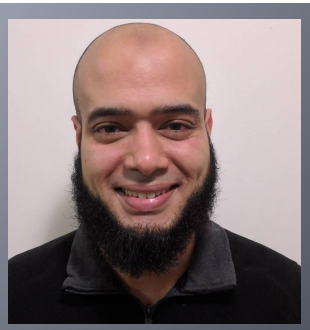
Competitive Programming

From Problem 2 Solution in O(1)

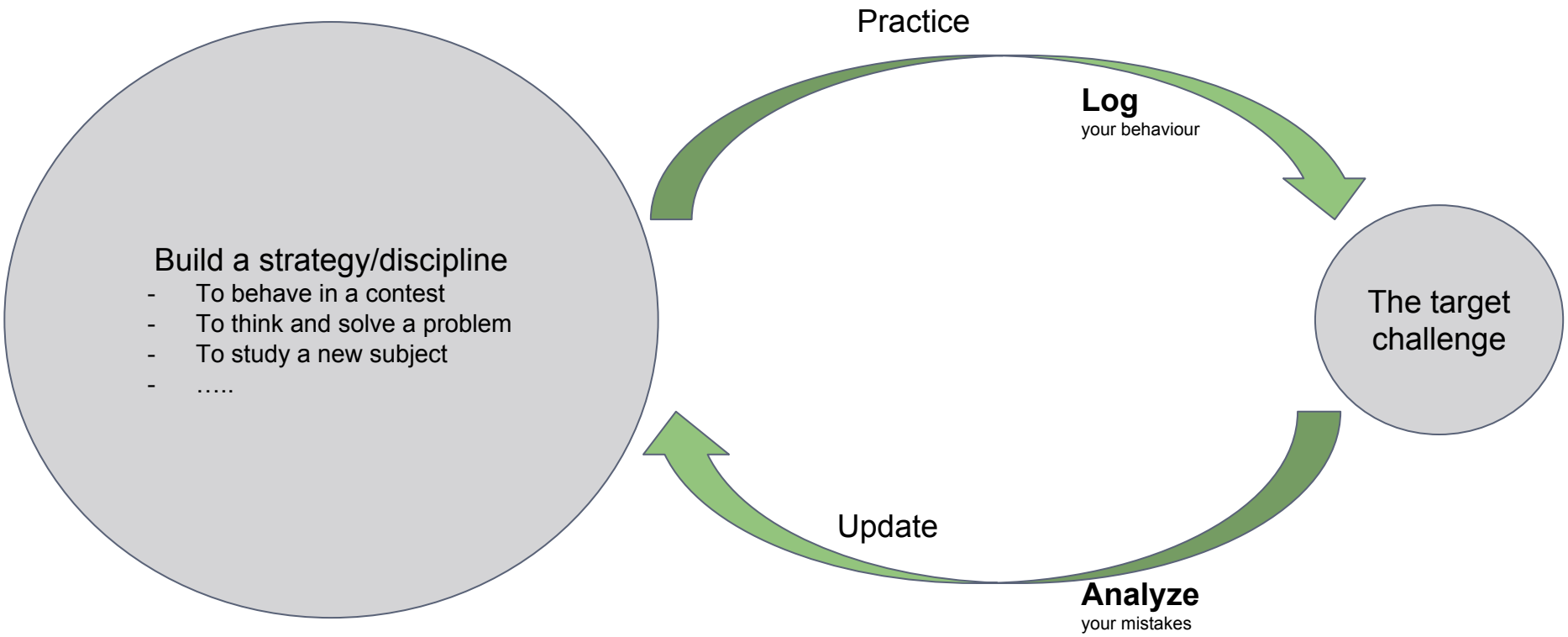
ACM ICPC Training

The Contest - 1 - Concerns

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University



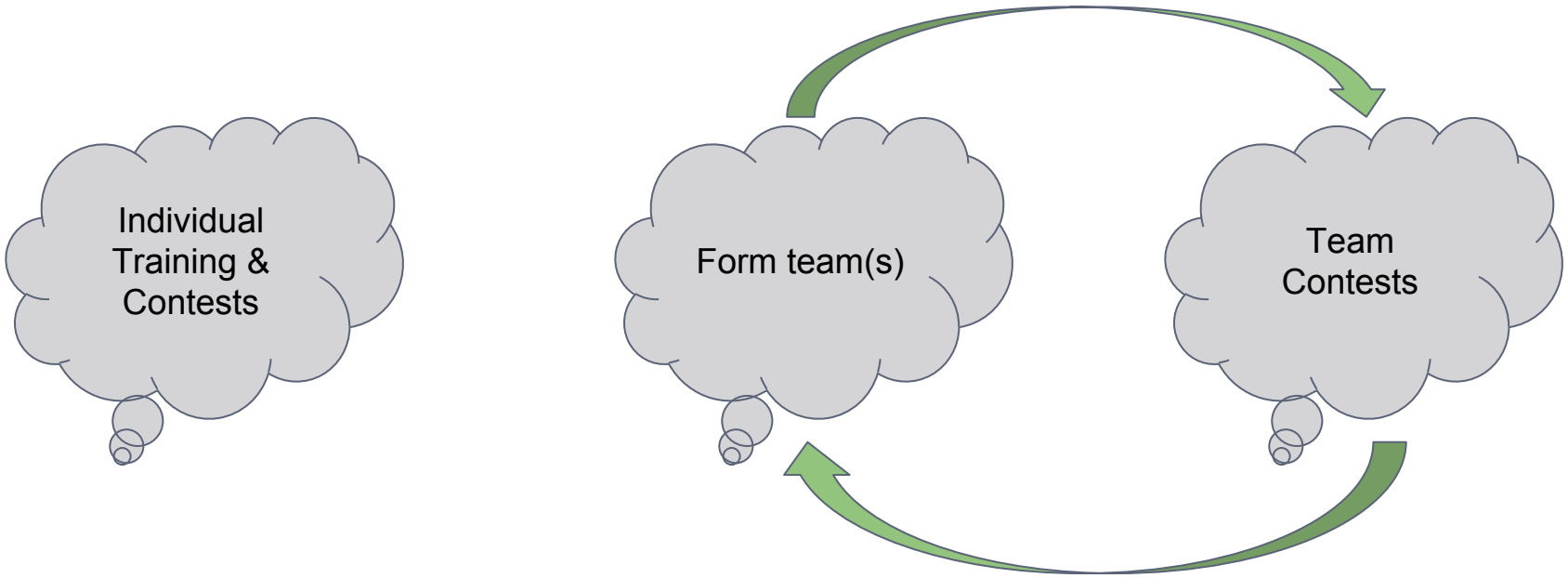
Great keys to *big* improvements



Practice makes perfect

Practice Smart … not Hard

Preparations: Yearly phases



Individual training & Contests

- ~8 months individual phase
- Develop a plan for improving your knowledge & skills
- Develop an individual contest strategy
- Participate in all online competitions (CF, TC, UVA, GCJ …)
- Do individual contests against guys from your region
- Get the problems in the right order as possible
- Get them first submission
- Get them using the minimum debugging time
- Learn how to make use of the scoreboard (don’t be mislead)
- Encourage colleges from your school to participate as possible

Building an effective team

- Recognize the power of teamwork
- Target building the right team
- What might be a good team? Think relative to your abilities
- Talk with team members: Their goals, dedication, knowledge and skills
- Select who can compliment the team missing skills
- Try several teams configuration, Identify pros & cons of them
- Monitor your team members progress (vs their promises)

Contestants Characteristics

- Creativity
- Passionate
- Self-discipline
- Self-Control
- Listen **vs** just care about his/her ideas?
- Individuality **vs** group thinking
- Critical thinking: To idea / code / test case

Team Contests

- 4 months team phase
- Develop a strategy that fits with your team skills
- Apply...Discuss Behaviour...Analyze...Update your strategy
- Train on the right contest **trend/style**
  - ECPC != JCPC != ACPC != NEERC != ICPC != ….
  - Different knowledge topics / problems styles & levels
- Be ready for the potential scenarios during the contest
  - Multiple Problem are WA so far (3-5 submissions)
  - Stuck in remaining problems
  - Weird scoreboard
- Pair training on hard contest problems

Strategy Adaptation

- There is **No** One-Size-Fits All
  - Team skills vary
  - Adjust the strategy to fit with your team
  - Apply your strategy:
  - 1) Fix whatever missing scenarios
  - 2) Tune it to team skills
  - Discuss your team behaviour during the contest with your coach..take notes and make use of them

Computer time: 5h vs 15h

- 1 shared computer + 3 contestants
  - "*Teams which fight for the terminal go nowhere*", Skiena
  - If every one almost used the computer when needed it, it is virtually available **3 x 5** ***hours*** during the contest
  - But if ***fighted*** to take chance in coding your problem = 5h
  - One of the ultimate goals of a strategy is to control the computer time to maximize the team benefits
    - To avoid contestant thinks about code blocks during coding
    - To avoid coding an algorithm that even doesn't work over the samples
    - To avoid coding an algorithm that even won't fit in time/memory
    - To avoid debugging on the computer for long times
    - Learning to code on papers a ***complete code*** is a great skill
    - Putting strict rules on ***debugging time*** is important
    - Some flexibility might be needed with the easiest 1-3 problems

Computer time: 5h vs 15h

- Think on papers NOT on PC
  - Force rules to protect the PC time.
  - Except the first easiest 1-2 problems, ***complete*** **code & thinking on papers**
  - **An exception might be for a very short problem**
  - PC is only for copying code from papers
  - PC is only for debugging
  - Force a time limit for debugging on PC (e.g. 10 min)
  - If more than 10 minutes are needed and some one needs to code, print code and debug on papers (need to print outputs too?). Priority for new codings.

Team Spirit

- Form a team of a **Challenging Spirit**
- Handling Members Mistakes
  - Avoid the mistakes: We agreed on a strategy, **respect** it
    - E.g. We agreed to **trace the sample** and make sure algorithm is ok with them
    - Coding the algorithm to tell us: it is wrong and won't work on samples = !ok
  - Leader: Create and sustain a *no-blame culture*
    - Don't let your emotions control you
    - **After** the contest: analyze the mistakes, avoid its reasons & update the strategy
    - Repeating mistakes? No self-discipline? Think about a **new team configuration**
- The last minutes in a contest
  - Some seniors put their eyes on the contest cup
    - If they got the impression they are not the first, they almost stop competing
    - Learn how to compete till the last minute, under pressure. Watch1 Watch2
    - **Hamza Darwish** team in 2007 regionals: 2 AC / some WAs before the blind hour, **won** the ACPC with 7 AC problems (14, 214, **243, 247, 272, 283, 298**)

Ordering the problems

- The best ordering problem theoretically?
  - Purpose: **Lowest completion time** for solving N problems
  - This study highlight several details of this question
  - There is not only ***no one best strategy***, but there is no polynomial time algorithm for finding the best strategy
  - Tie-break score is **minimal** when problems are solved in order of **increasing** complexity.
  - To win by problems solved => use **reverse** order of ease
  - To win tie-breaks => use order of ease
  - In real contests, many **factors** affect team's behaviour!

Ordering the problems

- Problems levels
  - E.g. in ACPC: (2, 3, 4, 5, 6, 6, 6.5, 6.5, 7, 7, 7.5, 8) / 10
  - E.g. in WF: (5, 6, 6.5, 6.5, 7, 7, 7.5, 7.5, 7.5, 8, 8.5, 9) / 10
  - Let's target getting the problems in **ease** order
  - *E.g. if your 2nd solved problem in ACPC of level 5/10, then you worked on the wrong problem!*
- Ordering
  - Initially, sort the problems based on their **text length**
    - Or one from the back & the other from front. Or read in random order
  - Be careful from Scary, but doable problems
    - Easy problem with **weird/scary figures/outputs**. Example
    - Very **lengthy** problems, which might be direct **simulation** problem!

Skimming the problems

- Skimming is a reading skill.
  - Here we means, read in a fast manner, don't focus with every tiny details (but u feel it won't matter)
    - This way, you can read a lengthy problem statement and discover it is direct simulation, better than reading it after 3.5 hours
  - Get a sense of the problem level (experience matters)
    - Some are clearly hard, some are clearly easy, some are tricky
    - Delay the hard as soon as possible.
  - If noticed a problem much easier than what is in coding now, then we may swap the solving
  - Again, remember. We want to find the problems and solve them as **soon** as possible in **the right order**
  - Once selected a problem to solve, read carefully

Reading the whole problems-set

- When to read all the problems?
  - It is important to read all the problems! *But when*?
  - Read all the problems within the **first 1.5 hours.**
  - Another one is exploring the problems based on the scoreboard status. You need to be very careful!
    - Whatever **unread by you, but solved** problem => Read it
    - You should decide if scoreboard is **biased enough** toward the next problem or you should read some unread problems
- Unclear problem text?
  - Most of the time, the text is enough. It just maybe not well written. Still ambiguous? **Don't debate** with your team members. **Send (<u>early</u>) a question** to the judge

The scoreboard

- Multiple submissions before AC problems
  - Sometimes a problem is AC but other teams needed 2-4 submissions. Be very careful with it.
  - Confusing problem text. Unclear constraints (Don't assume). Tricky test cases. Tight time limit?...
- What is the level of teams solved problem?
  - Juniors? Semi-seniors? Seniors teams?
  - Problem solved by strong teams only can be very tricky!
  - What if a problem seems hard, but AC with a junior team?
  - Then **think trivially** in a solution (or relax what prevents you, e.g. such as a scary time complexity)
    - Potential scenario: Juniors sovled this hard problem before in a campus

The scoreboard

- Our position vs others
  - Select time points to revise situation (e.g. 1.5 3 4 hours)
  - Be aware of your **current position**..and the **target** one
  - Decide what to do to **reach** your target
  - E.g. in the 3rd hour, 1 member work on 1 hard problem for next 2 hours, and others pair
  - Think in the maximal penalties to avoid reaching
  - If you mayn't reach your target position based on penalty, but # of accepted problems, you maybe ok with ***free submit*** mood
  - Free submit: Either in last 40 minutes or if you are performing so bad since the contest begin (hamza 07)

Compete against the problem set

- Problem set vs the scoreboard
  - Many guys have their eyes on other teams **all the time**
  - They just think lose/win & get nervous because of that
  - Most of time, one should compete against the problemset
  - Scoreboard is **only guide** for problems levels
  - You should **avoid** everything that puts pressure on the team or raise up negative feelings
  - Maybe in the last 1.5 hours you study your level with above teams to decide what to do to reach what

Preparing the code template

- Write a template
  - There should be some template that u guys **agree on**
  - E.g. in C++ some #defines / includes / freopen
  - Once contest started, let the fastest ones writes down those templates (e.g. for C++ / Java)
  - Makesure **all** team members get use to the same one
  - Then for every problem, just make copy of the template
  - And save much for your time
  - It also helps in code reviewing!

Working as pairs or triples

- Tackling hard problems
  - At some point, the next problems are hard for 1 member
  - Then you switch to 2-3 persons per a problem
  - Think together to get the main picture of the solution
  - Decide how to split the work.
  - It can be: we will do everything together (safer, longer)
  - Or divide the work as much as possible
    - E.g. one write the IO / Sample file / data structures / main functions to call / write from library. Other 2 members agree on the algorithm further details or split the details to 2 parts

Reviewer role

- Two Eyes are Not Enough
  - So you read a problem and found a solution!
  - What if you are wrong? Or missing a hidden case?
  - Another reviewer can help strongly
  - Force a rule: Before coding a problem, a reviewer must first listen to your problem summary and verify the solution (idea, complexity, potential cases).
  - If approved but later a WA. Reviewer read text by himself
  - Still WA? the 3rd member read the problem by himself (no summary) before helping.

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً