**CONTESTS OBSERVATIONS**

The most common approaches to try out for whenever you see a question is –

1. **READ QUESTION IN THE MANNER IT WAS GIVEN IN Geeksforgeeks doc - - HOW TO READ A COMPETITIVE PTOGRAMMING QUESTION**

Always follow the below order to read a question in competitive programming -

1. Test cases

2. Input format

3. Output format

4. Problem statement

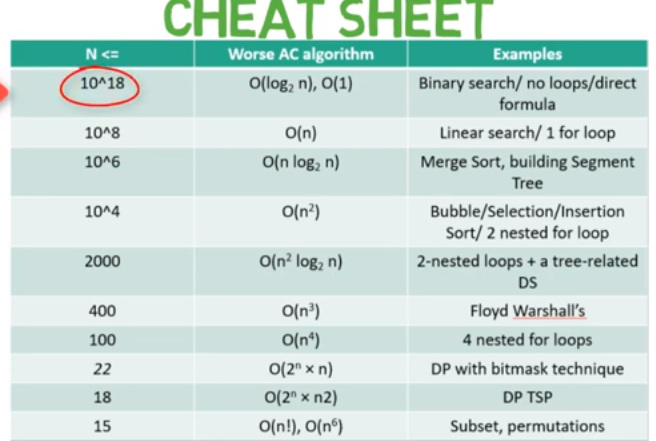
1. **READ QUESTIONS CAREFULLY & WITH TIME –** hard questions in long challenge are nothing but which have a lot of concepts. The have a lot of symbols. **DON’T BE AFRAID OF THE SYMBLOS.** Symbols means that it will take time to understand questions. That’s it. **BE CALMED DOWN. DON’T THINK YOU CAN’T DO THE QUESTION BEFORE UNDERSTANDING IT.**
2. **TRY FINDING PATTERN / OBSERVATION (MOST COMMONLY FROM GETTING ANSWERS TO VARIOUS TESTCASES) –** This is the most common step to solve contest problems. Answers to **Random test cases (for easy questions)** a lot of times show you the **pattern** which helps you to make the observation you need to solve problem. There are some methods for generating answers to random test cases which are –
3. **REASEARCH TECHNIQUES FOR INSPIRATION –** if you can’t find any pattern and have less idea about the technique you should use then you can use following techniques –
4. Go on GFG and search for pages, by typing keywords of the question you want to solve, in the GFG search bar. For easy to medium questions you will mostly get pages giving you the answer code(s) with explanation of the approach.
5. Mostly you get hints from above but if not then Google zindabad. Try different queries on it. Use keywords of the questions or the main idea of the problem. Look at all the recommendations links given at the end of the page.
6. In an ongoing contest – you can check goggle stats and look for the latest search trends. Most people use google to search for algo implementation. So, you can get to know what algos other people are trying and maybe have some hint at which algo can be used.
7. **HOW TO AVOID TLE -** <https://www.youtube.com/watch?v=S8V444xdNvM>

The main things in this video was -

1. **IF YOUR PROGRAM IS EXCEEDING MORE THAN 108 INSTRUCTIONS IN A SEC THEN IT WILL GIVE TLE -** judge is also a program and has particular capacity to process instructions in a sec. The capacity is 108. But if your program is executing more than 108 in a sec (if it is more than a sec that means the number of instructions will also increase) Then it will give TLE.

**BUT HOW IS THE ABOVE HELPFUL?**

1. **HELP IN CHOOSING THE ALGO –** now we can look at the **constraints** and on the **basis of the value of N (and T also sometime)** we can decide **WHAT SHOULD BE THE COMPLEXITY OF OUR PROGRAM TO BE AC.** The following picture shows it –



1. **INPUT AND OUTPUT –** sometimes it is also about reading input and output fast. Like for c++ - use printf and scanf instead of cin and cout because they are faster. So, look for fastest i/o methods in the language.
2. **SHORT CONTESTS TIPS –**
3. **Look at the input, output and constraints of each problem -** Usually, simple adhoc problems have simple input, output and also the constraints will be reasonable. I try to look at these to find easier problems and usually it works (not always).

In case of **if a problem has no subcase marking (partial grading) then it is the easiest.**

1. **BUGS –** This is the main problem with short contests. A lot of you time goes into debugging the problem. Some tips are -

**YOU SHOULD HAVE A CLEAR IDEA OF WHAT YOU WANT, DON’T HURRY EVEN FOR THE EASIEST OF QUESTIONS - because** If you have a clear idea of what you want to do and a lot of experience implementing algorithms, you will write fewer bugs. And even the bugs that come you will be able to solve them because you have a clear idea of what you want.

**THE ONLY WAY TO AVOID BUGS IS TO CREATE NO BUG -** The most important part of debugging lies before you start debugging"

I'm not kidding, the most important part of debugging is to **write a code that is SIMPLE and READABLE**. I often find code that is not properly indented or too long or **containing many specific conditions**, these are the biggest blockers for debugging.

So instead of putting more effort in debugging bad code, put your effort in writing good code. That gives two advantages, one good coding practice + debugging will become easier.

**READ ‘DEBUGGING TIPS AND WAYS’ FILE FOR MORE**