Lecture 10
Contest Strategy

Jingbo Shang & Zhengkai Wu
University of Illinois at Urbana-Champaign

Nov 3, 2017
Outline

◊ Competing with a single computer
◊ Triaging a problem set
  ▪ Complexity & input bounds analysis
◊ What to do during the practice contest
◊ Solving & debugging on paper
◊ Creating test cases
◊ Reference materials
Competing with a single computer

- During an ICPC contest, you will have one computer per team of 3 students
- Not everyone can code at the same time!
- How do you use the computer wisely?
Popular Strategies

◊ 3 DPS’s
  ▪ Take turns on the computer, based on how close you are to solving a problem.
  ▪ “Shortest Job First” scheduling.

◊ 2 DPS’s + 1 Support (recommended, more robust)
  ▪ Two people take turns for coding.
  ▪ One person focuses on thinking and debugging.
  ▪ Used in many top world finals teams.

◊ 1 DPS’s + 2 Support
  ▪ Maybe too tired for 5 hours.
Tips

◊ One person should be coding
  ▪ The total amount of “machine time” is fixed, i.e., 5 hours.
◊ Others should be solving problems/debugging on paper
  ▪ Maximize the usage of “human time”.
◊ Only code if you think you have already worked out the problem correctly
  ▪ Don’t waste the “machine time”!
Contest strategy

◊ Consider the following problems, with the given times to solve
◊ What is the time penalty for solving in the order:
  ▪ A, B, C?
    \[40 + 50 + 110 = 200\]
  ▪ C, A, B?
    \[60 + 100 + 110 = 270\]
  ▪ B, A, C?
    \[10 + 50 + 110 = 170\]
◊ Obviously, we want to solve the easier problems first

<table>
<thead>
<tr>
<th>Problem</th>
<th>Time to solve</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>40 minutes</td>
</tr>
<tr>
<td>B</td>
<td>10 minutes</td>
</tr>
<tr>
<td>C</td>
<td>60 minutes</td>
</tr>
</tbody>
</table>
Triaging a problem set

- In order to solve the easiest problems first, we need to order the problems by difficulty
- Called “triaging”
- **You should triage the problem set before coding anything!**
  - Otherwise, you might end up working on a harder problem first!
- Triaging effectively comes with practice
Components of triaging

◊ Need to:
  ▪ Identify the problem type (e.g., mathematics, graph, DP)
  ▪ Identify the intended complexity based on input bounds
  ▪ Identify the key algorithm/technique/data structure (e.g., fast exponentiation, minimum spanning tree, hash table)
    • Be able to prove to yourself the correctness of your approach
Complexity & input bounds analysis

◊ Once you have identified the problem type, you must figure out what classes of algorithms will pass
  ▪ Look to the input bounds!

◊ For example, if the maximum value of $n = 10^8$, an $O(n^2)$ algorithm will not pass
  ▪ Even at 10 GHz ($10^{10}$ operations per second), it would take over a week to complete $n^2 = 10^{16}$ operations!
Time complexity

- Rule of thumb: can perform $\sim 10^8$ operations/second
  - The number of operations in your solution should be at most between $10^8$ and $10^9$

- Asymptotically, complexity increases as follows:
  $$O(1) < O(\log n) < O(n) < O(n \log n)$$
  $$< O(n^2) < O(n^k) < O(2^n) < O(n!)$$
  - Practically speaking, for $n \leq 10^5$, $O(\log n) \approx O(1)$
  - Remember, $2^{10} \approx 10^3$

<table>
<thead>
<tr>
<th>Value of $n$</th>
<th>Worst possible complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\geq 10^9$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$10^8$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$10^5 - 10^7$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$10^3 - 10^4$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>$10^2$</td>
<td>$O(n^4)$</td>
</tr>
<tr>
<td>50</td>
<td>$O(n^5)$</td>
</tr>
<tr>
<td>25</td>
<td>$O(2^n)$</td>
</tr>
<tr>
<td>20</td>
<td>$O(n^2 2^n)$ (TSP DP solution)</td>
</tr>
<tr>
<td>12</td>
<td>$O(n!)$</td>
</tr>
<tr>
<td>9</td>
<td>$O(n! 2^n)$ (TSP brute force)</td>
</tr>
</tbody>
</table>
Time complexity

- Time complexities for common techniques:
  - Hash table lookup: $O(1)$
  - Binary search: $O(\log n)$
  - Sort: $O(n \log n)$
  - Dynamic programming: usually polynomial (e.g., $O(n^2)$, $O(n^3)$)
  - Gaussian elimination for matrix of size $n \times n$: $O(n^3)$
  - All subsets of size $k$: $O(n^k)$
  - All subsets: $O(2^n)$
  - All orderings/permutations: $O(n!) \gg O(2^n)$
Space complexity

◊ Rule of thumb: can create array of size:
  ▪ \( \sim 10^8 \) using dynamic memory allocation (e.g., `new`, `malloc`)
  ▪ \( \sim 10^6 \) using static allocation (e.g., local variables in C/C++)

◊ Depending on \( n \), you may need to change the data structure you use
  ▪ E.g., adjacency list vs. adjacency matrix when \( n > 10^3 \)
Space complexity

Space complexities for common data structures:
- Linear DS (list, stack, queue, heap, hash table, etc.): $O(n)$
- Most trees (binary search tree, AVL tree, etc.): $O(n)$
- Skip list: $O(n \log n)$
- Segment tree: $O(n \log n)$
- $k$-dimensional array: $O(n^k)$
- Adjacency list: $O(|E|)$
- Adjacency matrix: $O(|V|^2)$
- Number of primes less than $n$: $\pi(n) \approx \frac{n}{\log n - 1}$
Triaging during a real contest

- Triaging takes time, which accumulates in your time penalties
- We want to minimize the time triaging before the first solve, but how?
- Teams have 3 students, remember?
Triaging during a real contest

◊ Every problem set has a ridiculously easy problem
◊ Everyone should first find that problem, then have the fastest coder solve it while the other 2 continue triaging

◊ Problems should then be solved on paper in order of difficulty until the computer is free
  ▪ Switch off as soon as a person solves a problem or gets stuck
◊ The computer should rarely be idle!
Tips

◊ Use a sheet to record all (partially) known problem types
  ▪ Who to do this? The support guy.
◊ Check whether your teammates have read/thought the problem before you read/think it.
  ▪ Avoid double work.
How to practice triaging

◊ Problem sets from previous contests have already been triaged for you!
  • Just look at the number of solves on the final scoreboard 😊
  • Some contests release difficulty rankings in judging notes
    • E.g., our regionals, North American Qualification Contest

◊ Attempt to triage yourself, then compare with the actual difficulties

◊ Everyone’s triage ranking may be different, but likely fairly similar
Questions so far?
Exercise

Triage the provided problem set
Before the contest starts...

◊ You want to:
  ▪ Know the efficiency of the judge machine
    • How many addition operations can you run within a second?
  ▪ Know the memory limits
    • Maximum size of integer array you can allocate on the heap
  ▪ Know the stack size
    • Is tail recursion optimized by the compiler?
    • Maximum size of integer array you can allocate on the stack
    • Maximum depth of head recursive calls you can make
  ▪ Know the priority of judging results
    • If you have a Runtime Error and Wrong Answer, which will you see?
  ▪ Know the maximum size of code you can submit
What to do during the practice contest

◊ In some cases, the judges will specify all of the information on the previous slide
  ▪ Read it carefully!
◊ However, if any of it is not provided, use the practice contest to test the judge environment!
While the contest is running...

◊ You will need to:
  • Solve problems on paper
  • Code them up correctly
  • Make your own test cases and test your solution
  • Debug, potentially on paper
  • Get AC!
Solving on paper

◊ Write pseudocode
  ▪ At least the framework

◊ Identify the key algorithm/technique
  ▪ E.g., dynamic programming, math, simulation
  ▪ Write down the meanings of the important variables

◊ Think about techniques to make your coding faster
  ▪ How to efficiently cover corner cases
  ▪ How to arrange functions and methods
Debugging your own code

◊ On the computer
  ▪ Write down the meanings of the important variables
  ▪ Check whether those variables work properly as expected by printing some intermediate results
  ▪ Scan the code **thoroughly** before submission to make sure there are no typos (e.g., i → j)

◊ On paper
  ▪ Follow the same procedures as for debugging on the computer
  ▪ Run through corner cases and tricky test cases by hand, stepping through the code and keeping track of variable values
Debugging your teammates’ code

- Think about how will you implement the solution to the problem **before** looking at others’ code!
- Figure out whether your overall approach and framework are the same or similar
- Pay more attention to the differences
  - More likely that errors appear at those points
- In general, follow the same techniques as for debugging on paper
Creating test cases: Corner cases

◊ Often the cause of Wrong Answer (WA)
  ▪ Let $n$ be the input
  ▪ Smaller $n$’s, such as 0, 1, 2, 3, might be corner cases

◊ Example: Suppose in your algorithm, you are going to enumerate 3 different points out of $n$ points.

```c
answer = 0;
for (int i = 0; i < n; ++ i) {
    for (int j = 0; j < i; ++ j) {
        for (int k = 0; k < j; ++ k) {
            // Update answer
        }
    }
}
printf("%d\n", answer);
```

In this case, you should be careful when $n = 1, 2$. 
Creating test cases: Extreme cases

◊ Often the cause of Time Limit Exceeded (TLE)/Runtime Error (Segmentation Fault)

◊ Two types:
  1. Write code to generate a large case based on the limits in the problem
  2. Based on your algorithm, figure out the worst-case input (on which your algorithm is most inefficient), which might be different from the large test case
Questions so far?
A note on reference materials

◊ You should take tested implementations of common and useful algorithms & data structures
◊ Should also include miscellaneous like trigonometry identities, common integer sequences, number theory relationships, etc.

◊ Reference materials are for reference, not reading
  ▪ You should be able to flip to the page and apply it immediately, not search through your materials for the right information
◊ Make sure you are familiar with your references!
  ▪ Indexing helps!
Questions?
Problem set

- This week’s problem set will be special
- Triage the 1 problem set we provide
Resources

◊ **Chapter 1** of *Competitive Programming* by Steven Halim
  - We highly encourage you to read the entire chapter!!!

◊ **Chapter 1-2** of *Art of Programming Contest* by Ahmed Shamsul Arefin