Problem Strategy

4 easy steps to solving any problem!

Game Plan

- Sportsmen use it, so why not us?
  - In good times and bad, know what to do
  - Focus under pressure
  - Use time efficiently

Game Plan

- Besides solving the problems,
  - Note tricky details
  - Which problems when?
- Time management
  - Order problems based on required time/difficulty
  - Maximise your score
  - Know when to abandon
    - short time => debug
    - +/- 45 mins => solve a new problem?

Problem Solving

- Analysis
  - Usually the tough part
- Design
  - Describe your solution
- Implementation
  - Coding
- Testing
  - Covered Later

Analysis

- Get to grips with the problem
  - Understand the given sample
  - Try small cases
  - Ask questions within allotted time
  - Problem solving frameworks
- Brainstorm solutions
- Focus on constraints & limits
  - Memory/Time Complexity
- Try to break your solution
  - Degenerate, huge, tiny cases
  - Each part individually, and collectively

Brainstorming solutions

- Consider brute force first
  - Exhaustive/complete search
  - Try every combination
- Easy to think of, code & debug
- Usually too slow
  - Probably won't meet time/space limits
  - Still useful:
    - Can give you ideas
    - Use for future testing

Focus on Limits

- You can solve the problem using brute force
  - Now need a more efficient solution
- 10 or 100 mil operations per second
  - N < 1000 → O(N^2) is ok
  - $2^{10} \sim 1000$
- Goal is not to solve the problem, but to solve it within the constraints
  - Optimisation/New solution
- The limits may be informative

Brainstorming solutions

- Heuristics & Approximations
  - Sometimes Greedy can be proven correct
- Extend strategy from small cases
- Relate to a similar problem already seen
- There are only about 16 basic types of informatics olympiad problems (see USACO)

Problem Solving Paradigms

- Generating vs Filtering
  - Filters are easier but runs slower
- Forward vs Backward
  - Sometimes easier to suppose a solution and work backward (reverse engineer)
  - Sometimes look at things from a different direction, e.g. process data in reverse order

Techniques

- Precomputation
  - Compute everything once at the start – enables faster lookups later
  - Can reduce time complexity
- Exploit symmetry
  - Solve a fraction of the problem
- Rephrase/Simplify
  - To a problem you already know
  - To a problem that's easier to think about
    - Often Graphs

Techniques

- Decomposition
  - Break the problem into smaller parts
    - Not only smaller sub-problems (recursion, DP)
    - But also different parts of a single problem
  - Problems can contain components of each of the 16 types
- Proof Techniques
  - By contradiction
  - Induction
  - Etc.

Design

- Spending some time planning your solution can speed up the implementation
- Also identify logic errors

- Always choose easiest solution

Implementation

- Waste memory/time if it makes things easier
- Make code easy to debug
  - Whitespace
  - Comments
  - Meaningful variable names
  - Avoid pointers, dynamic memory, floating point

How to Get Better

- Learn & Practice more
  - Applying knowledge easier than inventing
  - Exposure to useful ideas
  - Recognising when you can apply techniques

Subset Sums

- Question from yesterday:
- How many ways can {1,2,…,N} be split into 2 partitions with equal sums?
- $N<50$
- E.g. $N=3$: Answer = 1 ({1,2} and {3})

Subset Sums: Analysis

- Get to grips
  - Understand the sample input/output
  - Do some small cases
  - Note anything important
- Focus on constraints
  - N < 50
    - O(N^3), maybe O(N^4) etc.
    - DP? (also unnecessary info – the path)

Subset Sums

- Brainstorm solutions
  - Brute Force
    - Generate/Filter every possible partitioning, sum each partition
    - Each element in first or second set
    - $O(2^N \times N)$
  - DP/Recursion?
    - Identify the state & recurrence relation
      - Reverse engineering is useful here: suppose a solution, and see what that implies for the smaller case (N→N-1)
    - Etc.

Subset Sums

- A DP solution exists
  - Is it good enough? Do the math
  - O(N^3) with N<50 is fine
- Check degenerate/small/large cases, the sample input/output, etc.
- Design, Implement, Test