# UNIT 1: INTRODUCTION

## MODULE1. ALGORITHMIC THINKING: PEAK FINDING

## LECTURE 1: INTRODUCTION AND PEAK FINDING

### THEME

- Efficient procedures for solving problems on large inputs (Ex: U.S. Highway Map, Human Genome)
    - inputs could be super large
- Scalability
    - the concept of "large" changes

# MODULES

1. Algorithmic Thinking: Peak Finding
2. Sorting & Trees: Event Simulation
3. Hashing: Genome Comparison
4. Numerics: RSA Encryption
5. Graphs: Rubik's Cube
6. Shortest Paths: Caltech $\rightarrow$ MIT
7. Dynamic Programming: Image Compression
8. Advanced Topics

# PEAK FINDER

## One-dimensional Version

### Straightforward Algorithm

$\Theta(n)$ Complexity

### Divide & Conquer Algorithm

- If $a[n/2] < a[n/2-1]$ then only look at left half $1 \ldots n/2 - - - 1$ to look for peak
- Else if $a[n/2] < a[n/2+1]$ then only look at right half $n/2+1 \ldots n$ to look for peak
- Else $n/2$ position is a peak

$T(n) = T(n/2) + \underbrace{\Theta(1)}_{\text{to compare a[n/2] to neighbors}} = \Theta(1) + \ldots + \Theta(1)$ $(\log_2(n)$ times

$) = \Theta\left(\log_2(n)\right)$

13 seconds vs. 0.001 seconds

## Two-dimensional Version

### Greedy Ascent Algorithm

$\Theta(mn)$ Complexity

### Divide & Conquer Algorithm

- Pick middle column $j = m/2$
- Find global maximum on column $j$ at $(i, j)$
- Compare $(i, j-1), (i, j), (i, j+1)$
- Pick left columns of $(i, j-1) > (i, j)$
- Similarly for right
- $(i, j)$ is a 2D-peak if neither condition holds
- Solve the new problem with half the number of columns.
- When you have a single column, find global maximum and you're done.

$T(n, m) = T(n, m/2) + \Theta(n)$ (to find global maximum on a column - (n rows))

$$\begin{aligned} T(n, m) &= \underbrace{\Theta(n) + \ldots + \Theta(n)}_{\log m} \\ &= \Theta(n \log m) = \Theta(n \log n) \text{ if m = n} \end{aligned}$$

## *THINKING*

**The first lecture introduced the content of the course. In addition to algorithms' effectiveness, the main concern is the efficiency on large data set.**

**Taking local peak finding as an example, we can see that the optimized algorithm has a great advantage in efficiency over the greedy algorithm. In addition, the algorithm proof and the calculation of asymptotic complexity are also discussed.**

算法基础的第一个讲座，介绍了一下课程的内容，关心的内容除了算法的有效性外，主要是在大数据集上的效率表现。

以局部峰值查找（peak finding）为例，可以看到优化后的算法比贪婪算法在效率上的很大的优势。另外也初步讨论了算法证明以及算法渐近复杂度（asymptotic complexity）的计算。

# R1. ASYMPTOTIC COMPLEXITY, PEAK FINDING

## ASYMPTOTIC COMPLEXITY

$$g(x) = (1 + \sin(x))x^{1.5} + x^{1.4}$$

$$\begin{aligned} g(x) &= \Theta(\quad) && \text{(upper and lower bound)} \\ &= \Omega\left(x^{1.4}\right) && \text{(lower bound)} \\ &= O\left(x^{1.5}\right) && \text{(upper bound)} \end{aligned}$$

In CS, we usually use $O$, it usually means $\Theta$

In CS, runtime complexity usually looks like:

$O(1)$
$O(\log N)$
$O(N)$
$O(N \log N)$
$O\left(N^2\right)$

e.g.

$$\begin{aligned}
\log\left(n^{100}\right) &- \Theta(\log N) \\
&= 100 \lg(n) \\
\log_5(n) &- \Theta(\log N) \\
&= \frac{\log(n)}{\log 5}
\end{aligned}$$

++ for 2d peak finder, for each column, finding the peak is faster than finding the maximum($\Theta(\log n)$ than $\Theta(n)$). Therefore, it might be better to use a 'wrong' algorithm first?

## *THINKING*

**This recitation class mainly reviewed the asymptotic notation (big O notation) and gave examples of derivation.**

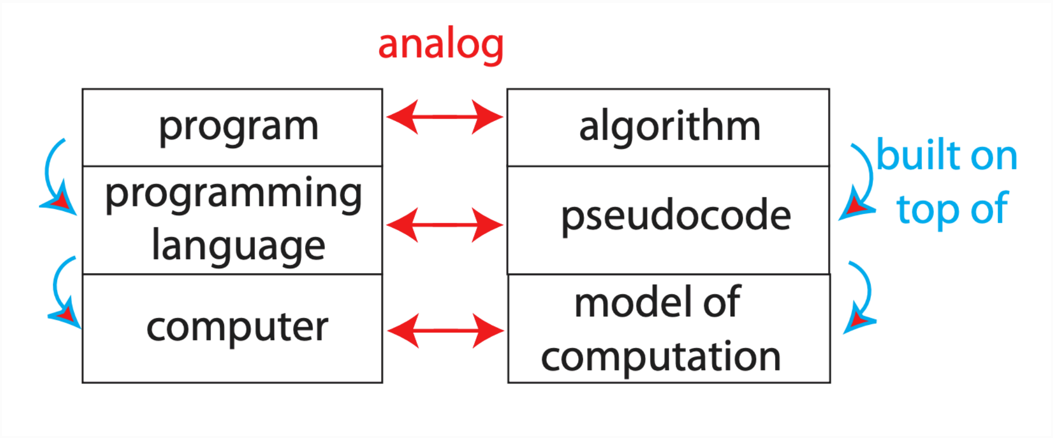
这节复习课主要回顾了渐近符号（大O符号），并举例进行了推导。

# L2. MODELS OF COMPUTATION, PYTHON COST MODEL, DOCUMENT DISTANCE

## WHAT IS AN ALGORITHM?

derivation: Al-Khwārizmi "al-kha-raz-mi" (c. $780 - 850$ )

- Mathematical abstraction of computer program
- Computational procedure to solve a problem



## RANDOM ACCESS MACHINE (RAM) (70S OR 80S)

- Random Access Memory (RAM) modeled by a big array
- $\Theta(1)$ registers (each 1 word )
- In $\Theta(1)$ time, can
    - load word @ $r_i$ into register $r_j$
    - compute $(+, -, *, /, \&, |, ^\wedge\ )$ on registers
    - store register $r_j$ into memory @ $r_i$
- What's a word? $w \geq \lg$ (memory size) bits
    - assume basic objects (e.g., int) fit in word
    - unit 4 in the course deals with big numbers
- realistic and powerful $\rightarrow$ implement abstractions

## POINTER MACHINE (80S)

- dynamically allocated objects (namedtuple)
- object has $O(1)$ fields
- field = word e.g., int $)$ or pointer to object/null (a.k.a. reference)
- weaker than (can be implemented on) RAM

e.g. Linked List

## PYTHON MODEL

*The previous two models: each step takes constant time

Python lets you use either mode of thinking

1. "list" is actually an array $\rightarrow$ RAM $L[i] = L[j] + 5 \rightarrow \Theta(1)$ time
2. object with $O(1)$ attributes (including references) $\rightarrow$ pointer machine
   $x = x$. next $\rightarrow \Theta(1)$ time

Python has many other operations. To determine their cost, imagine implementation in terms of (1) or (2):

1. list
   (a) L.append $(\mathrm{x}) \rightarrow \theta(1)$ time
   obvious if you think of infinite array
   but how would you have $> 1$ on RAM? via table doubling (Lecture 9)
   (b) $\underbrace{L = L1 + L2}_{(\theta(1+|L1|+|L2|)\text{ time })} \quad \equiv L = [] \rightarrow \theta(1)$

(c) $\left.\begin{matrix} \text{L1.extend } (L2) \equiv \text{ for } x \text{ in } L2 : \\ \equiv L1+=L2 \quad \text{L1.append } (\mathrm{x}) \rightarrow \theta(1) \end{matrix}\right\}$ $\theta\left(1+\left|L_2\right|\right)$ time

2. tuple, str: similar, (think of as immutable lists)
3. dict: via hashing (Unit $3 =$ Lectures $8-10]$ time w.h.p. $\mathrm{D}[\mathrm{key}] = \mathrm{val}$
   $\mathrm{key}$ in $\mathrm{D}$
4. set: similar (think of as dict without vals)
5. heapq: heappush & heappop - via heaps (Lecture 4) $\rightarrow \theta(\log(n))$ time
6. long: via Karatsuba algorithm (Lecture 11)

   $x + y \rightarrow O(|x| + |y|)$ time where $|y|$ reflects # words
   $x * y \rightarrow O\left((|x| + |y|)^{\log(3)}\right) \quad \approx O\left((|x| + |y|)^{1.58}\right)$ time

## DOCUMENT DISTANCE PROBLEM

1. split each document into words
2. count word frequencies (document vectors)
3. compute dot product (& divide)

## *THINKING*

**This lecture introduced the definition of some concepts, such as algorithms, runtime, etc., and established the analogy between these abstract concepts and the real world.**

**After that, two abstract calculation models are introduced, random access machine and pointer machine. Since these two models can only perform constant time operations, they can be used as the cornerstone of computational complexity.**

**Therefore, the lecture further introduced the Python model, first introduced the objects corresponding to RAM and pointer machine, and then introduced how other objects can be implemented using these two basic models to get their complexity. These implementation methods will be introduced in subsequent courses.**

**The lecture ended with an example Document Distance Problem, which was further analyzed in the review class.**

这个讲座做了一些概念的定义，如算法、运行时间（runtime）等，并建立了这些抽象概念与现实世界的类比。

之后介绍了两种抽象的计算模型，随机存取机（random access machine）和pointer machine，由于这两种模型只能进行constant time的操作，因此可以作为计算复杂度的基石。

因此讲座进一步介绍了Python模型，首先介绍了对应于RAM和pointer machine的对象，然后介绍了其他对象如何用这两个基础模型来实现，从而得到其复杂度。这些实现方法在之后的课程中都会介绍。

讲座最后以实例Document Distance Problem收尾，并在复习课中进一步展开分析。

# R2. PYTHON COST MODEL, DOCUMENT DISTANCE

## ANALYSIS OF ALGORITHM COMPLEXITY

### *THINKING*

**This recitation analyzes the algorithm's complexity for the first of the 8 algorithms for finding Document Distance. TA considered it to be the core of the entire course.**

**The first is the way of reading the code (Call Graph), then this may also be the order in which the code is written, from front to back, from main to secondary, depending on which the primary function calls other functions. Use indentation or tree diagrams to indicate levels.**

**Then, analyze the code line's complexity by line, delete the low-complexity parts, and calculate the total complexity.**

**There is a complete code handout and note (the note has a complete complexity analysis of 8 algorithms).**

这个复习课对8个寻找Document Distance算法的第一个进行了算法复杂度的分析。助教认为是整个课程的核心。

首先是阅读代码的方式（函式呼叫图Call Graph），那么这或许也应该是书写代码的顺序，从前到后，从主要到次要，看首要函数调用了哪些其他的函数。用缩进或是树图来表示层级。

然后对于代码逐行分析其复杂度，删除复杂度低的部分，从而计算总的复杂度。

有完整的代码的handout和note（note中有8个算法的完整的复杂度分析）。

## PS1

# PEAK-FINDING PROOF

We wish to show that algorithm1 will always return a peak, as long as the problem is not empty. To that end, we wish to prove the following two statements:

1. If the peak problem is not empty, then **algorithm1** will always return a location.
2. If **algorithm1** returns a location, it will be a peak in the original problem.

## *THINKING*

**Problem Set1 mainly examines the understanding of asymptotic complexity. Four local peak search algorithms are used to investigate the calculation of the complexity of the algorithm and the proof of the correctness.**

Problem Set1 主要考察了对于渐进复杂度的理解。用四个局部峰值查找算法考察了对于算法复杂度的计算和算法有效性的证明。