

# 数据结构与算法

## DATA STRUCTURE

— 用面向对象方法与C++描述

信息管理与工程学院  
2018 - 2019 第一学期

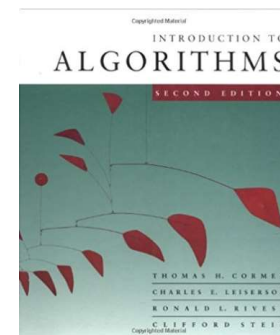
# 课程信息

- 助教：孙晨曦 [sufescx@163.com](mailto:sufescx@163.com)
- 评分标准：
  - 期末闭卷笔试（约70%左右）
  - 作业(功能实现，易读性，性能测试)（20%）
  - 出勤和课堂讨论（10%）
- 教师：胡浩栋
- 答疑时间：星期二下午3：30 - 5：30
- 答疑地点：信管学院606

## 参考教材

- 数据结构教程

复旦大学出版社  
施伯乐



- Introduction to Algorithms,  
3rd Edition,  
MIT Press

# 课程内容

- 复杂度分析和C++要点回顾
- 数组(Array)
- 链表(LinkedList)
- 字符串(String)和匹配算法
- 栈(Stack)
- 队列(Queue)
- 递归(Recursion)
- 树(Tree)
- 查找和索引(Search, Hashing)
- 图(Graph)
- 多种内部排序算法

# 课程目的

- 理解，正确使用在各种数据结构上的操作
- 分析代码的复杂度，来评估实现的优劣
- 只有自己制造过的轮子，才是自己的轮子

# 基本概念：

- **数据 (Data):**

客观事物抽象到计算机，能被计算机识别的集合

- **数据元素 (Data Element)**

数据的基本单位，一般当不可分割部分来操作

- **数据操作**

对数据元素之间按逻辑关系进行运算的一次操作

# 什么是数据结构

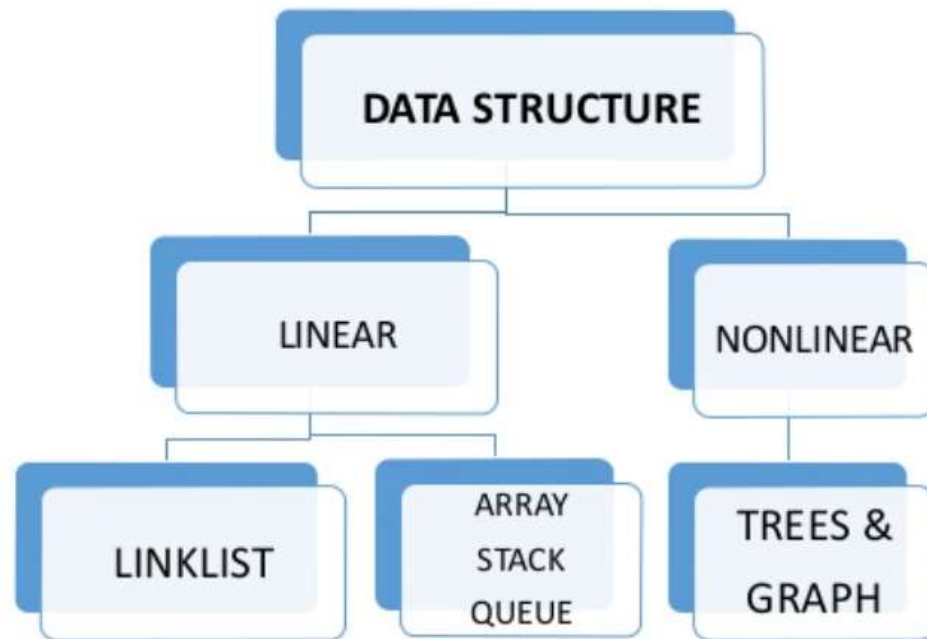
能有效地**存储**和**操作**在计算机里的**数据元素**

- 既是一种对数据元素存储方式的管理
- 也是一种方法，能对存储的数据元素进行有效的操作



# 数据结构例子

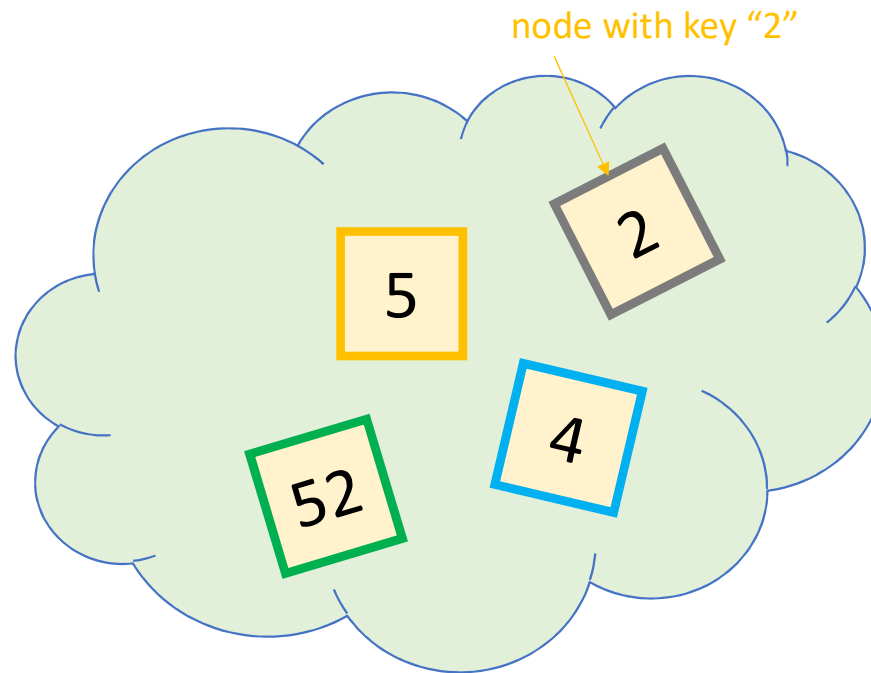
- 基本的数据结构
  - 数组(Array)
  - 链表(LinkList)
- 衍生的数据结构
  - 栈(Stack)
  - 队列(Queue)
  - 树(Tree)
  - 图(Graph)
  - 堆(Heap)
  - etc





# 一种问题原型

比如需要维护亿万位的整数数据（身份证号码），使得能快速查找，快速删除和快速插入



# “有效”的标准

解法所需的资源利用情况

- 时间复杂度
- 空间复杂度



# 复杂度描述：Big-O术语

- $f(n) = O(g(n))$  指的是函数 $f(n)$ 增长的上界
- 例子

$$n + 12 = O(n)$$

$$n^2 + 3n - 2 = O(n^2)$$

$$n^3 + 10n^2 \log(n) - 15n = O(n^3)$$

$$2^n + n^2 = O(2^n)$$

# 正式定义

- Let  $f(\cdot), g(\cdot): N \rightarrow N$ ,

Then  $f(n) = O(g(n))$  if and only if

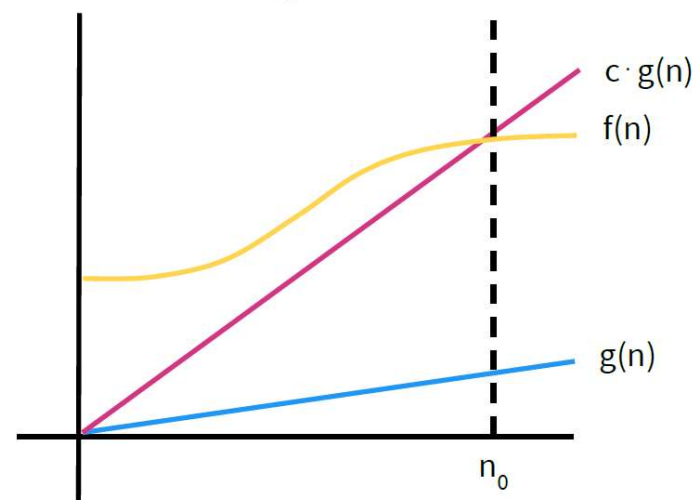
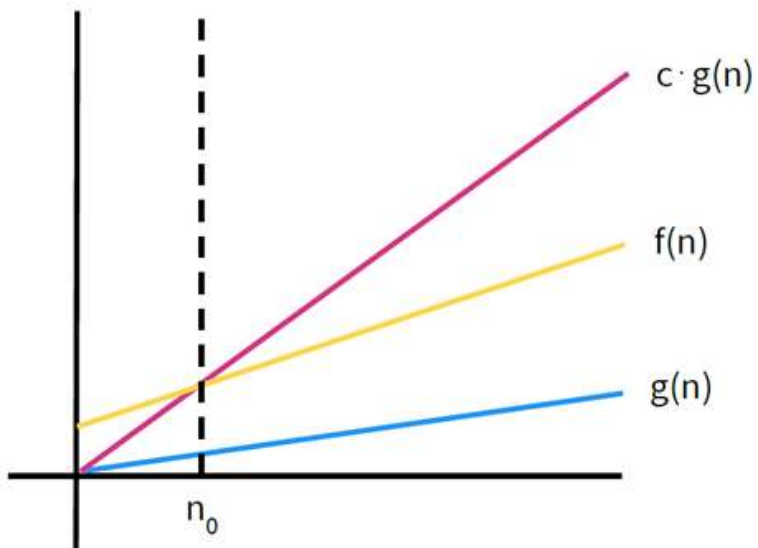
$$\exists n_0 \in N, c \in R, \forall n \in N, n \geq n_0 \rightarrow f(n) \leq c \cdot g(n)$$

也就是说 $f(n)$ 最多是和 $g(n)$ 一样大，除了一个常数倍数 $c$ 。

# Big- O术语

$f(n) = \mathcal{O}(g(n))$  iff

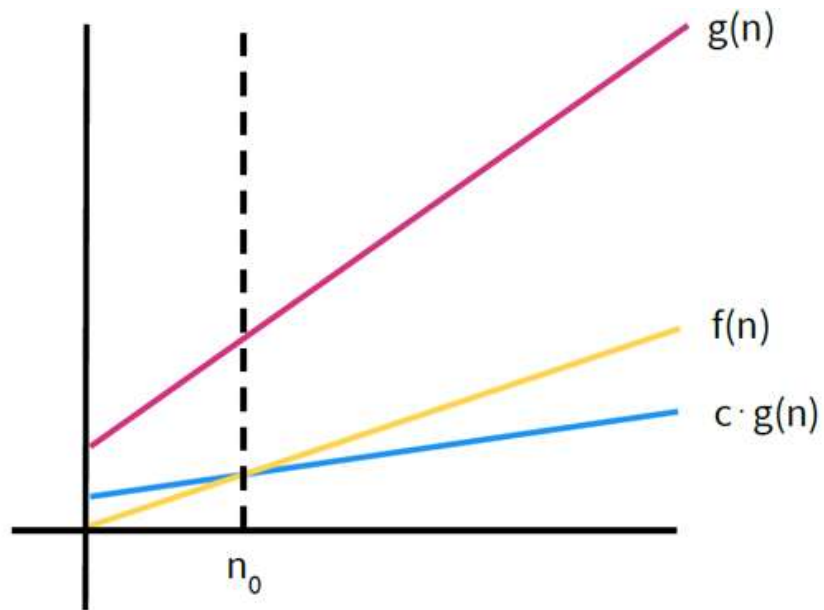
$$\exists n_0 \in \mathbf{N}, c \in \mathbf{R}, \forall n \in \mathbf{N}, n \geq n_0 \rightarrow f(n) \leq c \cdot g(n)$$



# Big- $\Omega$ 术语

$$f(n) = \Omega(g(n)) \text{ iff}$$

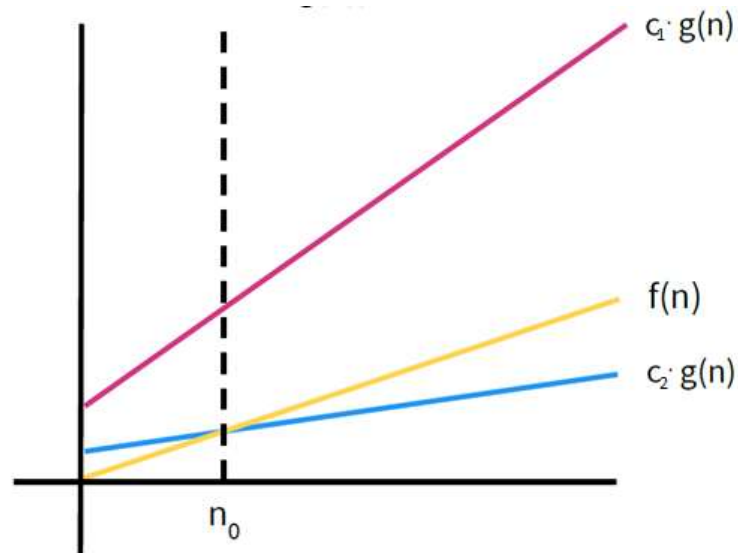
$$\exists n_0 \in \mathbf{N}, c \in \mathbf{R}, \forall n \in \mathbf{N}, n \geq n_0 \rightarrow f(n) \geq c \cdot g(n)$$



# Big- $\Theta$ 术语

$f(n) = \Theta(g(n))$  iff

$\exists n_0 \in \mathbf{N}, c \in \mathbf{R}, \forall n \in \mathbf{N}, n \geq n_0 \rightarrow f(n) \leq c_1 \cdot g(n) \text{ and } f(n) \geq c_2 \cdot g(n)$



## 复杂度例子：排序问题

- 给定一个无序元素数组Array  $\{a_1, a_2, \dots, a_n\}$ , 要求比较排序成  $\{a_{i1}, a_{i2}, \dots, a_{in}\}$ , 使得  $a_{i1} < a_{i2} < \dots < a_{in}$ 。

4	8	1	5	3	2	6	7
---	---	---	---	---	---	---	---



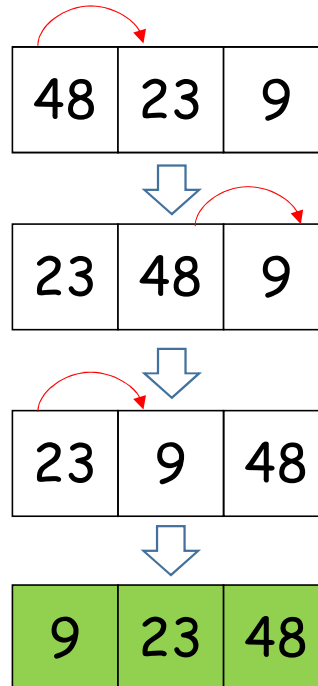
# 基于比较(Comparison)的排序

可比较的数学定义:

- $\forall a, b, a > b \text{ or } a = b \text{ or } a < b$
- If  $a > b, b > c$ , then  $a > c$

# 冒泡排序 BubbleSort

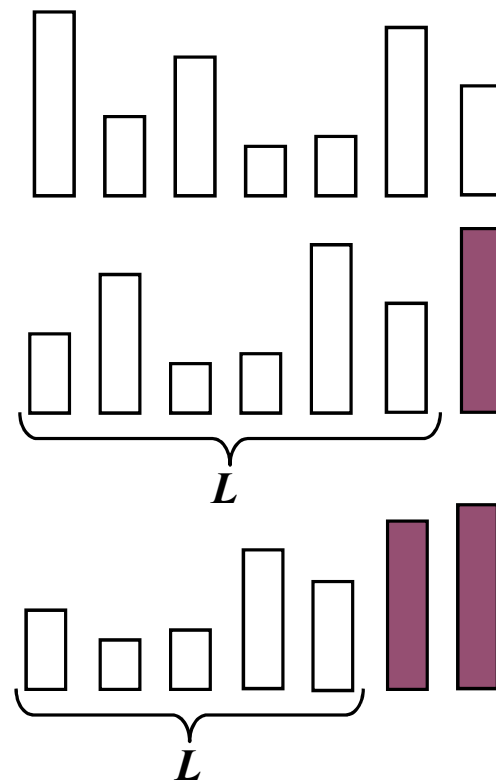
- **Naive:** 比较相邻的元素，如果第一个比第二个大，就交换他们两个。
- **Idea:** 通过两两比较交换，像水中的泡泡一样，大的先冒出来，小的后冒出来。



# BubbleSort复杂度

$$\begin{aligned}T(n) &= T(n-1) + (n-1) \\&= T(n-2) + (n-2) + (n-1) \\&= \dots \\&= 1 + \dots + (n-2) + (n-1) \\&= O(n^2)\end{aligned}$$

- 复杂度高，实际中没人用
- 名字取得好
- 适合入门算法

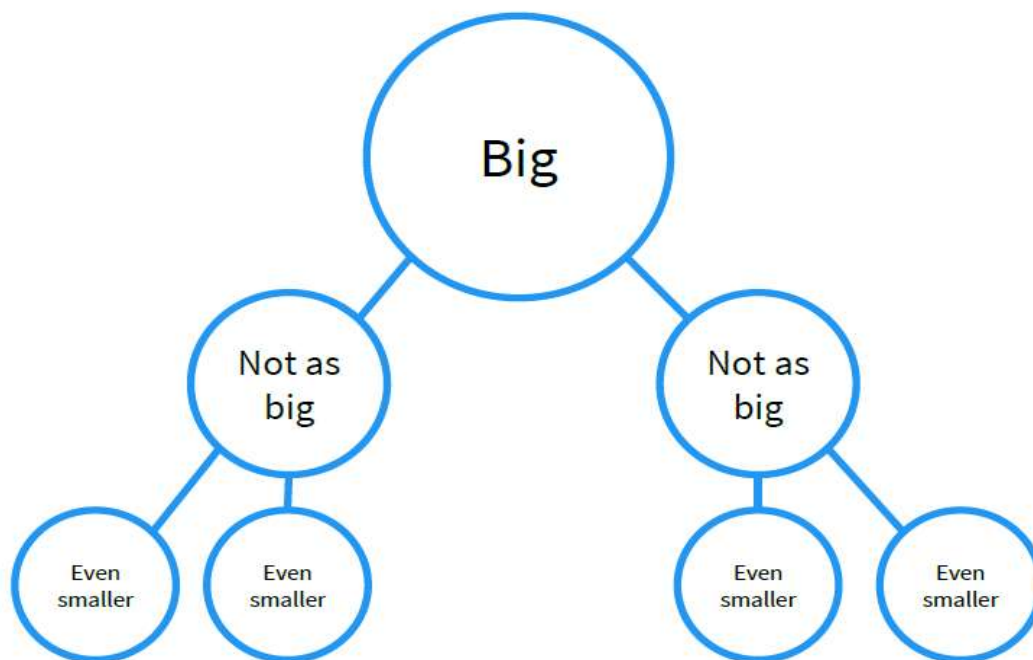


如何改进？

# Idea: 分治法 Divide & Conquer

- 基本思想是：
  - 将原问题分解为若干个规模更小但结构与原问题相似的子问题。
  - 然后递归地解这些子问题。
  - 最后将这些子问题的解组合为原问题的解。
- 计算机基础算法三大思想之一

# 分治法 – 从下往上的方法



改进的方法： **MergeSort** 采用了一种分治的策略。

# 归并排序 (MergeSort)

对长度为8的数组归并排序

- 分别对A[0,3]和A[4,7]排序
- 然后把两个排好序的半段归并

4	8	1	5	3	2	6	7
---	---	---	---	---	---	---	---

1	4	5	8	2	3	6	7
---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Index: 0 1 2 3 4 5 6 7

# 1 递归求解子问题

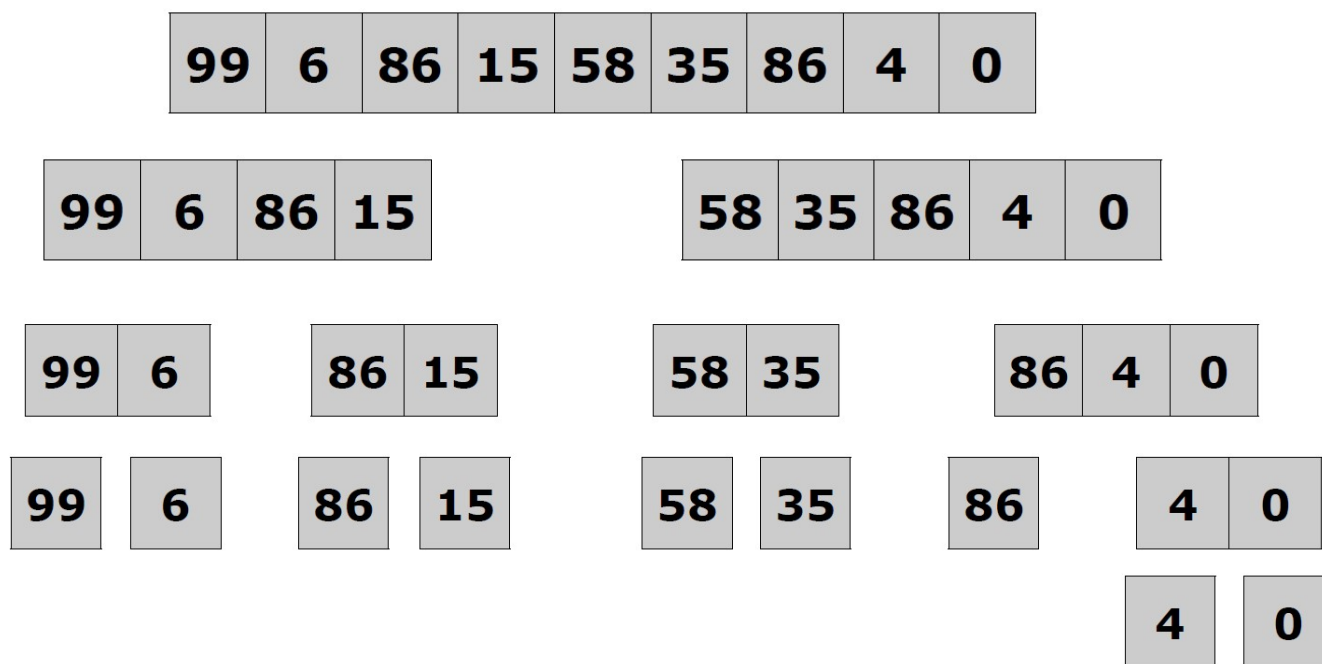
99	6	86	15	58	35	86	4	0
----	---	----	----	----	----	----	---	---

99	6	86	15
----	---	----	----

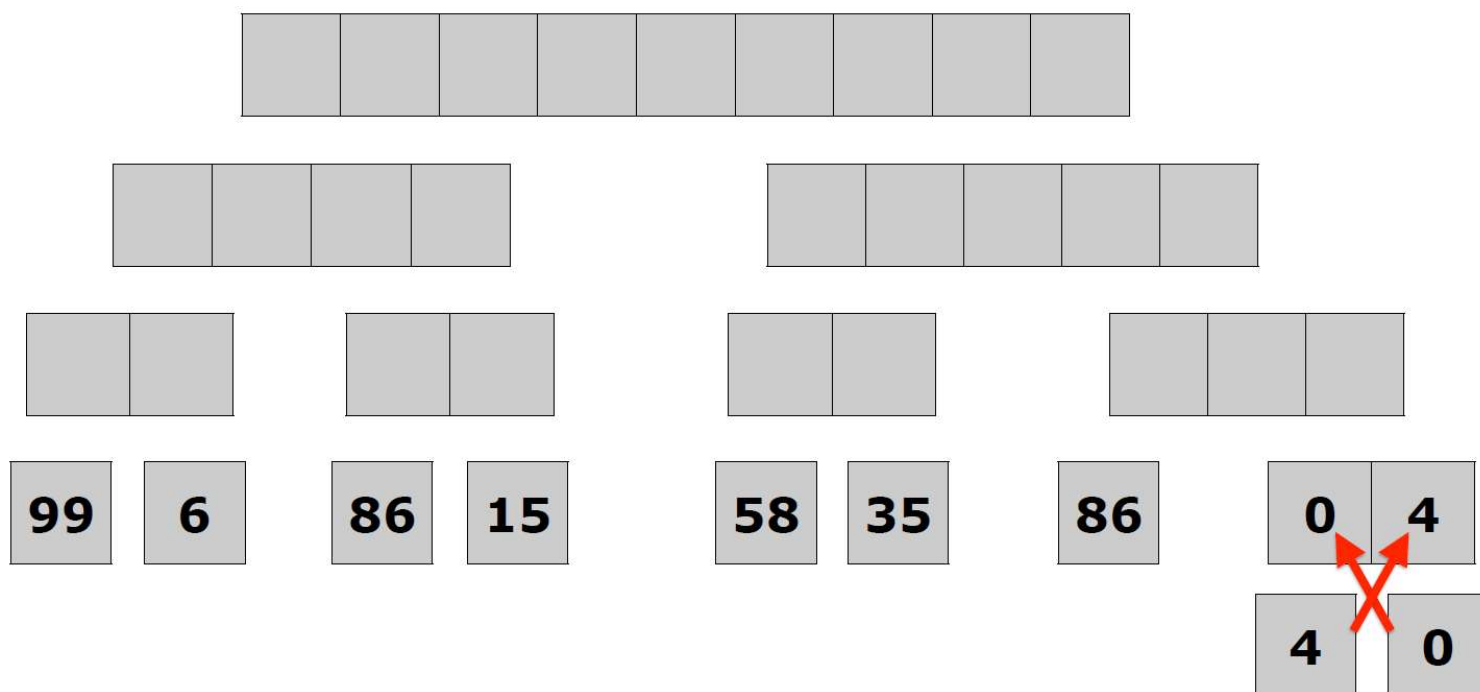
58	35	86	4	0
----	----	----	---	---



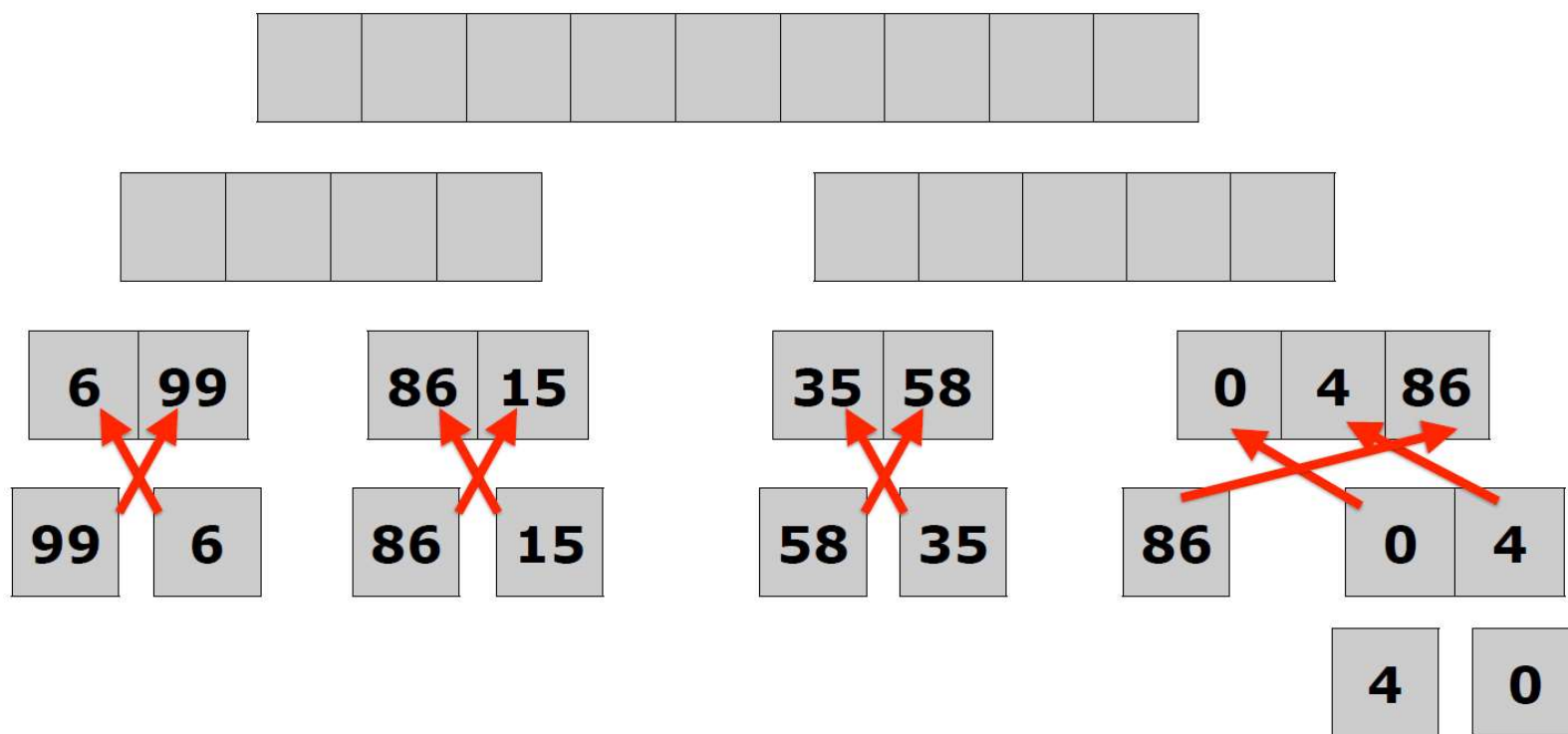
## 2 直到最小子问题（直接得到解）



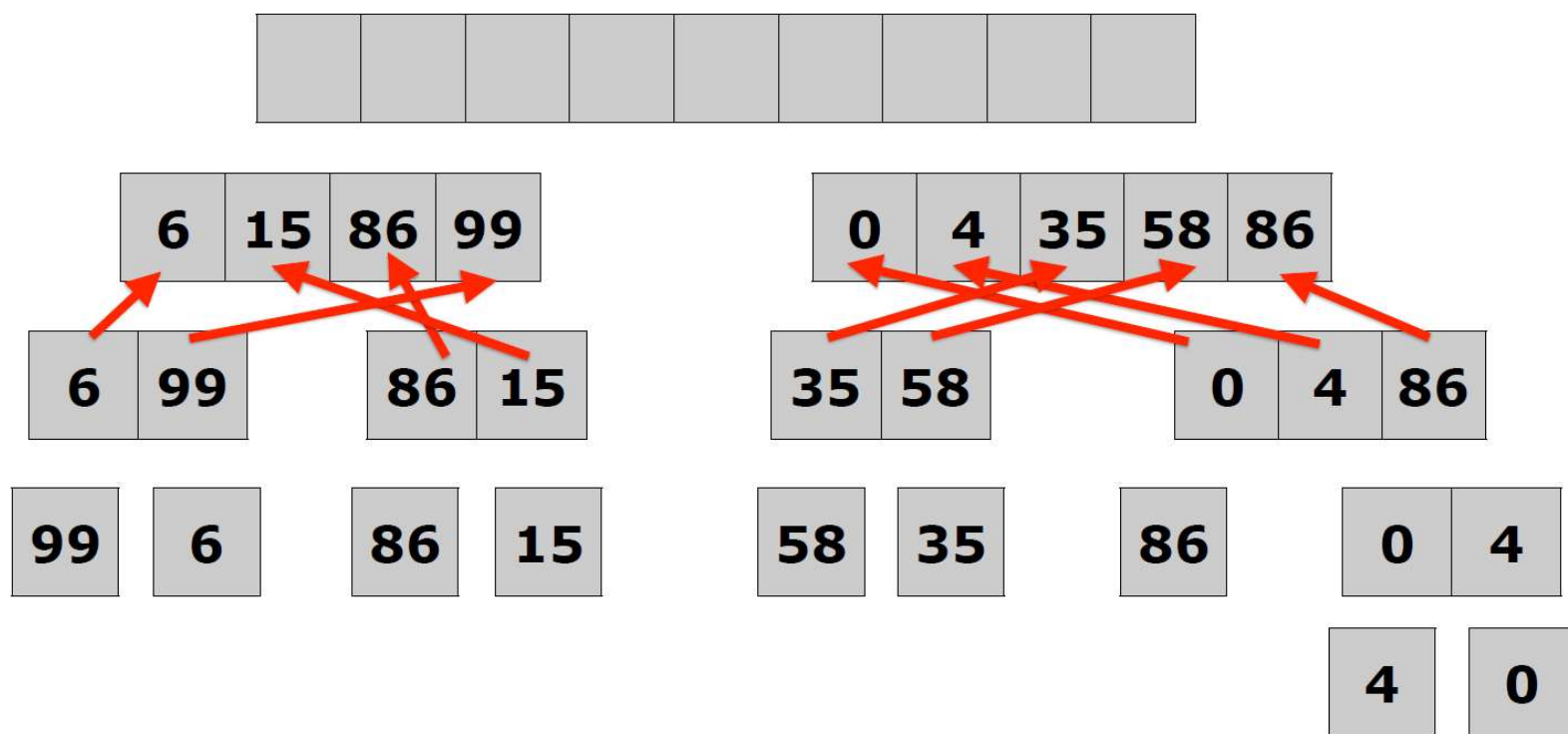
### 3 从下往上排序



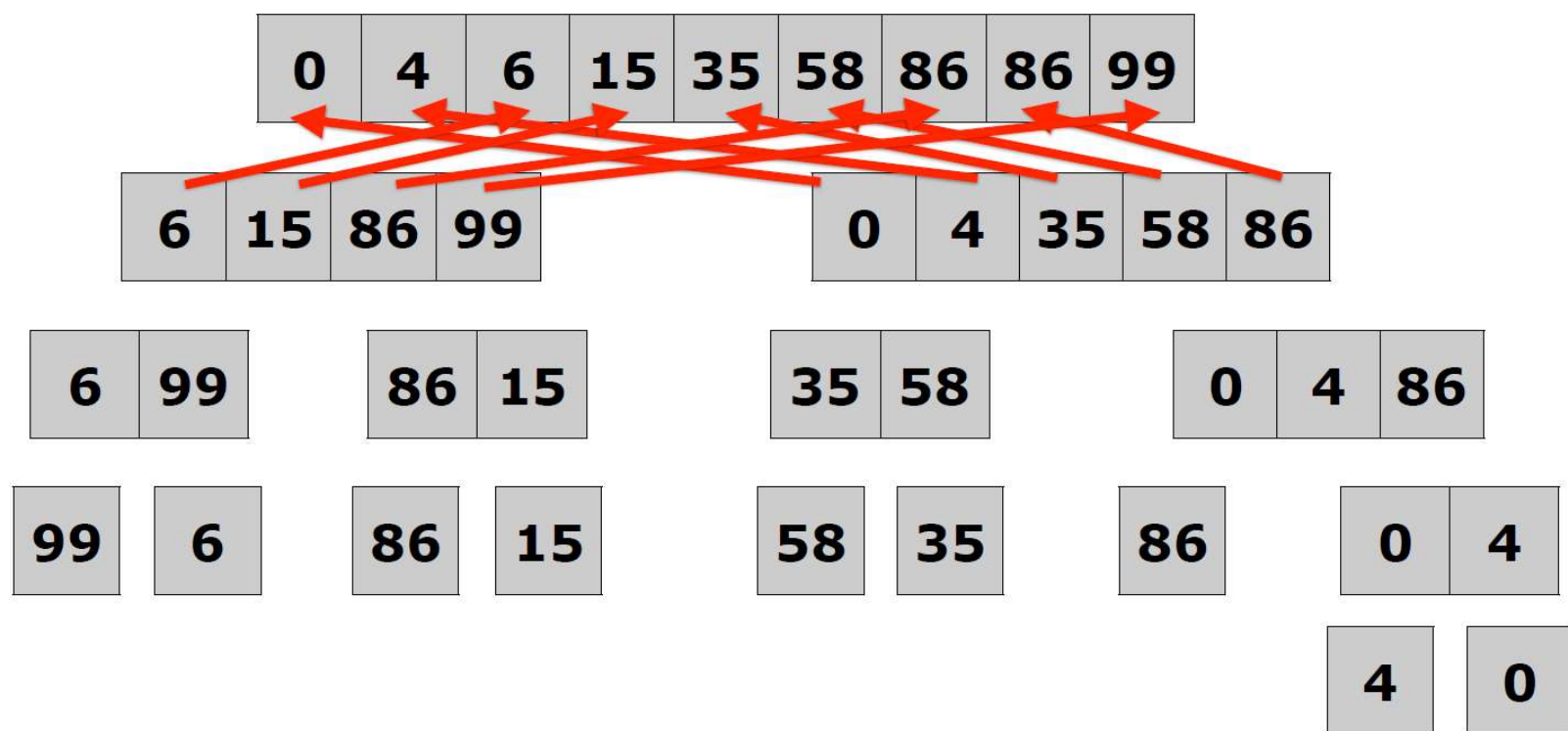
## 4 继续合并子问题



## 5 继续合并子问题



## 6 最后合并成原问题的解



## 归并算法（递归实现）

```
algorithm mergesort(list A):  
  if length(A) ≤ 1:  
    return A  
  let left = first half of A  
  let right = second half of A  
  return merge(  
    mergesort(left),  
    mergesort(right)  
  )
```

## 归并算法 (cont)

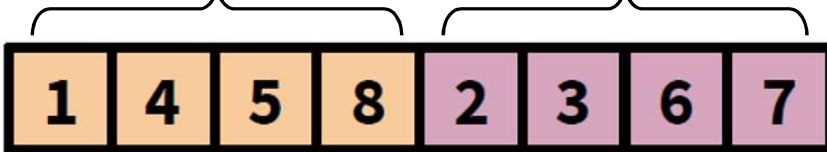
```
algorithm merge(list A, list B):  
  let result = []  
  while both A and B are nonempty:  
    if head(A) < head(B):  
      append head(A) to result  
      pop head(A) from A  
    else:  
      append head(B) to result  
      pop head(B) from B  
  append remaining elements in A to result  
  append remaining elements in B to result  
  return result
```

**Total work:**  $O(a+b)$ , where  $a$  and  $b$  are the lengths of lists  $A$  and  $B$ .

# 归并算法

```
void MergeSort(int arr[], int start, int end)
{
    if (start >= end)
    {
        return;
    }

    int mid = start + (end - start) / 2;
    MergeSort(arr, start, mid);
    MergeSort(arr, mid+1, end);
    Merge(arr, start, mid+1, end);
}
```



```
void Merge(int arr[], int start, int mid, int end)
{
    int lbuff[mid-start+1]{};
    for (int i=0; i < mid - start; i++)
    {
        lbuff[i] = arr[start + i];
    }

    lbuff[mid-start] = numeric_limits<int>::max();

    int rbuff[end-mid+2]{};
    for (int i=0; i<end-mid+1; i++)
    {
        rbuff[i] = arr[mid+i];
    }

    rbuff[end-mid+1] = numeric_limits<int>::max();

    // Have two pointer to left array and right array.
    // move the smaller number of both arrays to merged array.
    for (int k=start, i=0, j=0; k<=end; k++)
    {
        if (lbuff[i] < rbuff[j])
        {
            arr[k] = lbuff[i];
            i++;
        }
        else
        {
            arr[k] = rbuff[j];
            j++;
        }
    }
}
```



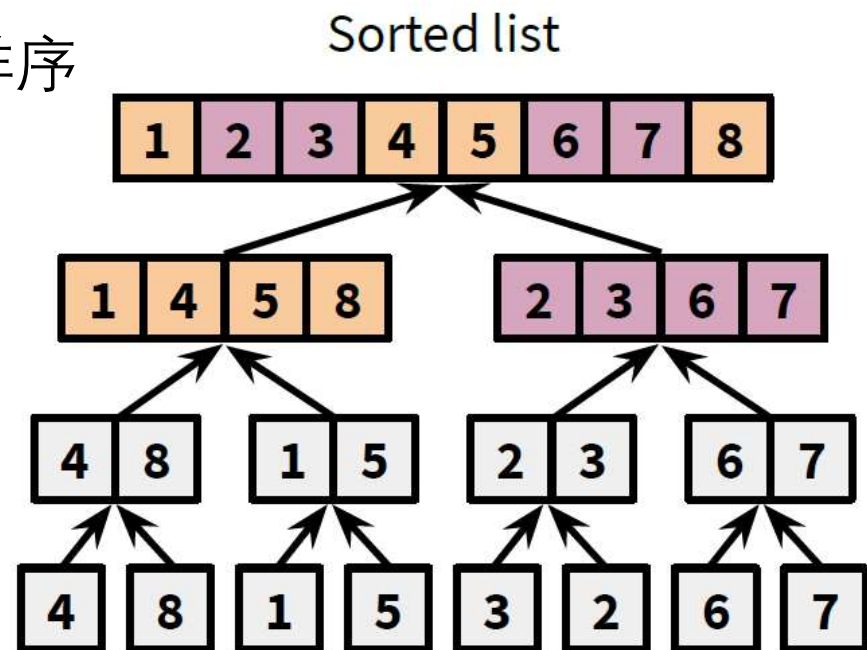
# Mergesort复杂度分析

- $T(n)$ 是用对长度为 $n$ 的数组归并排序

- 分别对 $A[0,3]$ 和 $A[4,7]$ 排序
- 复杂度 $2T(n/2)$
- 然后把两个排好序的半段归并
- 复杂度 $2n$

- $T(1) = 1$

- $T(n) = 2T(n/2) + 2n$

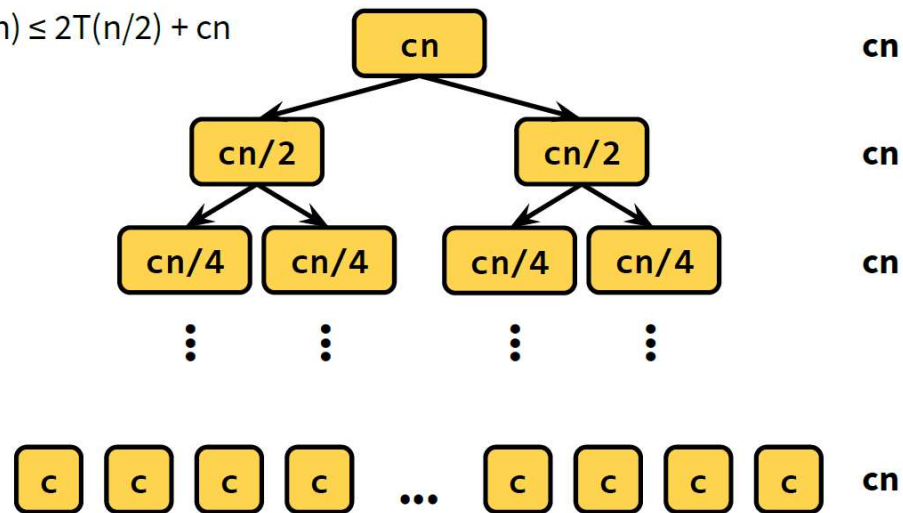


# 递归求解

## Recursion Tree Method

$$T(1) \leq c$$

$$T(n) \leq 2T(n/2) + cn$$

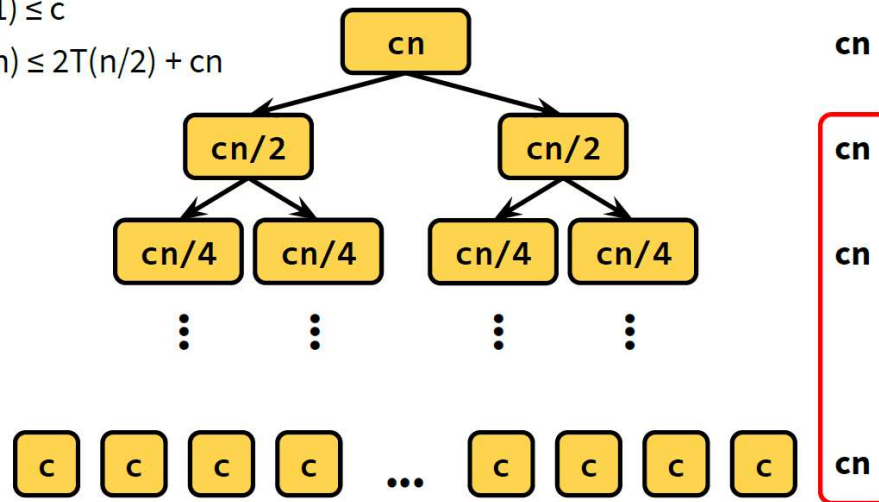


# 递归求解

## Recursion Tree Method

$$T(1) \leq c$$

$$T(n) \leq 2T(n/2) + cn$$



**Total work:**  $cn \log_2 n + cn$

# 复杂度

- 时间复杂度

$$\begin{aligned}T(n) &= 2T(n/2) + 2n \\&= 2^2T(n/2^2) + 2(n/2) + 2n \\&= \dots \\&= 2^{\log n}(n/2^{\log n}) + \dots + 2(n/2) + 2n \\&= n \times (\log n + 2) = O(n \log n)\end{aligned}$$

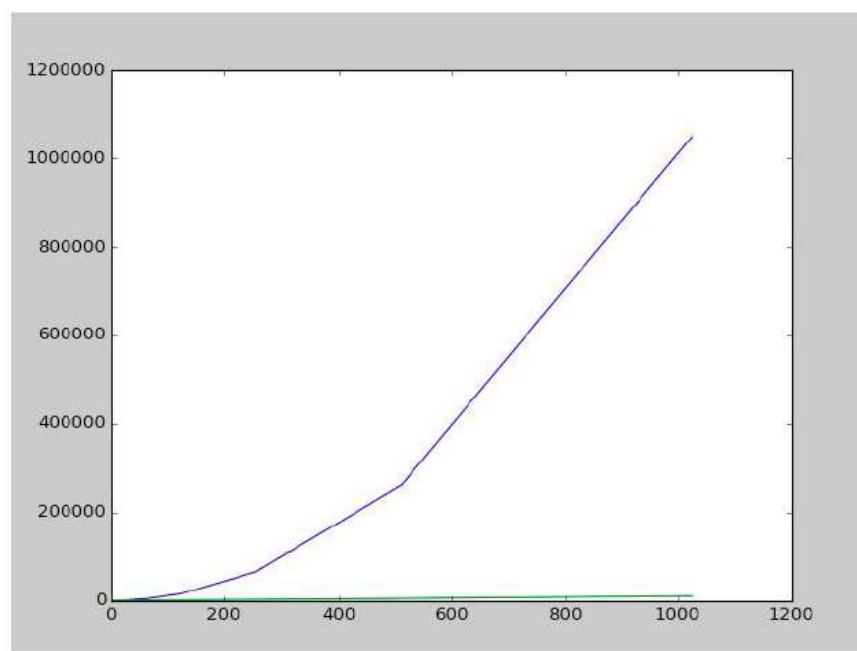
- 空间复杂度

$O(n)$ , 需要额外数组来实现合并

- **思考题**: 能不能用上述归并排序代码测试百万数据集?

# 冒泡排序 *vs* 归并排序

- $O(n^2)$  *vs*  $O(n \log n)$



# Master Theorem

If  $T(n) = a T\left(\frac{n}{b}\right) + f(n)$ , then

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a} \log n) & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ AND } af(n/b) < cf(n) \text{ for large } n \end{cases} \left\{ \begin{array}{l} \varepsilon > 0 \\ c < 1 \end{array} \right.$$

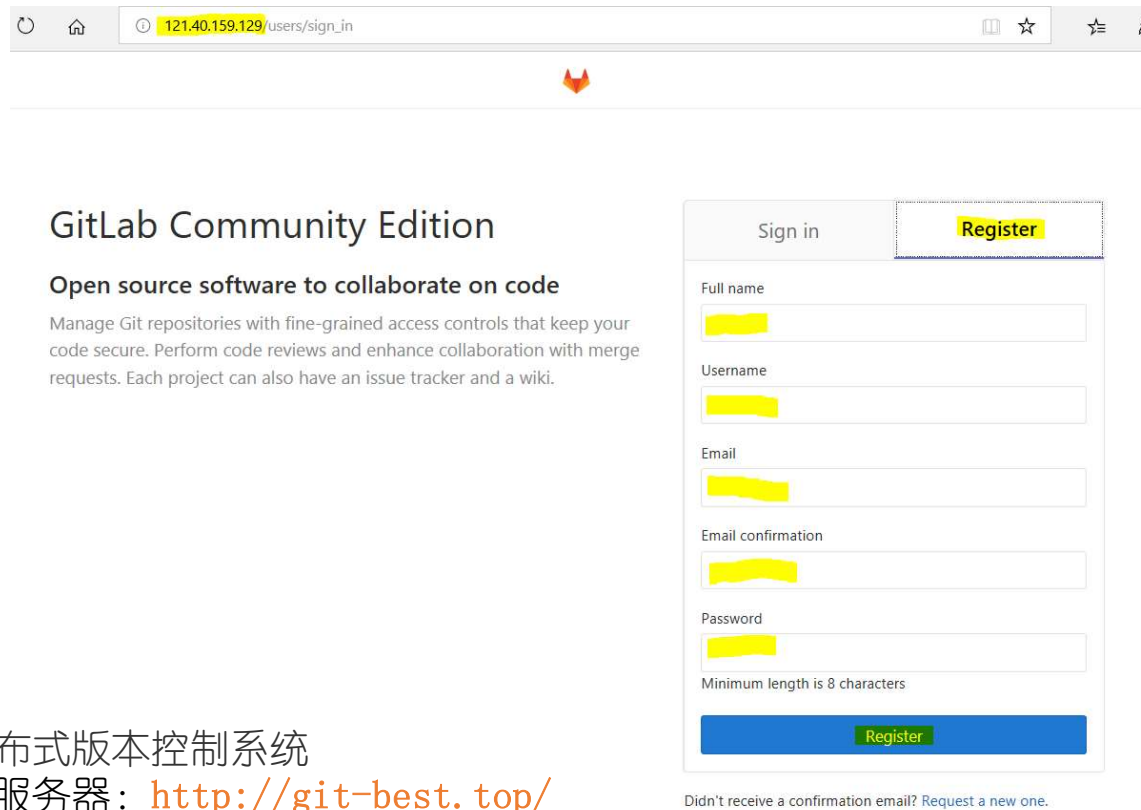
- $T(n) = 2T(n/2) + \Theta(n)$

$$\begin{array}{l} a = 2 \\ b = 2 \\ d = 1 \end{array}$$

$$\begin{aligned} T(n) &= O(n^d \log n) \\ &= O(n \log n) \end{aligned}$$

If  $f(n) = \Theta(n^{\log_b a} \log n)$ , then  $T(n) = \Theta(n^{\log_b a} \log^2 n)$

# GIT onboard



The screenshot shows the GitLab Community Edition sign-in and registration page. The browser address bar displays the URL `121.40.159.129/users/sign_in`. The page features the GitLab logo at the top center. Below the logo, the heading "GitLab Community Edition" is followed by the tagline "Open source software to collaborate on code". A descriptive paragraph states: "Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki." On the right side, there is a registration form with two tabs: "Sign in" and "Register". The "Register" tab is active. The form includes input fields for "Full name", "Username", "Email", "Email confirmation", and "Password". A note below the password field indicates "Minimum length is 8 characters". A blue "Register" button is at the bottom of the form. Below the button, a link reads "Didn't receive a confirmation email? Request a new one."

GitLab Community Edition

**Open source software to collaborate on code**

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Sign in Register

Full name

Username

Email

Email confirmation

Password

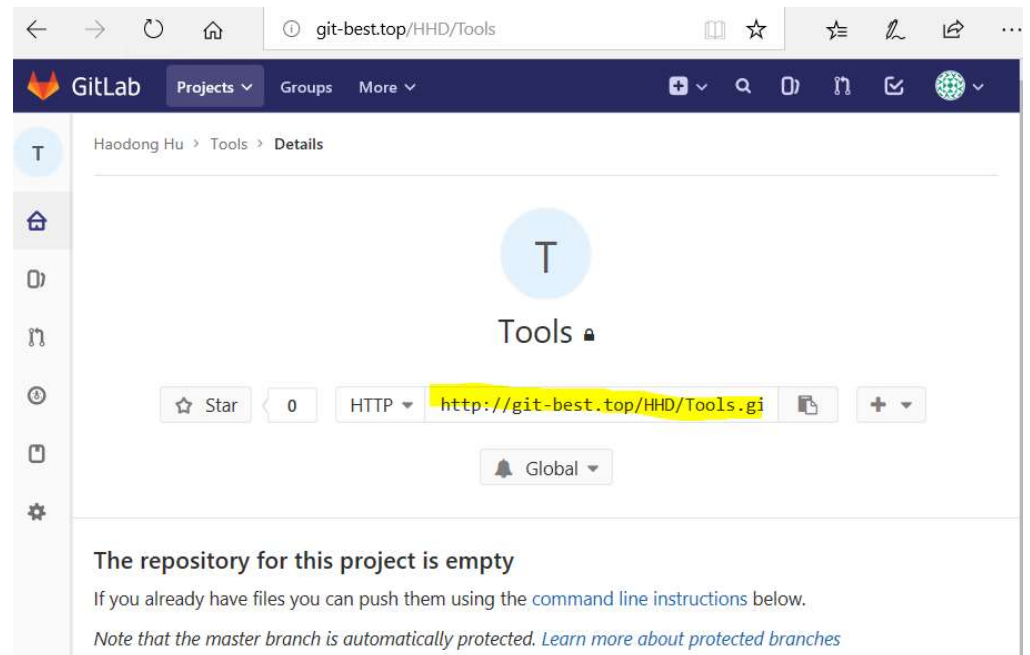
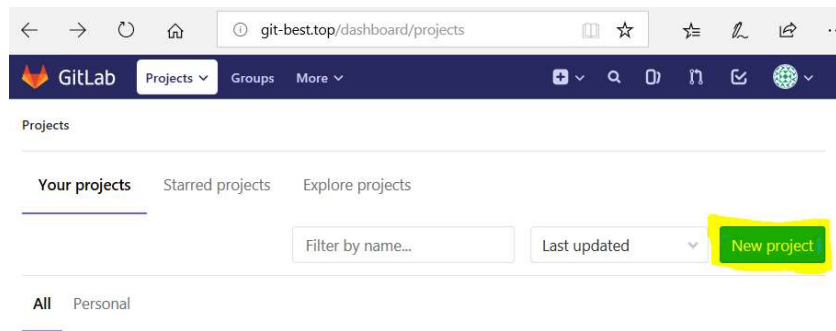
Minimum length is 8 characters

Register

Didn't receive a confirmation email? [Request a new one.](#)

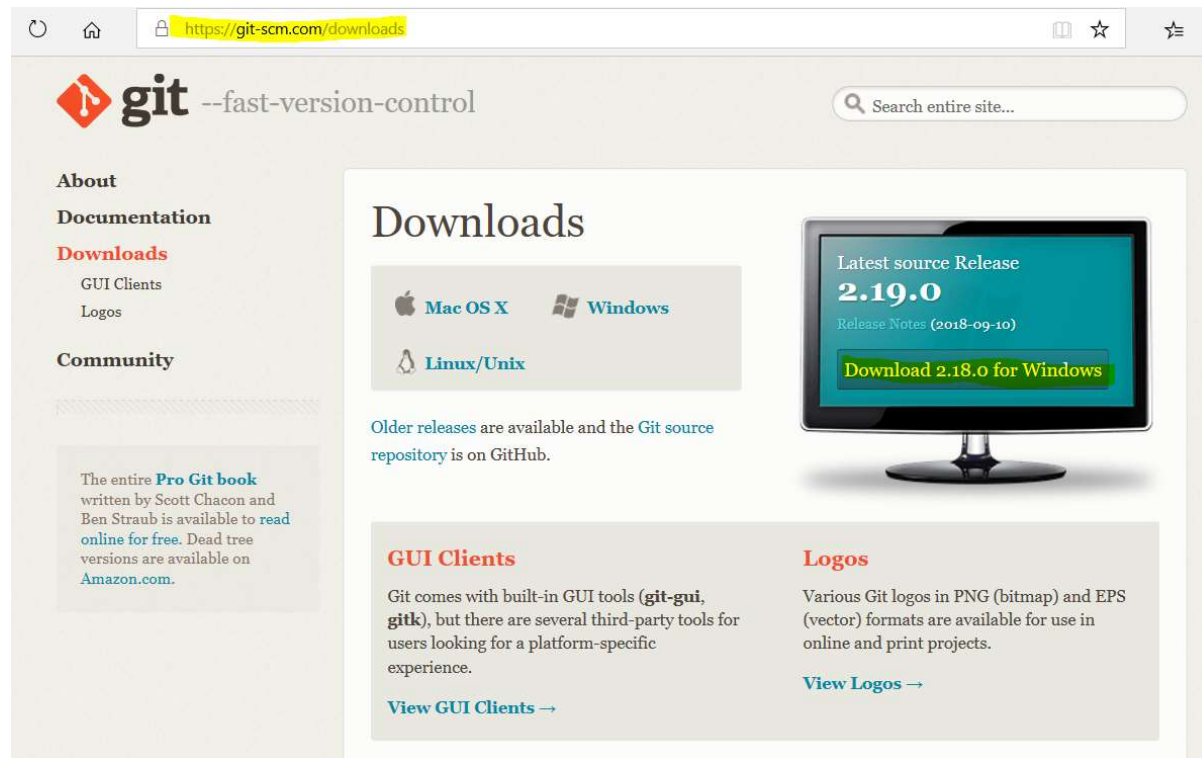
GIT: 开源的分布式版本控制系统  
学校无限制GIT服务器: <http://git-best.top/>

# GIT – 服务器端设置新project





# GIT – 下载客户端



教程：<http://rogerdudler.github.io/git-guide/>

# GIT – cmd窗口命令行（同步服务器）

- **Git global setup**

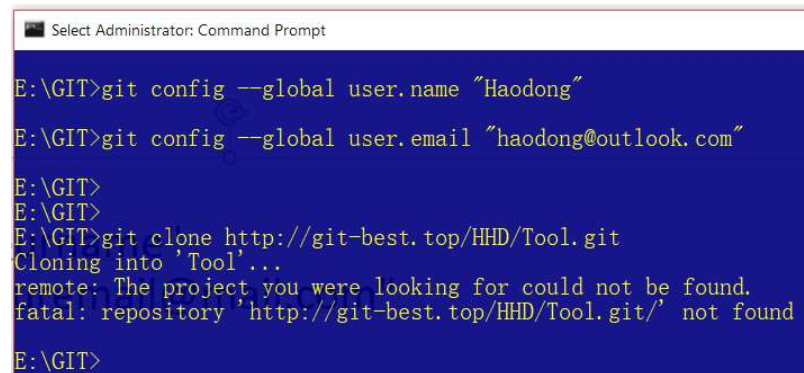
- git config --global user.name "yourname"
- git config --global user.email "youremail@mail.com"

- **Create a new repository**

- git clone http://git-best.top/UserName/Tools.git
- cd Tools
- touch README.md
- git add README.md
- git commit -m "add README"
- git push -u origin master

- **Existing folder**

- cd Tools
- git add .
- git commit -m "xxx message"
- git push -u origin master



```
Select Administrator: Command Prompt

E:\GIT>git config --global user.name "Haodong"
E:\GIT>git config --global user.email "haodong@outlook.com"
E:\GIT>
E:\GIT>
E:\GIT>git clone http://git-best.top/HHD/Tool.git
Cloning into 'Tool'...
remote: The project you were looking for could not be found.
fatal: repository 'http://git-best.top/HHD/Tool.git/' not found
E:\GIT>
```

# Q&A

下一讲: C++高级知识点回顾

# Thanks!