* Favorite book with physical tabs added to locate the most important parts
* Problems are non-interactive and follow the simple pattern:  read data from standard input, process data, output data to standard output.
* Input data will always have a count at the beginning or a sentinel at the end
* File naming conventions must be followed exactly, if any
* Exact input and output formats are specified
* Judge’s input data is generally much more thorough (borders, min, max, very small/large numbers)
* Think of input data to test with – the sample input is often NOT a complete test
* Debugging info needs to go to the standard *error* stream, not standard output
* The source file name and main class name are all *lower* case
* Team work strategies:
* Splitting the problems up and basically working alone
* Working together
* For input in Java use java.util.Scanner

import java.util.Scanner;

public class B {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

int n = input.nextInt();

* For input in C/C++ use scanf

#include <stdio.h>

int main()

{

int n;

scanf("%d",&n);

* For input in Python use input

n = int(input())

* Know how the basic libraries in your language work for containers (lists, sets, Hash tables), strings, sorting (classes Arrays and Collections in Java), formatting
* Team must have the same coding style / preferred programming Language
* Figure out who is the fastest typist
* Scan the problems and looking for the easiest first
* One person can write the input/output code, and at least the sample input file, while others clarify the algorithm
* Procedural programming using the global data that you input is usually the fastest and most efficient route.
* Popular problem categories
* Translate a simple, straightforward description
* Shortest or longest paths in graphs
* Sorting
* Recursive searches
* Geometrical output (always text, but graphical ideas for positioning parts are likely to come in).
* Problems involving combinations, permutations, or subsets.
* Java and C++ I/O skeletons: Prepare general/standard skeleton for the programming language you use at the competition.
* Some general tasks that you need to do in almost every problem:

1. How to read inputs
   1. String conversion to number`
   2. String conversion to digits
   3. Tokenizing string input
   4. Store into array
2. Sorting inputs based on their values, length, etc.
3. Loops
   1. Suitable loop types (while, for, do): almost every input set includes number of test cases.
   2. Loop termination conditions: In many cases, inputs have some termination values, such as 0 or -1.

Type of inputs:

There are different input formats in ACM problems, and it would be a good idea to have a separate template for every popular input format ready in your preferred language(s). These are some popular input formats:

* One line of unknown number of inputs, or unknown number of inputs, one in each line
* Unknown number of inputs, one in each line and one sentinel value, say 0, for termination of the inputs
* One number in the first line, N, indicating the number of inputs in the second line, and then N numbers of inputs on the second line
* One number in the first line, N, indicating the number of inputs in other lines, and then N numbers of inputs, one in each line
* An unknown number of sequence of inputs like the previous case along with a sentinel value, say 0, for termination of the inputs

 Popular utility methods:

* Maximum/Minimum value in an array
* Sort an array

 Sometimes, you need to do some conversions. Prepare some templates for the following popular tasks:

* String conversion to number
* String conversion to digits
* Tokenizing string input
* Store into array

Time management

Review all the problems first and find simplest one first

Ranges of inputs/outputs

No wrong input, no need to check input types

Prepare algorithms and code for every popular problem/utility, print, and have with you at the contest

Prime Numbers

Prime Factors

Optimization (memoization)

Line intersection