) = 9$\log_2$ n + 11 (operations)
- In computer science, we rarely calculate the time to such a level.
- We ignore all the constants, but only worry about the dominating term. ⇒ $f(n)=n$ $g(n)=\log_2 n$
  - Why not constant? 10n VS. 5n? Which one is faster?
  - "it depends", 10n comparison, 5n multiplication cpu cycle n
  - Why dominating term: 3n VS. $\log_2$ n ? Which one is faster
  - "$\log_2$ n " is better than 3n in theoretical computer science

忽略底层cpu并行运行
时间导致的差异

Big-O notation

a way to ignore constant

- Let $f(n)$ and $g(n)$ be two functions of n.
- We say that $f(n)$ grows asymptotically no faster than $g(n)$ if there is a constant $c_1 > 0$ such that:

在此处并不意为"渐近"

$$\exists \quad f(n) \leq c_1 \cdot g(n)$$

holds for all $n \geq c_2$.

- We denote this by $f(n) = O(g(n))$
- We say that 5n is considered equally fast as on with 10n, why?
- Big-O capture this by having both of following true (can you prove that?):

$f(n)=10n$ $c_1=5$
$c_2=0$
$g(n)=5n$

$$10n = O(5n)$$

$$5n = O(10n)$$

Big-O example

- $10000\log_2 n$ is considered better than n. Big-O capture this by having both of following true:

$$10000\log_2 n = O(n)$$

$$n \neq O(10000\log_2 n)$$

- Proof of $\boldsymbol{10000\log_2 n = O(n)}$
- There are constants $c_1 = 1, c_2 = 2^{20}$ such that

$$10000\log_2 n \leq c_1 n$$

holds for all $n \geq c_2$

Big-O example

$\log_a^n = O(\log_b^n)$

$\log_a^n \le \log_b^n \cdot c_1 \Longrightarrow \frac{\lg a}{\lg b} \le c_1$

$n \ge c_2$

$\downarrow$ constant

$\Longrightarrow$ a.b can be ANY constant

$\Longrightarrow$ 在计算算法的复杂度时直接写 log n. 不用再写其中的常数.

- Proof of $n \neq O(10000\log_2 n)$
- We can proof it by contradiction. Suppose that are constant c_1, c_2 such that

先找 c_1

$\Longrightarrow$ 用 n 表示出 c_1

$$n \le c_1 \cdot 10000\log_2 n$$

holds for all $n \ge c_2$. The above can be rewritten as:

$$\frac{n}{\log_2 n} \le c_1 \cdot 10000$$

$$\lim_{n\to\infty} \frac{n}{\log_2 n} = \lim_{n\to\infty} \frac{1}{\frac{1}{\ln 2 \cdot n}} = \infty$$

however, $\frac{n}{\log_2 n}$ tends to be ∞ as n increases.

Therefore, the inequality cannot hold for all $n \ge c_2$

$$\left.\begin{array}{l} \log_2 n = O(n) \\ n \neq O(\log_2 n) \end{array}\right\} \Longrightarrow \log_2(n) \text{ is faster}$$

Exercise

- Is $(5n^2 + 3n) = O(n^2)$?
  - Fix c=6 and n_0=3, then prove $f(n) \leq c\, g(n)$

    [note: other choices also possible]
- Is $(5n^2 + 3n) = O(n^3)$?
- Is $(5n^2 + 3n) = O(n)$?

- Proof the following statements:

$$10000 = O(1)$$

$$100\sqrt{n} + 10n = O(n)$$

$$1000n^{1.5} = O(n^2)$$

$$(\log_2 n)^3 = O(\sqrt{n})$$

$$\log_a n = O(\log_b n) \text{ for } a>1, b>1$$

Asymptotic Analysis

- Henceforth, we will describe the running time of an algorithm only in the asymptotical (i.e., big-O) form, which is also called the algorithm's time complexity.

- Instead of saying the running time of binary search is $g(n) = 8\log_2 n + 10$, we will say $g(n)=O(\log n)$, which captures the fastest-growing term in the running time. This is also the binary search's time complexity.

NOT merge sort !

$T(n) = T(\frac{n}{2}) + 1$

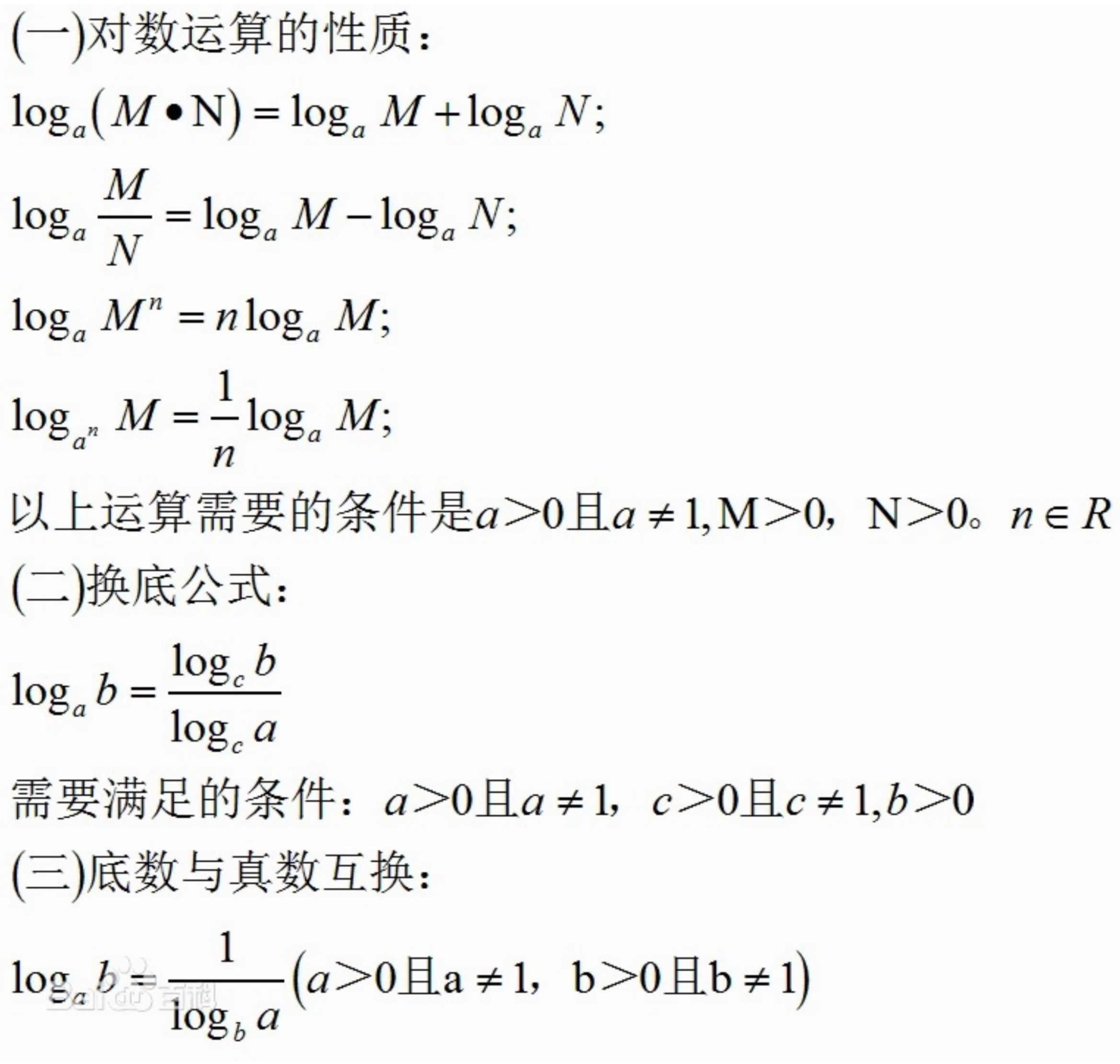
$n \cdot (\frac{1}{2})^x = 1$

$x = \log_{\frac{1}{2}}^{\frac{1}{n}}$

$= \log_{\frac{1}{2}}^{1} - \log_{\frac{1}{2}}^{n}$

$= 0 - \log_{\frac{1}{2}}^{n}$

$= \log_2^{n}$



(一)对数运算的性质：

$$\log_a\left(M \bullet \mathrm{N}\right) = \log_a M + \log_a N;$$

$$\log_a \frac{M}{N} = \log_a M - \log_a N;$$

$$\log_a M^n = n\log_a M;$$

$$\log_{a^n} M = \frac{1}{n}\log_a M;$$

以上运算需要的条件是$a>0$且$a \neq 1$,M＞0，N＞0。$n \in R$

(二)换底公式：

$$\log_a b = \frac{\log_c b}{\log_c a}$$

需要满足的条件：$a>0$且$a \neq 1$，$c>0$且$c \neq 1, b>0$

(三)底数与真数互换：

$$\log_a b = \frac{1}{\log_b a}\left(a>0 \text{且} \mathrm{a} \neq 1\text{，} \mathrm{b}>0 \text{且} \mathrm{b} \neq 1\right)$$

Worst-Case of Algorithms

Complexity		***Algorithm***
O(1)	Constant time	E.g., Compare two numbers
O(log n)	Logarithmic	E.g., Binary search (on a sorted array)
O(n)	Linear time	E.g., Search (on a unsorted array)
O(n log n)		E.g., Merge sort heap sort
O(n^2)	Quadratic	E.g., Selection sort Bubble Insertion
O(n^3)	Cubic	E.g., Matrix multiplication $A_{n\times n} \times B_{n\times n}$
O(2^n)	Exponential	E.g., Brute-force search on boolean satisfiability Brute Force Algorithms are exactly what they sound like – straightforward methods of solving a problem that rely on sheer computing power and trying every possibility rather than advanced techniques to improve efficiency.
O(n!)	Factorial	E.g., Brute-force search on traveling salesman

many algorithm can be described as O(n!). BUT it is NOT as tight as possible

✓ $4n = O(n!)$

⇒ but we need the TIGHTEST one

The travelling salesman problem (also called the travelling salesperson problem or TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in theoretical computer science and operations research.

Big-Ω & Big θ are NOT required to be understand very well

Big-Ω notation

- Let *f(n)* and $g(n)$ be two functions of *n*.
- We say that *f(n)* grows asymptotically no slower than $g(n)$ if there is a constant $c_1 > 0$ such that:

both Big-Ω & Big-O requires $c_1>0$

$$f(n) \geq c_1 \cdot g(n)$$

Big-O notation

≤0

holds for all $n \geq c_2$.

- We denote this by $f(n) = \Omega(g(n))$
- Examples:
  - $\log_2 \mathrm{n} = \Omega(1)$
  - $0.001\mathrm{n} = \Omega(\sqrt{\mathrm{n}})$

Big-Θ notation

- Let *f(n)* and $g(n)$ be two functions of n.
- If *f(n)* $= O(g(n))$ and *f(n)* $= \Omega\big(g(n)\big)$, then we define: *f(n)* $= \Theta\big(g(n)\big)$ to indicate *f(n)* grows asymptotically as fast as $g(n)$

f(n) grows as fast as g(n)

- Examples:
  - $1000 + 30\log n + 1.5\sqrt{\mathrm{n}} = \Theta(\sqrt{\mathrm{n}})$