算法设计与分析

Lecture 1: Introduction

卢杨

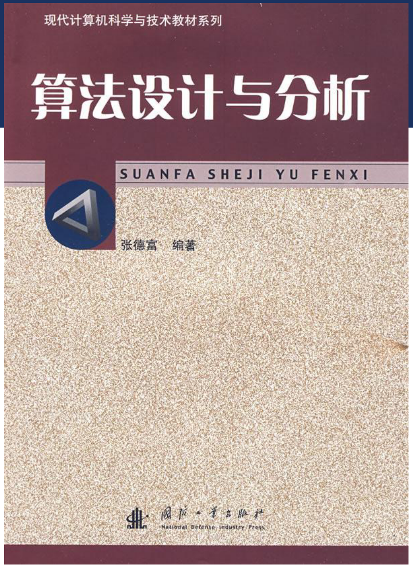
厦门大学信息学院计算机科学系

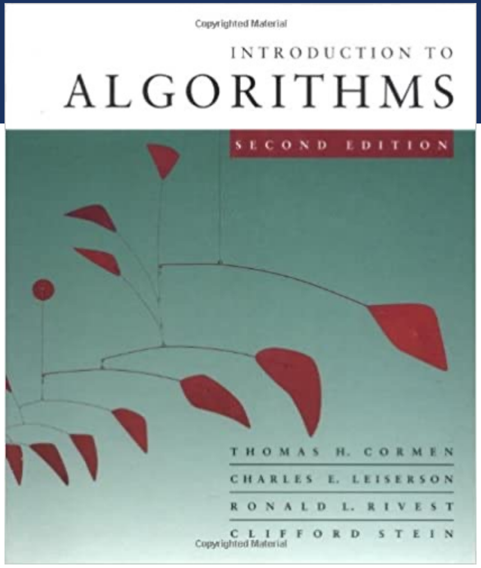
luyang@xmu.edu.cn

课程简介

- “算法设计与分析”是计算机科学与技术专业一门重点专业基础课程，也是学科核心专业基础课程。本课程主要介绍算法设计与分析的基本方法以及算法复杂性理论基础。
- 通过本课程的学习，要求学生理解并熟练掌握递归与分治法、贪心法、动态规划方法、回溯法、分支定界法，以及高级图论算法、线性规划算法等，理解并掌握算法复杂性的分析方法、NP完全性理论基础。
- 通过教学和实践，培养学生从算法的角度运用数学工具分析问题和解决问题的基本能力，从而使他们能够正确地分析和评价一个算法，进一步设计出真正有效的算法。
- 此外，配合实验课程的教学，学生应理论联系实际，理论指导实践，通过规范地完成一系列算法设计实验进一步巩固所学的相关书本知识，在知识、能力、素质上得到进一步的提高。

教材及参考书





- 教材：
  - 张德富, 算法设计与分析, 国防工业出版社, 2009.
- 参考书：
  - T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms (the second edition), The MIT Press, 2001. 算法导论（第二版）（英文影印版）, 高等教育出版社, 2003.
  - M. H. Alsuwaiyel， Algorithms Design Techniques and Analysis， World Scientific Publishing Company, 1998. 吴伟昶等译, 算法设计技巧与分析（中文版）, 电子工业出版社, 2004.

Image source: http://www.bookschina.com/4203199.htm
https://www.amazon.com/Introduction-Algorithms-Second-Thomas-Cormen/dp/0070131511

考核方式

- 成绩构成 (暂定):
  - 期末笔试50%
  - 实验上机考试15%
  - 实验大作业20% (选最高分的5次, 代码+报告)
  - 平时成绩15% (作业+出勤+课堂表现)

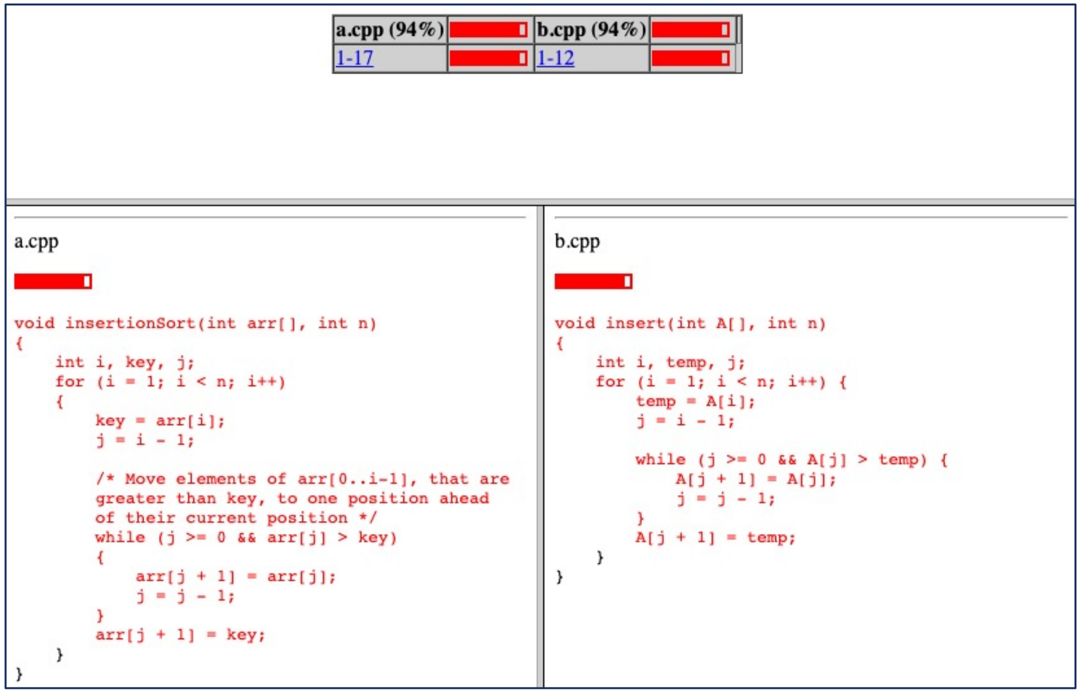
厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

课程资料

- 课件, 资料, 作业发布与递交均在厦门大学SPOC系统上进行.
  - 课程主页: https://l.xmu.edu.cn/course/view.php?id=6174
  - 选课密码: finch5deer
- 课件以英文为主, 授课语言为中文
- 作业使用中英文均可, 并在SPOC上递交电子版.

代码抄袭

- 所有实验大作业将**严格施行代码查重**
  - 第一次警告
  - 第二次该次作业0分
  - 第三次…



a.cpp (94%)	🟥	b.cpp (94%)	🟥
1-17	🟥	1-12	🟥

a.cpp

```
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
        greater than key, to one position ahead
        of their current position */
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

b.cpp

```
void insert(int A[], int n)
{
    int i, temp, j;
    for (i = 1; i < n; i++) {
        temp = A[i];
        j = i - 1;

        while (j >= 0 && A[j] > temp) {
            A[j + 1] = A[j];
            j = j - 1;
        }
        A[j + 1] = temp;
    }
}
```

非常严格的代码查重系统

Why Study Algorithms?

- Internet
  - Web search, packet routing, distributed file sharing, ...
- Artificial Intelligence
  - Autopilot vehicle, computer vision, machine translations, ...
- Computers
  - Circuit layout, file system, compilers, ...
- Computer graphics
  - Movies, video games, virtual reality, ...
- Security
  - Cell phones, e-commerce, voting machines, ...
- Multimedia
  - MP3, JPG, DivX, HDTV, face recognition, ...
- Social networks
  - Recommendations, news feeds, advertisements, ...
- Physics
  - Aerodynamics simulation, particle collision simulation, ...
- Biology
  - Human genome project, protein folding, ...
- ⋮

Why Study Algorithms?

- Algorithm engineer is rich!

Job title	Average salary per year in US (2020)
Algorithm Engineer	$135,272
Database Engineer	$119,281
Software Engineer	$108,058
System Engineer	$102,266
Network Engineer	$96,220
IT Project Manager	$95,712
Web Developer	$75,671

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

Data source: https://www.indeed.com/salaries?from=gnav-title-webapp

What is An Algorithm

- Example: Find the name "Lisa Barber" in the phone book.
  - Strategy 1
    - Starting with the first name "Anderson Aaron".
    - If it is not matched, check the next one.
    - Until find "Lisa Barber".
  - Strategy 2
    - Slide your finger along the Alphabet bar to "B".
    - Check the current surname is before or after "Ba" in alphabetical order.
    - Find surname "Barber".
    - Look for the first name "Lisa".

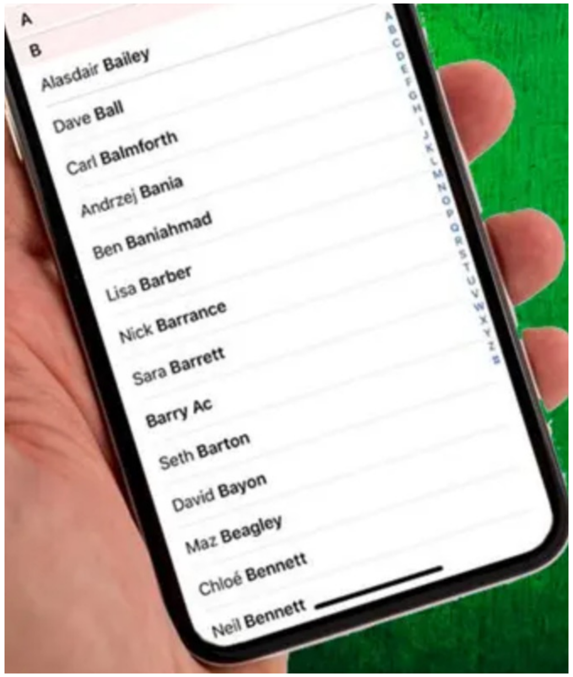


Image source: https://messagingapplab.com/wp-content/uploads/2019/04/Knowing-Who-Has-My-Number-in-Their-Contact-List.jpg.webp

What is An Algorithm

- An algorithm is a sequence of computational steps that transform the input into the output to solve a given problem.
- Example: The sorting problem.
  - Input: A sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$.
  - Output: A permutation (reordering) $A' = \langle a_1', a_2', \ldots, a_n' \rangle$ of the input sequence such that $a_1' \leq a_2' \leq \cdots \leq a_n'$.

Definitions

- Problem (问题): A question to which we seek an answer.
  - Find the non-decreasing order of a sequence of n numbers A.
- Algorithm (算法): A step-by-step procedure applying a technique for solving the problem.
  - Insertion sort
  - Quicksort
  - Mergesort
  - …
- Parameters (参数): Variables that are not assigned specific values in the statement of the problem.
  - A.
- Instance (实例): Specific assignment of values to the parameters.
  - $A = \langle 3,6,1,7,2 \rangle$.
- Solution (答案): The answer to the problem in that instance.
  - $A' = \langle 1,2,3,6,7 \rangle$.

Properties of Algorithm

- Finite (有穷性): An algorithm consists of finite number of operations.
- Feasible (可行性): Every operation is executable.
- Deterministic (确定性): Every operation must not be ambiguous (generate random results).
- Input (输入): 0 or more, describe the initial state.
- Output (输出): 1 or more, describe the result process from the input.

Pseudocode

- Pseudocode (伪代码) is a plain language description of the steps in an algorithm, irrelevant to specific programming language.
- We have the following conventions:
  - "←" represents variable assignment (变量赋值).
  - Indentation (缩进) indicates block structure. "{}" is not needed.
  - While, for, repeat, if, then, and else have the same interpretation as in most programming language.
  - // indicates comments.
  - Composite data type (复合类型) are typically organized into objects, which are comprised of attributes or fields, e.g. T.Lchild.
  - Array elements (数组) are accessed by specifying the array name followed by the index in square brackets, e.g. A[i].
    - Most of the time, we start the index i from 1, instead of 0.

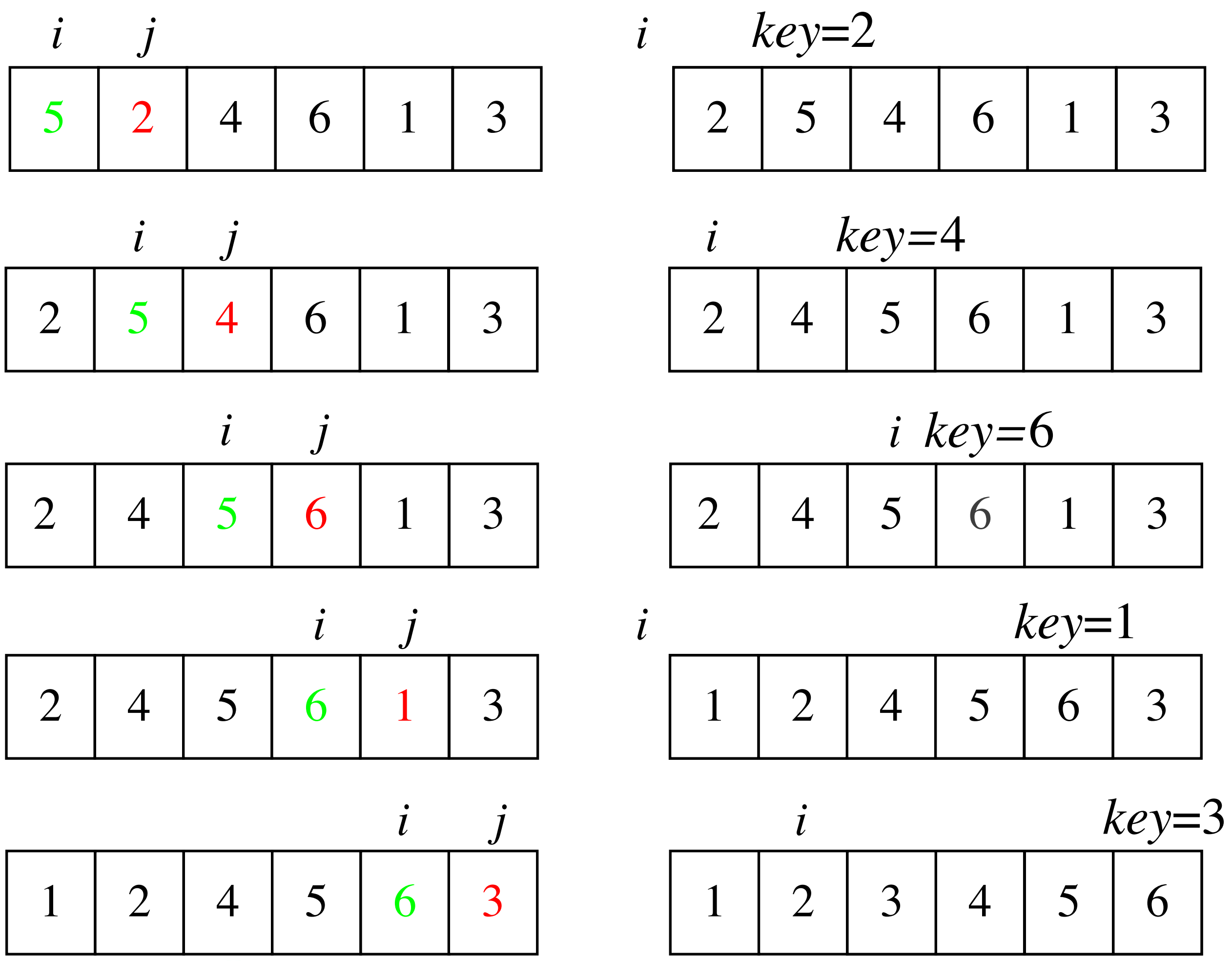
Insertion Sort for the Sorting Problem

- Insertion sort (插入排序) is one of the simplest sorting algorithms.
  - We use it all the time when we are playing cards.

```
InsertSort($A$)
1 for $j \leftarrow 2$ to $n$ do
2     $key \leftarrow A[j]$
3     $i \leftarrow j - 1$
4     while $i > 0$ and $A[i] > key$ do
5         $A[i+1] \leftarrow A[i]$
6         $i \leftarrow i - 1$
7     $A[i+1] \leftarrow key$
8 return $A$
```

Image source: 张德富, 算法设计与分析, 国防工业出版社, 2009.

Insertion Sort for the Sorting Problem



Classroom Practice

- Write every step of insertion sort for the input array $A = \langle 3,6,1,7,2,4 \rangle$

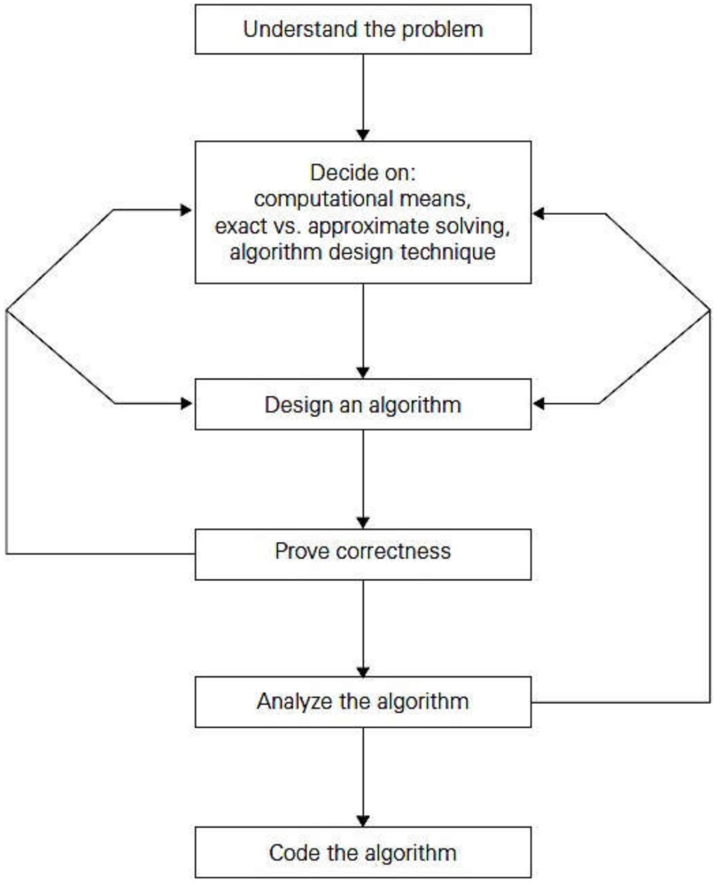
Algorithm Design Techniques and Strategies

- In this course, you are going to learn:
  - Recursive algorithms (递归算法)
  - Divide-and-conquer algorithms (分治算法)
  - Dynamic programming (动态规划)
  - Greedy algorithms (贪心算法)
  - Graph algorithms (图算法)
  - Backtracking (回溯)
  - Branch and bound (分支限界)
- They are not specific algorithms, but algorithm families.
- The algorithms in the same family share the same general idea to solve problems.

Important Problem Types

- Sorting: Rearrange the items of a given list in nondecreasing order.
- Searching: Finding a given value, called a search key, in a given set or multiset.
- String processing: e.g., matching of text strings, find subsequences.
- Graph problems: e.g., modelling transportation, communication, social and economic networks (graph traversal algorithms, shortest path algorithms).
- Combinatorial problems: e.g., resource scheduling for wireless communications.
- Geometric problems: e.g., closest-pair problem, convex-hull problem for computer graphics and robotic vision.
- Numerical problems: Solving equations and systems of equations, computing definite integrals, evaluating functions.

Process of Analysis and Design of Algorithm



厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

Image source: https://codes.pratikkataria.com/algorithmic-problem-solving/

The Correctness of Algorithms

- Before we discuss the efficiency of an algorithm, we should make sure one thing: Correctness (正确性).
  - For every input instance, the algorithm halts with the correct output.
- How to prove the correctness? Try every input instance and verify the output?
  - Can we obtain all input instances for the sorting problem?
- We can theoretically prove the correctness.

Loop Invariants

- At the start of each iteration of the for loop, the subarray $A[1 \ldots j-1]$ consists of the elements originally in $A[1 \ldots j-1]$ but in sorted order.
- We state these properties of $A[1 \ldots j-1]$ formally as a loop invariants (循环不变量).
- We can use loop invariants to prove why an algorithm is correct.

The Correctness of Insertion Sort

The loop invariants are:

$A[1 \ldots j-1]$ is sorted before each iteration.

The proof is similar to mathematical induction (数学归纳法):

- Initialization (base case): It is true prior to the first iteration of the loop.
- Maintenance (inductive step): If it is true before an iteration of the loop, it remains true before the next iteration.
- Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

The Correctness of Insertion Sort

Initialization

- We start by showing that the loop invariant holds before the first loop iteration, when $j = 2$.
- The subarray $A[1 \dots j-1]$, therefore, consists of just the single element $A[1]$, which is in fact the original element in $A[1]$.
- Obviously, the subarray with only one element is sorted, which shows that the loop invariant holds at the start of the first iteration of the loop.

The Correctness of Insertion Sort

Maintenance

- At the start of the jth iteration of loop, we assume $A[1 \dots j-1]$ is sorted and then prove $A[1 \dots j]$ is sorted after this iteration.
- The body of the while loop (lines 4-7) works by moving $A[j-1]$, $A[j-2]$, $A[j-3]$,… by one position to the right until the proper position for the key $A[j]$ is found.
- We insert the key at the point when $A[i] < key < A[i+2]$.
- Therefore, $A[1 \dots j]$ is sorted before next iteration, which shows that the loop invariant holds at the start of the $(j+1)$th iteration of loop.

The Correctness of Insertion Sort

Termination

- For insertion sort, the outer for loop ends when j exceeds n, i.e. when $j = n + 1$.
- By loop invariant of maintenance step, we have that the subarray $A[1 \dots n]$ consists of the elements originally in $A[1 \dots n]$, but in sorted order.
- The subarray $A[1 \dots n]$ is the entire array! Hence, the entire array is sorted, which means that loop invariant holds at every iteration and the algorithm is correct.

Efficiency of Algorithm

- Is correctness of an algorithm enough? How good is an algorithm?
- Time complexity (时间复杂度)
  - Indicates how fast an algorithm runs.
  - How many CPU cycles needed.
- Space complexity (空间复杂度)
  - Amount of memory units required by an algorithm.
- Does there exist a better algorithm?
- How to compare algorithms?

Time Complexity

- Running time of an algorithm on a particular input generally be the number of primitive operations or "steps" executed.
- Define the notion of step so that it is as machine independent as possible.
  - Compare one algorithm running on i3 CPU with another algorithm on i9 CPU is meaningless.
- The comparison should be instance independent.
  - We are not comparing two algorithms on a specific input instance, but in a general case.

Time Complexity

- Consider insertion sort, let $A = \langle 1,2,3,4, \ldots n \rangle$ and consider all $n!$ permutations of the elements in A. Each permutation corresponds to one possible input.
- Consider three permutations:
  - Case 1: The elements in A are sorted in decreasing order.
  - Case 2: The elements in A are sorted in increasing order.
  - Case 3: The elements in A are randomly ordered.
- Will insertion sort have the same running time on these three cases?

Time Complexity

- The worst-case time complexity (最坏情形时间复杂度) of an algorithm is the function defined by the maximum number of steps taken on any instance of size n.
- The best-case time complexity (最好情形时间复杂度) of an algorithm is the function defined by the minimum number of steps taken on any instance of size n.
- The average-case time complexity (平均情形时间复杂度) of an algorithm is the function defined by an average number of steps taken on any instance of size n.
- Which of these is the best to use?

Time Complexity

We use the following convention to analyze an algorithm:

- A constant amount of time is required to execute each line (operation) of our pseudocode.
- Each line may take a different amount of time than another line, so we shall assume that each execution of the ith line takes time c_i , which is a constant.
- For a loop, we calculate how many times the loop iterates and then multiply with the total time within this loop.

Analysis of Insertion Sort

```
InsertSort($A$)                                        Cost     Times
1  for $j \leftarrow 2$ to $n$ do                      $c_1$    $n$
2      $key \leftarrow A[j]$                           $c_2$    $n-1$
3      $i \leftarrow j-1$                              $c_3$    $n-1$
4      while $i > 0$ and $A[i] > key$ do              $c_4$    $\sum_{j=2}^{n} t_j$
5          $A[i+1] \leftarrow A[i]$                    $c_5$    $\sum_{j=2}^{n}(t_j-1)$
6          $i \leftarrow i-1$                          $c_6$    $\sum_{j=2}^{n}(t_j-1)$
7      $A[i+1] \leftarrow key$                         $c_7$    $n-1$
8  return $A$                                          $c_8$    $1$
```

- In each for loop, we run different times of while loop. Therefore, we use t_j as the number of condition tested of the jth while loop.

Analysis of Insertion Sort

- Generally, the running time increases with n.
  - Sorting 5 numbers is obviously faster than sorting 10,000 numbers.
- Therefore, it is natural to represent the total running time $T(n)$ as a function of n:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j$$

$$+c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1) + c_8$$

Best Case of Insertion Sort

- For inputs of a given size, an algorithm's running time may depend on which input of problem.
- The best case occurs for insertion sort if the array is already sorted. In this case:
  - The while loop condition fails in the beginning, i.e. $t_j = 1$ for all j.
  - Line 5-6 in the while loop are not executed.
- Then, $T(n)$ becomes:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) + c_8 \\ &= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7 - c_8) \end{aligned}$$

- This running time is a linear function of n.

Worst Case of Insertion Sort

- The worst case occurs if the array is in reverse sorted order. In this case:
  - The key is always moving to the front in the while loop and thus $t_j = j$.
- Then, $T(n)$ becomes:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} j$$

$$+c_5 \sum_{j=2}^{n} (j-1) + c_6 \sum_{j=2}^{n} (j-1) + c_7(n-1) + c_8$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right) n$$

$$-(c_2 + c_3 + c_4 + c_7 - c_8)$$

- This running time is a quadratic function of n.

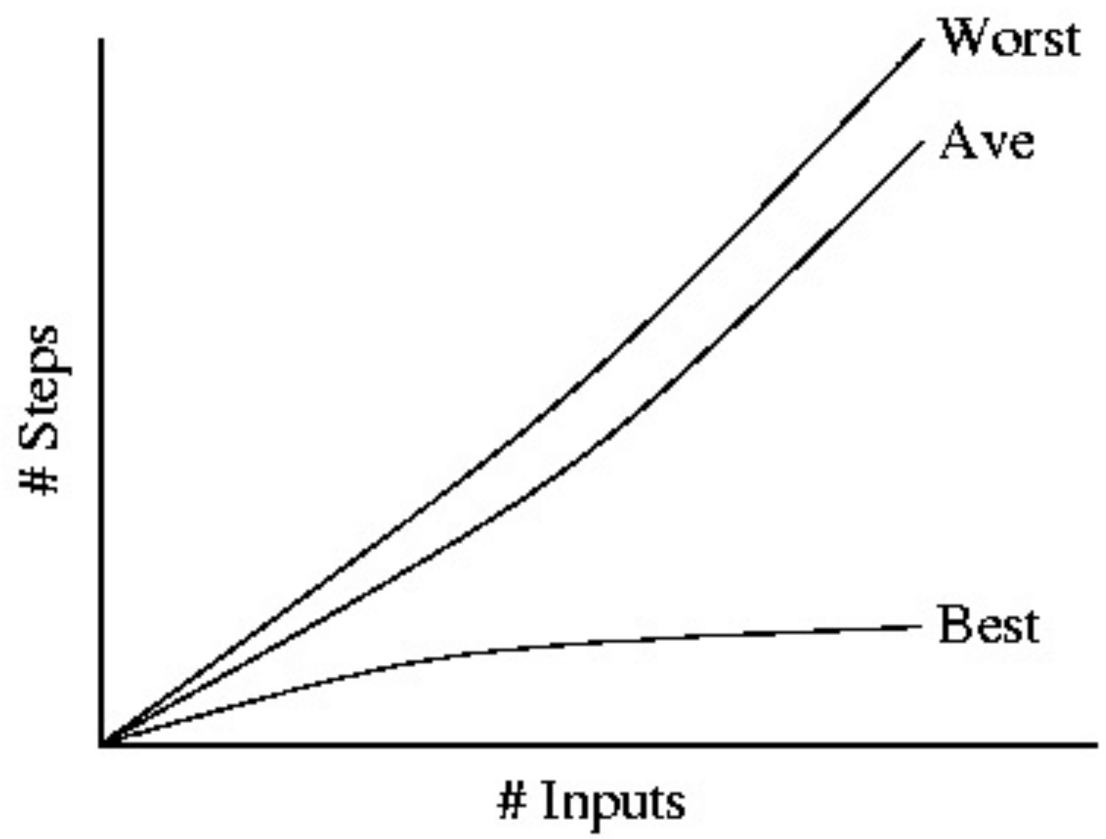
Average Case of Insertion Sort

- The average-case or expected running time is taken to be the average time over all inputs of size n.
- The average-case analysis needs know the probabilities of all input occurrences, i.e., it requires prior knowledge of the input distribution.
- The analysis is complex and lengthy in many cases.
- We will discuss it with probabilistic analysis in Lecture 3.

Best-, Worst-, and Average-Case

- Best-case analysis is not used in practice, as it does not give useful information about the behavior of an algorithm in general.
  - We will not bet our algorithm on luck.
- Generally, we concentrate on the worst-case analysis.
  - The worst-case running time of an algorithm is an upper bound on the running time for any input instance.
  - For some algorithms, the worst case occurs fairly often.
  - The average-case is often roughly as bad as the worst case.

Best-, Worst-, and Average-Case



Comparison of Time Complexity

- Given two algorithms, we obtain two $T(n)$ and each of them has a lot of c_i. How to compare them?
- Consider the best-case of insertion sort $T(n) = cn - b$, where $c = c_1 + c_2 + c_3 + c_4 + c_7$ and $b = c_2 + c_3 + c_4 + c_7 - c_8$ are constants.
- As the problem size n increases, the running time is dominated by n and does not matter much by constants c_i.

Comparison of Time Complexity

- Consider two sorting algorithms:
  - Insertion sort takes time roughly equal to $c_1 n^2$ to sort n items.
  - Mergesort takes time roughly equal to $c_2 n \lg n$ to sort n items.
- Assume Mergesort has larger constant: $c_1 = 1$ and $c_2 = 100$.
- For a small problem with $n = 10$:
  - Insertion sort takes $1 \times 10^2 = 100$ operations.
  - Mergesort takes $100 \times 10 \lg 10 \approx 3322$ operations.
- However, for a big problem with $n = 10^6$:
  - Insertion sort takes $1 \times 10^{12} = 10^{12}$ operations.
  - Mergesort takes $100 \times 10^6 \lg 10^6 \approx 1.99 \times 10^9$ operations.

Comparison of Time Complexity

- In the theoretical analysis of an algorithm, we are interested in the eventual behavior.
  - We compare algorithms for sufficiently large n.
- We can show that, no matter how large c_2 is, we can always find a $n > n_0$ such that $c_1 n^2 > c_2 n \lg n$.
- This is because the term n^2 has higher order than $n \lg n$.
- Therefore, when we compare the efficiency of algorithms, we concentrate on the order and ignore constants.

Asymptotic Running Time

- We are actually interested in the rate of growth, or order of growth. Intuitively, just look at the dominant term.

$$T(n) = \cancel{0.1}n^3 + \cancel{10n^2 + 5n + 25}$$

  - Drop lower-order terms $10n^2 + 5\mathrm{n} + 25$.
  - Ignore constant 0.1.
- But we can't say that $T(n)$ equals to n^3.
  - It grows like n^3. But it doesn't equal to n^3.
- We will define asymptotic notations in Lecture 2 to formally describe the time complexity of an algorithm.

Lower Bound of A Problem

- Given a problem, how can we know if we still have chance to improve an algorithm?
- The lower bound of a problem characterizes the complexity of a problem, informally as “the best you can do.” Namely, the minimum time needed by any algorithm to solve the problem.
- If we have an algorithm whose worst-case is equal to the lower bound of the problem, we can claim that this algorithm is optimal to the problem and the problem is solved.
  - Then, we don’t have to waste time on this problem and focus on other interesting problems.

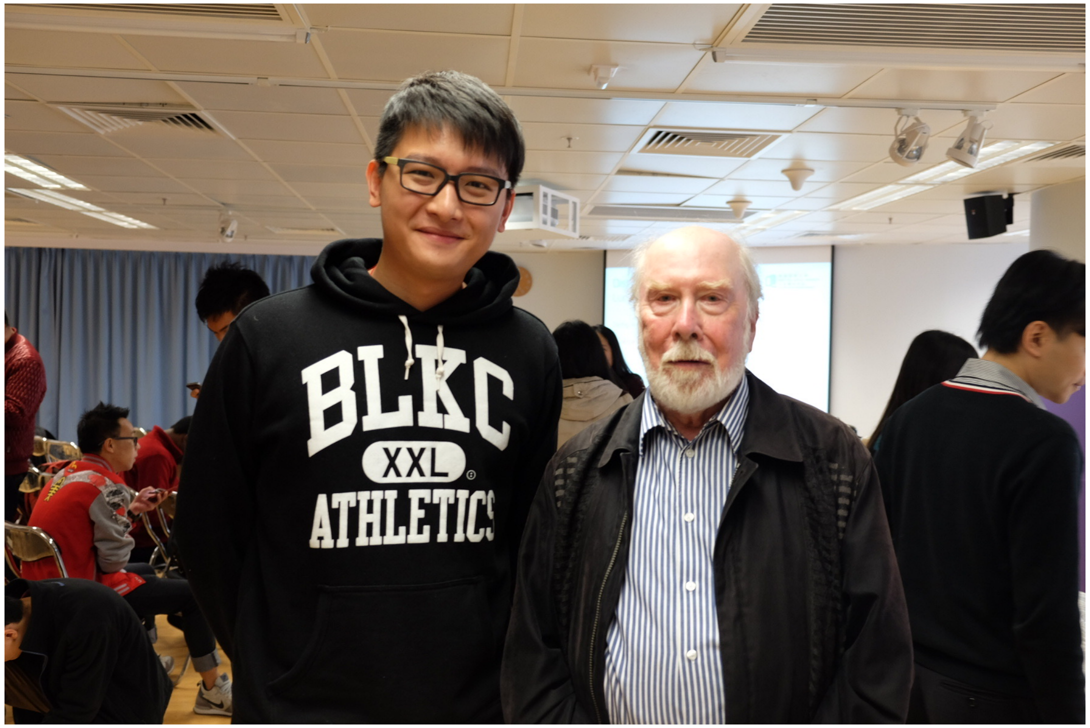
Find the Maximum Value

```
FindMax()
1  max ← $A[1]$
2  **for** $j$ ← 2 **to** $n$ **do**
3      **if** $A[j] >$ max **then**
4          max ← $A[j]$
5  **return** max
```

- Lower bound of the problem is $n - 1$.
- The worst-case of this algorithm is $n - 1$.

Conclusion

- *“Program = Data structure + Algorithm”* - Niklaus Wirth, 1984 Turing Award Laureate



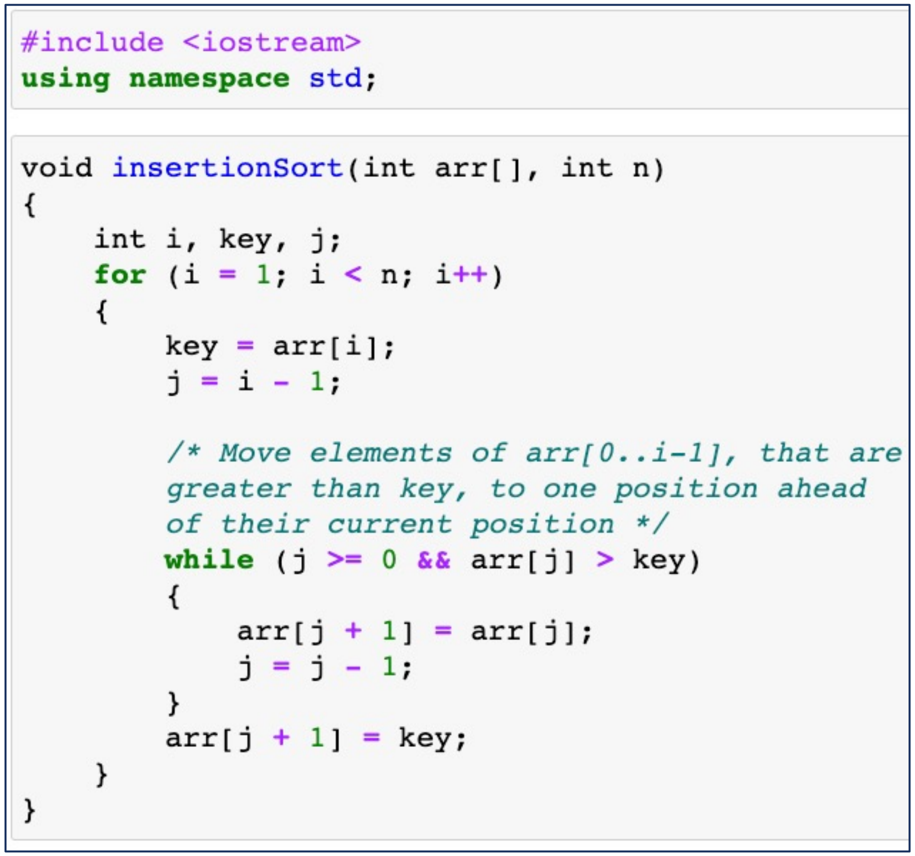
Niklaus Wirth and I at HKBU on 19 Jan 2015

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

Jupyter Notebook

- Jupyter Notebook is a web-based interactive computational environment for creating Jupyter notebook documents.
- It can execute code and show results, figures, markdown documents in one browser page.
- You can go through a quick tutorial from https://www.dataquest.io/blog/jupyter-notebook-tutorial/
- You can install Jupyter kernel for C++ , or simply use Online Jupyter Notebook for C++.
- For all experiment assignment, the report should be represented by the notebook file .ipynb.

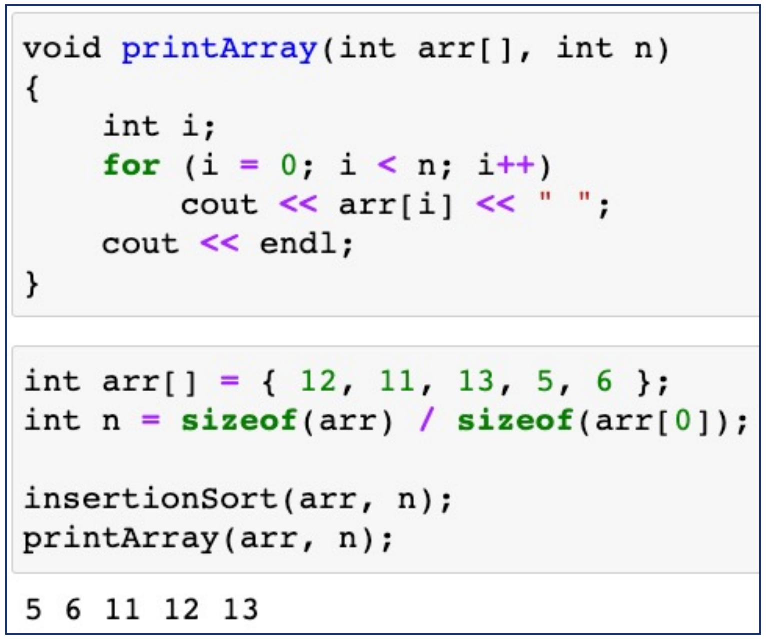
Jupyter Notebook



```cpp
#include <iostream>
using namespace std;
```

```cpp
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
        greater than key, to one position ahead
        of their current position */
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```



```cpp
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

```cpp
int arr[] = { 12, 11, 13, 5, 6 };
int n = sizeof(arr) / sizeof(arr[0]);

insertionSort(arr, n);
printArray(arr, n);
```

```
5 6 11 12 13
```

Conclusion

After this lecture, you should know:

- What is a problem and an algorithm?
- Why do we need to study algorithm?
- How to write pseudocode for an algorithm?
- Why do we need to study the efficiency of an algorithm?
- What are the three cases of time complexity?
- What is the lower bound of a problem?

Homework

- Page 11-12

  1.2 1.5

  1.6 1.7

  1.8 1.9

  1.10 1.11

Experiment

编写程序 (任选一道, 可以不选, 但是每章不得选超过一道)

- 微信红包程序: 给定一个钱数m, 发红包人数n, 其中$10 \leq m$, $n \leq 200$, 将钱数拆成几个指定的吉利数 (如1.66,1.68, 16.8,1.78,17.8,1.88,18.8,1.99,5.20,0.66,6.6,6.66,0.08,0.88,8.8,8.88,0.99,9.9,9.99) 并发出, 要求要发出n个红包, 分布比较均匀.
- 随机点名程序 (越不来上课的人, 被点中的概率越高, 实现抽查问题, 预警等功能)

谢谢

有问题欢迎随时跟我讨论