Chapter 1: Basic Concepts

Hsuan-Tien Lin

Dept. of CSIE, NTU

February 21–22, 2011

Some Historical Notes

Once upon a time, when I was a freshman in NTU CSIE (1997).....

- 「計程」有兩學期，上學期教C，下學期教C++
- 大二上學期教「資料結構」
- 大二下學期教「演算法」

Then, in my senior year (2001).....

- 「計程」變成一學期，大一下學期教「物件導向程式設計」(Java)
- 大二上學期教「資料結構與演算法上」
- 大二下學期教「資料結構與演算法下」

Then, in your freshman year (2010).....

- 物件導向程式設計變爲選修
- 大一下學期教「資料結構與演算法」
- 大二上學期教「演算法設計與分析」

Reasons

- 兩學期的「計程」變成一學期、「物件導向程式設計」變成選修：
  相信同學們可以有自己學習不同語言的能力。
- 把「資料結構」及「演算法」合成一門課：
  兩者互相依賴，其實不容易分散來教。
- 把「資料結構與演算法上/下」區分成「資料結構與演算法」和「演算法設計與分析」：
  強調前者以實作爲主，銜接計程做更深入的程式練習；後者以分析爲主，建立在前者的基礎上探討更多不同的演算法。

B99 teaching plan

- myself: two classes of Data Structures and Algorithms; one class of Algorithm Analysis
- another professor: one class of Algorithm Analysis

Programming ! = Coding

Programming is like building a house, 蓋房子需要了解

- Requirements: purpose, input/output
  (需求爲何，投入多少資金，產出什麼品質的房子)
- Analysis:
  - bottom-up: small pieces $\Longrightarrow$ ultimate goal
    (把每一面牆，每一塊磚設計好，再想辦法拼起來)
  - top-down: ultimate goal $\Longrightarrow$ small pieces
    (整體的需求爲何，再思考每一面牆，每一塊磚怎麼完成)
- Design: choices of data structures and algorithms (要用什麼建材和工法)
- Refinement and coding: actual implementation (施工)
- Verification:
  - Proof (in math)
    (設計圖上的計算與確認，例如載重量或耐震度)
  - Test and debug (on machines)
    (工地現場的牢固度測試、監工等)

This Class: from Coding to Programming

	Introduction to C	this class
Requirements	simple	simple
Analysis	simple	simple
Design	simple	⋆
Coding	⋆	◯
Proof	none	◯
Test	simple	⋆
Debug	⋆	◯

Pointers and Dynamic Memory Allocation (Sec. 1.2)

Reading Assignment

be sure to go ask the TAs or me if you are still confused

What is an Algorithm?

pass

Algorithm (演算法) $\Longleftrightarrow$ 程式譜

- 食譜
- 樂譜
- 陣譜
- 暗器譜
- 程式譜

* 五線譜、六線譜

? 族譜、光譜、臉譜

goal: solve some computational problems efficiently and correctly

What is a Data Structure?

Data Structure (資料結構) $\Longleftrightarrow$ 收納資料的方式

- 食譜： 調味料結構
- 樂譜： 交響樂團(樂器)結構
- 陣譜： 將士(火炮)結構
- 暗器譜： 暗器結構
- 程式譜： 資料結構

goal: economic and effective use of storage power
(usually, to fit the platform and/or to help the algorithm)

Why Learn Data Structures and Algorithms?

好的程式，來自於有效地運用計算機上的「資源」。其中最重要的兩項資源，一是計算核心：例如中央處理器、浮點運算器等；二是儲存媒體：例如快取、記憶體、硬碟等（並配合連線裝置）。資料結構與演算法，主要探討的即是如何有效並正確地運用這兩項資源來解決各式的問題。

- algorithms: often focus on computational resources
- data structures: often focus on storage resources

understanding DS&A helps you write better programs

Five Criteria of 食譜

阿基師的蕃茄炒蛋食譜

蕃茄3顆、蛋3顆、蔥2支、薑1小塊、太白粉、鹽、糖

1. 切蔥花備用；薑末備用。
2. 蕃茄去蒂畫十字刀，下鍋汆燙。
3. 撈起蕃茄，放入冷水中除外皮。
4. 蛋液打勻，加入少許鹽巴；將蕃茄切適當大小。
5. 起油鍋，爆香薑末，加入蕃茄，倒入適量的水，加入少許鹽、糖。
6. 加入少許太白粉勾芡。
7. 加入蛋液，輕輕翻炒。
8. 起鍋前撒上蔥花。

- Input:
  食材
- Output:
  菜色
- Definiteness:
  清楚的步驟
- Finiteness:
  一定可以做完
- Effectiveness:
  煮菜的人做得到

Five Criteria of Algorithm

SMALLEST-NUM-INDEX-FINDING
(integer array *list*, integer size *n*)

1. set *min* to 0
2. set i as $1, 2, \cdots, n-1$
   - if $list[i]$ is smaller than $list[min]$, then set *min* as i
3. return *min*

- Input: external supplies
- Output: desired output
- Definiteness: clear steps
- Finiteness: will terminate
- Effectiveness: can be done by computers

Selection Sort

SEL-SORT(integer array *list*, integer size *n*) outputs an in-place sorted list

- for i from 0 to $n-1$
  1. let *min* be the index of the smallest number from $list[i]$ to $list[n-1]$
  2. interchange $list[i]$ and $list[min]$

- Input
- Output
- Definiteness
- Finiteness
- Effectiveness

- step one: can be done by the computer with a simple modification of SMALLEST-NUM-INDEX-FINDING
- step two: can be done by the computer easily (How?)

Correctness of Selection Sort

SEL-SORT(integer array *list*, integer size *n*)
outputs an in-place sorted list

Given: integer array *list* with integer size *n*

- for i from 0 to $n-1$
  1. let *min* be the index of the smallest num. from $list[i]$ to $list[n-1]$
  2. interchange $list[i]$ and $list[min]$

Theorem

After the loop of $i = q$, for any $j > q$,

$$arr[0] \leq arr[1] \leq \cdots \leq arr[q] \leq arr[j].$$

Proof by Mathematical Induction:

- When $i = 0$, statement true (why?).
- Assume statement true when $i = t$;
  then when $i = t + 1$, (what happens?)

Sequential and Binary Search

- Input: a **sorted** integer array *list* with size n, an integer *searchnum*
- Output: if *searchnum* is within *list*, its index; otherwise -1

SEQ-SEARCH (*list*, n, *searchnum*)

```
for i ← 0 to n − 1 do
    if list[i] == searchnum
        return i
    end if
end for
return −1
```

BIN-SEARCH (*list*, n, *searchnum*)

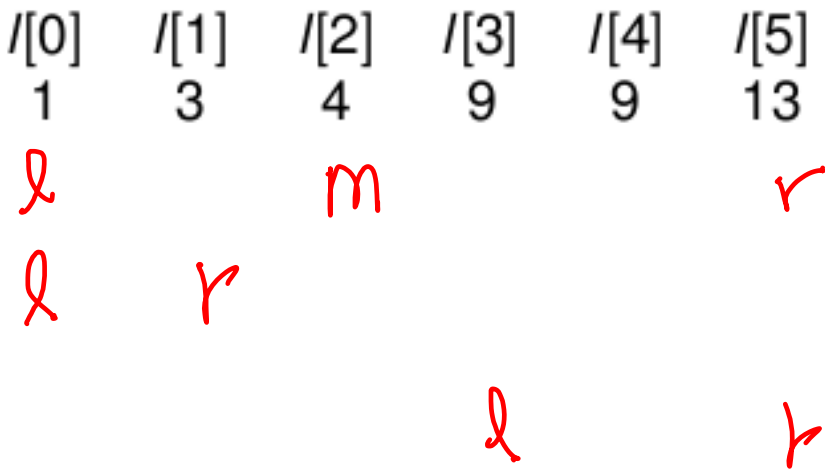
```
left ← 0, right ← n − 1
while left ≤ right do
    middle ← floor((left + right)/2)
    if list[middle] > searchnum
        right ← middle − 1
    else if list[middle] < searchnum
        left ← middle + 1
    else        /* list[middle] == searchnum */
        return middle
    end if
end while
return −1
```

note: use pseudo code here instead of actual C code in the textbook

Binary Search: Example

BIN-SEARCH (*list*, *n*, *searchnum*)

```
left ← 0, right ← n − 1
while left ≤ right do
    middle ← floor((left + right)/2)
    if list[middle] > searchnum
        right ← middle − 1
    else if list[middle] < searchnum
        left ← middle + 1
    else
        return middle
    end if
end while
return −1
```



Another Version of Binary Search

BIN-SEARCH (*list*, *n*, *searchnum*)

```
left ← 0, right ← n − 1
while left ≤ right do
    middle ← floor((left + right)/2)
    if list[middle] > searchnum
        right ← middle − 1
    else if list[middle] < searchnum
        left ← middle + 1
    else
        return middle
    end if
end while
return −1
```

BIN-SEARCH (*list*, *l*, *r*, *s*)

```
if l ≤ r
    m ← floor((l + r)/2)
    if list[m] > s
        return BIN-SEARCH(list, l, m − 1, s)
    else if list[m] < s
        return BIN-SEARCH(list, m + 1, r, s)
    else
        return m
    end if
end if
return −1
```

iterative versus recursive

Recursive Algorithms (Subsec. 1.3.2)

Reading Assignment

be sure to go ask the TAs or me if you are still confused

Data Abstraction (Section 1.4)

Suggest Prep-Reading;
Will Cover in the Next Class