Competitive Programming, Big-O Notation

PRESENTER: LEUNG MAN HO (LMH)

Competitive Programming – What is it?

- Solving problems
  - Logic & Mathematic
  - Algorithm & Data Structure
  - Coding
- Limit
  - Time Limit
    - Contest Time
    - Program Runtime
  - Memory Limit
- Example
  - Google Code Jam
  - Facebook Hackercup
  - ACM ICPC

Competitive Programming – What are the problems

- Topics
  - Ad-hoc
  - Mathematics
    - Number Theory
    - Combinatorics
    - Geometry
    - Graph Theory
  - String Analysis
  - Searching
  - Dynamic Programming
  - Game Theory
  - Data Structures
- OI is not just about Programming

Competitive Programming – Expectation

- Before the long journey…
- After this lecture, you should know ways to:
  - Become a good problem solver
  - Become a good programmer
  - Train yourself to be a grandmaster
  - Outperform your opponents in competitions

Competitive Programming – Flow

- Learn more about competitive programming
  - Programming Competitions & Scoring
  - Tasks
- Learn ways to become better
  - Foundation of Competitive Programming
  - Strategies & Tricks
  - Self Learning Platform

Competitive Programming – Programming Competitions & Scoring

- HKOI-Final-Style
  - Individual
  - Score given to each correct test cases
  - Live Feedback
  - Partial solution is important
- IOI-Style (HKOI-TFT-2014-Style)
  - Individual
  - Batch processing
    - Score only given when passes all data in a batch
  - Live Full Feedback
  - Partial solution is important

Competitive Programming – Programming Competitions & Scoring

- ACM-ICPC-Style
  - Team-Based (3)
  - No partial credit
  - Time penalty
    - Accepted Time
    - #Wrong Submission
  - Allow use of "Weapon"
  - Live Full Feedback
  - Score ~ #Tasks Solved & Time penalty

- Codeforces/Topcoder–Style
  - Individual
  - No partial credit
  - Time penalty
    - Accepted Time
    - #Wrong Submission
  - Pretest & Hack system
  - Live Feedback

Competitive Programming – Tasks

- Normal
  - Input & Output
- Interactive
  - Ask question to system repetitively, compute solution using system response
  - Game
  - Example
    - Given an unknown sequence of A
    - Ask the system to compare A[i] and A[j]
    - Output the sorted order of A within minimal possible questions

Competitive Programming – Tasks

- Output Only
  - Challenging Tasks
  - Allow more offline precompute & Manuel Computation
- Communication Task
  - Write two (or more) programs that communicate to each other by specific interface (i.e. limited information)
  - Encryption & Decryption
  - Example : HKOI 2014 Senior Dividing the Cities

Competitive Programming - Foundation of Competitive Programming

- Algorithm
  - Procedure to solve problems
  - Examples
    - Sorting – Bubble Sort, Merge Sort, Quick Sort
    - Shortest Path – Ford Warshall, Bellman Ford, SPFA
    - Searching – BFS, DFS
- Data Structure
  - Manner to organize data
  - Examples
    - Hash Table
    - Segment Tree
- Both are related
- Learning these knowledge is important
  - If I have seen farther than others, it is because I was standing on the shoulders of giants
  - Don't reinvent the wheels

Competitive Programming - Strategies & Tricks

- Reading the problems
- Choosing a problem
- Reading the problem
- Thinking
- Coding
- Testing
- Finalizing the program

Competitive Programming - Reading the problems

- Title
- Problem Description
- Constraints
- Input/Output Specification
- Sample Input/Output
- Scoring

Competitive Programming - Reading the problems

- Constraints
  - Range of variables
  - Execution Time
- NEVER make assumptions yourself
  - Ask whenever you are not sure
  - (Do not be afraid to ask questions!)
- Read every word carefully
- Make sure you understand before going on

Competitive Programming - Thinking

- Classify the problem into certain type(s)
  - Oiers Intinct
  - Required Algorithm/Data Structure?
- Rough works
- Special cases, boundary cases
- No idea? Give up first, do it later. Spend time for other problems.

Competitive Programming - Thinking

- Make sure you know what you are doing before coding
  - Hand writing code could be useful
- Points to note:
  - Complexity (BOTH time and space)
    - Give up solutions when it greatly fail in complexity analysis
  - Coding difficulties

Competitive Programming - Coding

- Short variable names
  - Use i, j, m, n instead of no_of_schools, name_of_students, etc.
- No comments needed
- As long as YOU understand YOUR code, okay to ignore all “appropriate“ coding practices

Competitive Programming - Coding

- Avoid using floating point variables if possible
  - real, double
- Do not do small (aka useless) “optimizations”
- Save and compile frequently

Competitive Programming - Testing

- Sample Input/Output

  "A problem has sample output for two reasons:

  1. To make you understand what the correct output format is

  2. To make you believe that your incorrect solution has solved the problem correctly "
- Manual Test Data
- Program-generated Test Data (if time allows)
- Boundary Cases (0, 1, other smallest cases)
- Large Cases (to check for TLE, overflows, etc)
- Tricky Cases
- Test by self-written program (again, if time allows)

Competitive Programming - Debugging

- Debugging – find out the bug, and remove it
  - Easiest method: writeln/printf/cout
- Debug message

Competitive Programming - Finalizing

- Check output format
  - Any trailing spaces? Missing end-of-lines? (for printf users, this is quite common)
  - better test once more with sample output
  - Remember to clear those debug messages
- Check I/O – filename? stdio?
- Check exe/source file name
- Is the executable updated?
- Method of submission?
- Try to allocate ~5 mins at the end of competition for finalizing

Competitive Programming - Tricks

- Solve for simple cases
  - 50% (e.g. slower solution, brute force)
  - Special cases (smallest, largest, etc)
  - Incorrect greedy algorithms
  - Very often, slow and correct solutions get higher scores than fast but wrong solutions
- Hard Code
  - "No solution"
  - Stupid Hardcode: begin writeln(random(100)); end.
  - Naïve hardcode: "if input is x, output hc(x)"
  - More "intelligent" hardcode (sometimes not possible):
    - pre-compute the values, and only save some of them

Competitive Programming - Pitfalls

- Misunderstanding the problem
- Not familiar with competition environment
- Output format
- Using complex algorithms unnecessarily
- Choosing the hardest problem first

Competitive Programming - Summary

- They are only soft skills
  - Useful
  - Not Panacea
- Hard Skills
  - Studying Algorithm & Data Structure
  - Practice
  - Contest Experience

Becoming Grandmaster – Online Judge

- HKOI Judge System
  - Updated, what's new?
    - User Friendliness
    - Elegant
    - Features
  - https://judge.hkoi.org
- Peking University Online Judge
  - http://poj.org/
- UVa Online Judge
  - http://uva.onlinejudge.org/

Becoming Grandmaster – POJ

- Unlimited Tasks
  - Wide range of difficulties & topics
- Large Community
  - Don't know how to do?
    - Discussion Board
    - Search for problem's name
  - Don't know which problems to start?
    - Search for classification
- Great platform to test your skill
  - Test your algorithm & data structure's implementation

Becoming Grandmaster – Online Contest

- HKOI Mini-competitions
  - Larger range of difficulties would be provided this year
- Codeforces
  - Two divisions
  - Contest per week (approx.)
  - http://codeforces.com/
- Topcoder
  - Advance
  - http://www.topcoder.com/

Becoming Grandmaster – Codeforces

- Nice UI
- Suitable level for HKOIers
- Regular Competitions
- Exciting rating system
- Large community
- Mutual learning system
  - Can read others' submission

Becoming Grandmaster – Hall of Fame

RATING FRIENDS RATING

Country: Hong Kong, 92 City: any city

Organization: any organization

Rating

	Who	#	=
1 (277)	alex20030190	33	2321
2 (281)	GagGuy	104	2318
3 (516)	AlanC	26	2223
4 (612)	Sampson	54	2197
5 (695)	murphy	13	2172
6 (698)	thinfaifai	24	2171
7 (760)	whhone	34	2156
8 (810)	I_know_nothing	68	2144
9 (1033)	tckwok0	3	2093
10 (1218)	ytau	20	2057

Becoming Grandmaster – Is it possible?

International master I_know_nothing

Theogry Leung, Sha Tin, Hong Kong
From CUHK

Contest rating: 2144 (max. grandmaster, 2217)

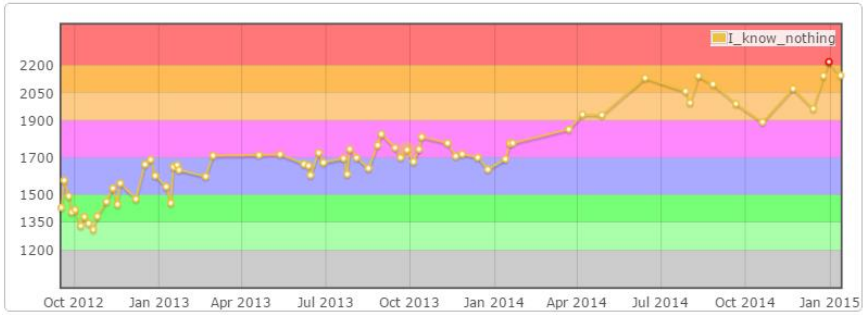
Contribution: +13

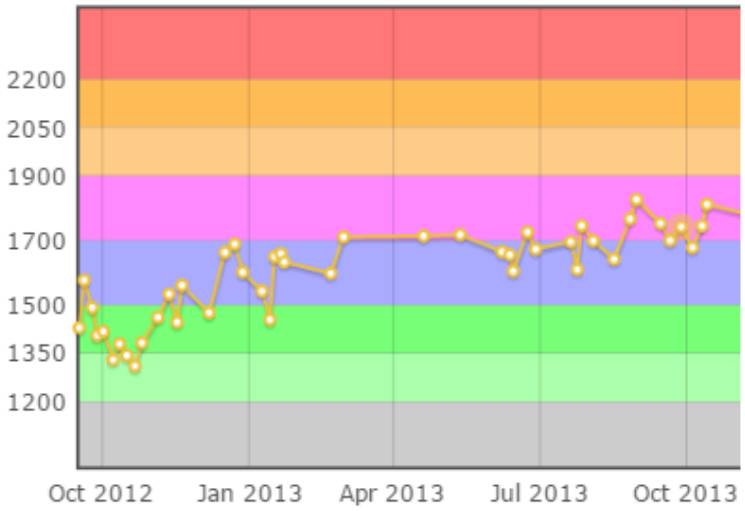
Last visit: 4 hours ago

Registered: 2 years ago

Comments

Send message to I_know_nothing





Becoming Grandmaster - Key

- Patience
  - OI is a big topic
  - Usually take years to master
- Passion
  - Road to Grandmaster is tough
  - Self Learning
- Practice
  - Problems
  - Contests

Becoming Grandmaster - Suggestion

- Communicate with cows
  - Online
  - Live (HKOI Training)
- Learn from cows
  - Read their code, learn their programming practice
- Learn from mistakes
  - Re-do past problems in contests
- DO NOT fear of failing
  - Confront your weaknesses
  - Fastest way to learn

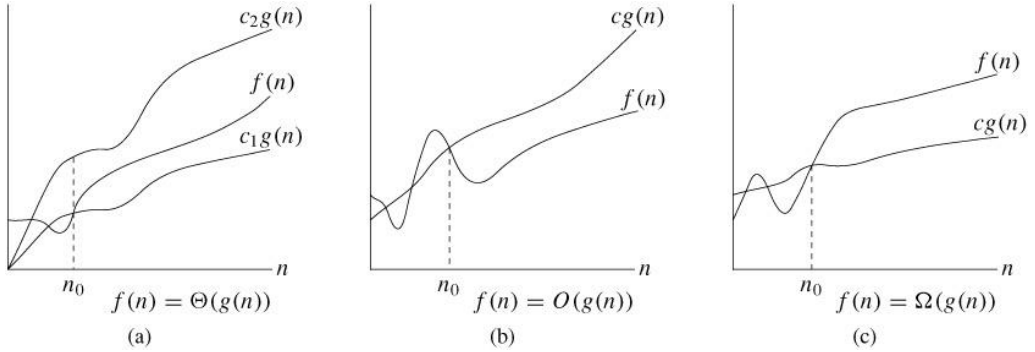
Big O Notation – Usage

- Describe functions' growth rate
- In OI sense, Big-O is used to…
  - Analysis Complexity
    - Time
    - Memory
  - Evaluate Algorithms & Data Structures

- After this chapter, you should know how to …
  - Understand why others'/your solution is better
  - Estimate your scores in competitions
  - Save time from useless optimization

Big O Notation – Definition (Mathematics Sense)

- $f(x) = O\big(g(x)\big) \leftrightarrow \exists x_0, M\colon |f(x)| \leq M\,|g(x)|, \forall x > x_0$
- Forget this after getting the feeling of what Big-O is
- Big-O is upper bound of a function
- It says that there is some point x_0 past which $M|g(x)|$ is always at least as large as $|f(x)|$
- There is other O notations
  - Small O
  - Omega



Big O Notation – Computation

- Generally interested in tightest upper bound
- Not interested to constant, smaller term
  - $O\big(f(n)\big) + O\big(g(n)\big) = \max\big(O\big(f(n)\big), O\big(g(n)\big)\big)$
- Example
  - $O(6n^2 + 4n + 2) = O(n^2)$

Big O Notation – Exercises

- State the Big O Notation of the following functions
  - $f(n) = 689$
  - $f(n) = 689n$
  - $f(n) = 6n + 89$
  - $f(n) = 6n^8 + 9n$
  - $f(n) = 6^n + 8n^9$

Big O Notation – Exercises Answers

- State the Big O Notation of the following functions
  - $f(n) = 689$
    - $O(n) = 1$
  - $f(n) = 689n$
    - $O(n) = n$
  - $f(n) = 6n + 89$
    - $O(n) = n$
  - $f(n) = 6n^8 + 9n$
    - $O(n) = n^8$
  - $f(n) = 6^n + 8n^9$
    - $O(n) = 6^n$

Big O Notation – Order of Recurrence Function (Challenging)

- Recurrence Function
  - Function calling itself
  - Base Cases
  - Example
    - Fibonacci
      - $f(n) = f(n-1) + f(n-2), n \geq 2$
      - $f(n) = 1, n \leq 1$
      - 1, 1, 2, 3, 5, 8……
    - Binary Search
      - $f(n) = f\left(\frac{n}{2}\right) + 1, n \geq 2$
      - $f(n) = 1,\ n \leq 1$

Big O Notation – Order of Recurrence Function (Challenging)

- Hard Way
  - Find the exact form of function
    - Solving Characteristic Function
    - Guess & Verify Method
    - Attend our Mathematics (I) :D
    - Exact Form of Fibonacci Function $= \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$
- Easy way
  - Master Theorem
    - $T(n) = aT\left(\frac{n}{b}\right) + f(n), a \geq 1, b > 1$
    - Common form for computing algorithm

Big O Notation – Master Theorem

- $T(n) = aT\left(\frac{n}{b}\right) + f(n), a \geq 1, b > 1$
- $n^c \log^k n$ is strict bound of $f(n)$ where $c = \log_b a$

  $\rightarrow n^c \log^{k+1} n$ is also strict bound of $T(n)$
- Example: $T(n) = 2T\left(\frac{n}{2}\right) + n$
  - $a = 2, b = 2, c = 1, f(n) = n$
  - $n^c \log^k n$ is strict bound of $f(n)$ where $c = 1, k = 0$
  - $\log_b a = \log_2 2 = 1 = c$
  - $n^c \log^{k+1} n = n \log n$ is also strict bound of $T(n)$
  - $T(n)$ is # elementary operation in Merge Sort

- There are other forms too

Big O Notation – Applications in OI

- Time Complexity
  - How long the program run
  - ~ Number of elementary operations
- Memory Complexity
  - How much ram it consumed
  - ~ Number of basic data slot
- Generally interested in worst case performance

Big O Notation – Time Complexity

- Intuition
  - Imagine a secret counter behind each elementary operations
  - How large is it after your program run? (Under Big-O)
- Common Time Complexities
  - $O(1)$, Constant
  - $O(\log n)$, Logarithmic
  - $O(n)$, Linear
  - $O(n \log n)$, Linearithmic
  - $O(n^2)$, Quadratic
  - $O(n^3)$, Cubic
  - $O(2^n)$, Exponential
  - $O(n!)$, Factorial

Big O Notation - $O(1)$, Constant

- Summation of first n non-negative integers

```
sum = n * (n + 1) / 2
```

- Summation of first 10000 non-negative intergers

```
FOR i = 1 TO 10000 DO
 sum = sum + i
ENDFOR
```

- Runtime do not scale with any variables

Big O Notation - $O(\log n)$, Logarithmic

- Binary Search

```
WHILE (max >= min) DO
  mid = midpoint(min, max)
  IF (A[mid] = key)
    // key found at index mid
  ELSEIF (A[mid] < key)
    min = mid + 1
  ELSE
    max = mid - 1
ENDWHILE
```

- Log base does not matter
  - Change-Of-Base Formula
    - $\log_b x = \frac{\log_d x}{\log_d b}$
    - Factor of constant

- Counting number of bits

```
WHILE (x != 0) DO
  bits = bits + x MOD 2
  x = x / 2
ENDWHILE
```

Big O Notation - $O(n)$, Linear

- Read n integer as input

```
FOR i = 1 to n DO
  INPUT(A[i])
ENDFOR
```

- Finding maximum value among n integer

```
FOR i = 1 to n DO
  ans = MAX(ans, A[i])
ENDFOR
```

Big O Notation - $O(n \log n)$, Linearithmic

- Merge Sort

```
MERGESORT(a, b)
  HANDLE BASE CASE
  mid = (a + b) / 2
  MERGESORT(a, mid)
  MERGESORT(mid + 1, b)
  MERGE(a, mid, b)
END
```

- $T(n) = 2T\left(\frac{n}{2}\right) + n$
  - Master Theorem

Big O Notation - $O(n^2)$, Quadratic

- Bubble Sort

```
FOR i = 1 TO n DO
  FOR j = 2 TO n - i DO
    IF (A[j] > A[j - 1])
      swap(A[j], A[j - 1])
    ENDIF
  ENDFOR
ENDFOR
```

  - #Max swaps = #Max Inversions = $\frac{n \times (n-1)}{2}$

- Initialize a 2D array ($n \times n$)

```
FOR i = 1 to n DO
  FOR j = 1 to n DO
    A[i][j] = 0;
  ENDFOR
ENDFOR
```

Big O Notation - $O(n^3)$, Cubic

- Counting number of different triangles with integer side-length within n

```
FOR i = 1 TO n DO
  FOR j = i TO n DO
    FOR k = j TO n DO
      IF (i + j > k AND i + k > j AND j + k > i)
        cnt = cnt + 1
      ENDIF
    ENDFOR
  ENDFOR
ENDFOR
```

Big O Notation - $O(2^n)$, Exponential

- Exhaust all N-bitset

```
BITSET(x)
  IF (x = n)
    OUTPUT(Bit)
  ENDIF
  Bit[n] = 0
  BITSET(n + 1)
  Bit[n] = 1;
  BITSET(n + 1)
END
```

Big O Notation - $O(n!)$, Factorial

- Exhaust all permutations of $(1..n)$

```
WHILE HAS_NEXT_PERMUTATION DO
  OUTPUT(NEXT_PERMUTATION)
```

Big O Notation – Guess the solution

- Common Skill
- Assumption
  - Modern computer can perform $\sim 10^7 - 10^8$ elementary operations
  - Constant factor hidden in Big O is insignificant
  - Constraints are well set :o)
- Conversion Table
  - Highly Machine & Compiler Dependent
    - CPU
    - Flags
  - Only as reference, be FLEXIBLE!
- Can you evaluate your Final score now?

n	Worst Acceptable Solution
$10^{10^8} < n$	$O(1)$
$10^6 < n \leq 10^{10^8}$	$O(1)$ or $O(\log n)$
$10^4 < n \leq 10^6$	$O(n)$ or $O(n \log n)$
$500 < n \leq 10^4$	$O(n^2)$
$100 < n \leq 500$	$O(n^3)$
$25 < n \leq 100$	$O(n^4)$
$12 < n \leq 25$	$O(2^n)$
$1 < n \leq 12$	$O(n!)$

Big O Notation – Memory Complexity

- Similar to Time Complexity
- Beware of
  - Memory limit
  - Size of self-declared data structure
  - Size of imported data structure (map, vector, etc)
  - Stack overflow

END