Contest Strategy

Marco Gallotta

February 27, 2009

Introduction

$$\text{score} \approx \text{skill} + \text{strategy}$$

Outline

Outline

Every Point Matters

- With nearly 300 contestants in the IOI, scores are extremely close.

22	Daniel Grunwald	Germany	90	73	50	100	50	30	393	Gold
23	Rostislav Rumenov	Bulgaria	80	100	25	92	84	10	391	Gold
24	Christopher Chen	Australia	100	100	32	100	56	0	388	Gold
24	Shang-En Huang	Chinese Taipei	100	100	23	100	50	15	388	Gold
26	Phitchaya Phothilimthana	Thailand	100	73	40	100	74	0	387	Silver
27	Cosmin Gheorghe	Romania	80	55	100	100	50	0	385	Silver
27	Shahar Papini	Israel	80	100	25	100	50	30	385	Silver
29	Jae Hyun Park	Korea	100	73	16	100	90	0	379	Silver

- Not even Bruce is immune: he once missed gold by 6 points!

Medal Cut-Offs

Year	Bronze	Silver	Gold
2005	275	383	496
2006	219	314	285
2007	187	286	388
2008	127	229	356
Average	**202**	**303**	**381**

Brute Force

- Problems are designed to reward correct attempts
- Points increase as solution approaches optimal time and space complexity
  - e.g. “For a number of tests, worth a total of 40 points, N will not exceed 18.”
- Brute force can get ~30 points or more
  - Submit at least a brute force solution for *all* questions!

Brute Force

- Problems are designed to reward correct attempts
- Points increase as solution approaches optimal time and space complexity
  - e.g. “For a number of tests, worth a total of 40 points, N will not exceed 18.”
- Brute force can get ~30 points or more
  - Submit at least a brute force solution for *all* questions!

Back to Medal Cut-Offs

- Brute force can get ~30 points or more — what does this mean?

Year	Bronze	Silver	Gold
2005	275	383	496
2006	219	314	285
2007	187	286	388
2008	127	229	356
Average	**202**	**303**	**381**

- Single 100 and the rest brute force should earn you bronze (~250)
- Throw in a second 100 and you're in silver territory (~320)
- Gold's a tough nut to crack, but with a little luck it's possible

Back to Medal Cut-Offs

- Brute force can get ~30 points or more — what does this mean?

Year	Bronze	Silver	Gold
2005	275	383	496
2006	219	314	285
2007	187	286	388
2008	127	229	356
Average	**202**	**303**	**381**

- Single 100 and the rest brute force should earn you bronze (~250)
- Throw in a second 100 and you're in silver territory (~320)
- Gold's a tough nut to crack, but with a little luck it's possible

Back to Medal Cut-Offs

- Brute force can get ~30 points or more — what does this mean?

Year	Bronze	Silver	Gold
2005	275	383	496
2006	219	314	285
2007	187	286	388
2008	127	229	356
Average	**202**	**303**	**381**

- Single 100 and the rest brute force should earn you bronze (~250)
- Throw in a second 100 and you're in silver territory (~320)
- Gold's a tough nut to crack, but with a little luck it's possible

Back to Medal Cut-Offs

- Brute force can get ~30 points or more — what does this mean?

Year	Bronze	Silver	Gold
2005	275	383	496
2006	219	314	285
2007	187	286	388
2008	127	229	356
Average	**202**	**303**	**381**

- Single 100 and the rest brute force should earn you bronze (~250)
- Throw in a second 100 and you're in silver territory (~320)
- Gold's a tough nut to crack, but with a little luck it's possible

Outline

Correctness

- Correctness of algorithm
  - Proof of correctness (Kosie/Francois to talk on this)
  - Compare to brute force solution
- Correctness of implementation
  - Careful analysis of code
  - Testing

Correctness

- Correctness of algorithm
  - Proof of correctness (Kosie/Francois to talk on this)
  - Compare to brute force solution
- Correctness of implementation
  - Careful analysis of code
  - Testing

Test Runs

- Test runs comprise multiple test cases
  - Better to have slow, but correct solution than fast, but incorrect solution
  - Problem with incomplete solutions
- Detailed feedback returns results of your submission on some of the judging data
  - In IOI 2008 three of six problems had detailed feedback
  - Each had about five detailed feedback cases, some testing correctness and some testing runtime/memory
  - 100% here doesn't imply a correct solution, but good chance it is
  - Use it as early testing

Test Runs

- Test runs comprise multiple test cases
  - Better to have slow, but correct solution than fast, but incorrect solution
  - Problem with incomplete solutions
- Detailed feedback returns results of your submission on some of the judging data
  - In IOI 2008 three of six problems had detailed feedback
  - Each had about five detailed feedback cases, some testing correctness and some testing runtime/memory
  - 100% here doesn't imply a correct solution, but good chance it is
  - Use it as early testing

Test Runs

- Test runs comprise multiple test cases
  - Better to have slow, but correct solution than fast, but incorrect solution
  - Problem with incomplete solutions
- Detailed feedback returns results of your submission on some of the judging data
  - In IOI 2008 three of six problems had detailed feedback
  - Each had about five detailed feedback cases, some testing correctness and some testing runtime/memory
  - 100% here doesn't imply a correct solution, but good chance it is
  - Use it as early testing

What To Test?

- Boundary cases:
  - small/large values, off-by-one errors
- Extreme cases:
  - maximum time *and* memory
  - overflow
- Code coverage

What To Test?

- Boundary cases:
  - small/large values, off-by-one errors
- Extreme cases:
  - maximum time *and* memory
  - overflow
- Code coverage

What To Test?

- Boundary cases:
  - small/large values, off-by-one errors
- Extreme cases:
  - maximum time *and* memory
  - overflow
- Code coverage

How To Test?

- Assertions: `assert(condition)` in `#include <cassert>`
- Create list of test cases from when you start reading
  - Add to this list as you develop your algorithm and implementation
- Generate random test cases
- Verify small cases with brute force program
- After fixing a bug, run *all* test cases again
  - Script to automate testing

How To Test?

- Assertions: `assert(condition)` in `#include <cassert>`
- Create list of test cases from when you start reading
  - Add to this list as you develop your algorithm and implementation
- Generate random test cases
- Verify small cases with brute force program
- After fixing a bug, run *all* test cases again
  - Script to automate testing

How To Test?

- Assertions: `assert(condition)` in `#include <cassert>`
- Create list of test cases from when you start reading
  - Add to this list as you develop your algorithm and implementation
- Generate random test cases
- Verify small cases with brute force program
- After fixing a bug, run *all* test cases again
  - Script to automate testing

When You Find a Bug?

- Could be either an incorrect algorithm or bug in implementation
- Debug using gdb (Max to talk on this)
- Know when to accept “failure” and move on

When You Find a Bug?

- Could be either an incorrect algorithm or bug in implementation
- Debug using gdb (Max to talk on this)
- Know when to accept “failure” and move on

Outline

Time Management

- Allocate your time wisely (rough guidline):
  - 1 hour to read and formulate preliminary solutions for *all* questions
  - 30 minutes per question at the end when you give up and go for brute force
  - 10 minutes for final checks
  - About 3 hours left
    - Implementation, testing, debugging
- Solve easiest questions first

Time Management

- Allocate your time wisely (rough guidline):
  - 1 hour to read and formulate preliminary solutions for *all* questions
  - 30 minutes per question at the end when you give up and go for brute force
  - 10 minutes for final checks
  - About 3 hours left
    - Implementation, testing, debugging
- Solve easiest questions first

Time Management

- Allocate your time wisely (rough guidline):
  - 1 hour to read and formulate preliminary solutions for *all* questions
  - 30 minutes per question at the end when you give up and go for brute force
  - 10 minutes for final checks
  - About 3 hours left
    - Implementation, testing, debugging
- Solve easiest questions first

Time Management

- Allocate your time wisely (rough guidline):
  - 1 hour to read and formulate preliminary solutions for *all* questions
  - 30 minutes per question at the end when you give up and go for brute force
  - 10 minutes for final checks
  - About 3 hours left
    - Implementation, testing, debugging
- Solve easiest questions first

Time Management

- Allocate your time wisely (rough guidline):
  - 1 hour to read and formulate preliminary solutions for *all* questions
  - 30 minutes per question at the end when you give up and go for brute force
  - 10 minutes for final checks
  - About 3 hours left
    - Implementation, testing, debugging
- Solve easiest questions first

Time Management

- Allocate your time wisely (rough guidline):
  - 1 hour to read and formulate preliminary solutions for *all* questions
  - 30 minutes per question at the end when you give up and go for brute force
  - 10 minutes for final checks
  - About 3 hours left
    - Implementation, testing, debugging
- Solve easiest questions first

Developing Implementation

- Consider writing a template you can memorise
- Implement core algorithm first
- Abstract data structures and operations
  - Fill with brute force methods
  - Test on small cases to check correctness
  - Replace with efficient algorithms if core is correct
  - Test against brute force method to ensure correctness
- You “always” have a working solution if you run out of time

Developing Implementation

- Consider writing a template you can memorise
- Implement core algorithm first
- Abstract data structures and operations
  - Fill with brute force methods
  - Test on small cases to check correctness
  - Replace with efficient algorithms if core is correct
  - Test against brute force method to ensure correctness
- You “always” have a working solution if you run out of time

Developing Implementation

- Consider writing a template you can memorise
- Implement core algorithm first
- Abstract data structures and operations
  - Fill with brute force methods
  - Test on small cases to check correctness
  - Replace with efficient algorithms if core is correct
  - Test against brute force method to ensure correctness
- You “always” have a working solution if you run out of time

Developing Implementation

- Consider writing a template you can memorise
- Implement core algorithm first
- Abstract data structures and operations
  - Fill with brute force methods
  - Test on small cases to check correctness
  - Replace with efficient algorithms if core is correct
  - Test against brute force method to ensure correctness
- You “always” have a working solution if you run out of time

Developing Implementation

- Consider writing a template you can memorise
- Implement core algorithm first
- Abstract data structures and operations
  - Fill with brute force methods
  - Test on small cases to check correctness
  - Replace with efficient algorithms if core is correct
  - Test against brute force method to ensure correctness
- You “always” have a working solution if you run out of time

Developing Implementation

- Good idea to leave brute force method for small cases
- Compare:
  - Simple bug in complex data structure: 10 points
  - Brute force: 30 points
  - Brute force for small cases only: 40 points
- Also for unproven optimisations (e.g. greedy):
  - Lucky if correct, unaffected if incorrect

Developing Implementation

- Good idea to leave brute force method for small cases
- Compare:
  - Simple bug in complex data structure: 10 points
  - Brute force: 30 points
  - Brute force for small cases only: 40 points
- Also for unproven optimisations (e.g. greedy):
  - Lucky if correct, unaffected if incorrect

Developing Implementation

- Good idea to leave brute force method for small cases
- Compare:
  - Simple bug in complex data structure: 10 points
  - Brute force: 30 points
  - Brute force for small cases only: 40 points
- Also for unproven optimisations (e.g. greedy):
  - Lucky if correct, unaffected if incorrect

Outline

Problem Solving

- Admissible time complexities (~100 million operations per second):
  - $O(N^2)$ — DP?
  - $O(N \log N)$ — sorting, divide and conquer?
- Limits on parameters:
  - Small — DP?
  - Large — “direct” approach

Problem Solving

- Admissible time complexities (~100 million operations per second):
  - $O(N^2)$ — DP?
  - $O(N \log N)$ — sorting, divide and conquer?
- Limits on parameters:
  - Small — DP?
  - Large — “direct” approach

Speed Ups

- Faster data structures (know the STL)
- Precomputation
- Relations between values
- Pruning

Questions

?