Notice

For all tasks:

- The limits are available in the "Overview" page in the contest system.
- There is an attachment package that you can download from the contest system, containing sample graders, sample implementations, example test cases, and compile and run scripts.
- You may make up to 50 submissions for each task, and you have to submit exactly one file in each submission.
- When testing your programs with the sample grader, your input should match the format and constraints from the task statement, otherwise, unspecified behaviors may occur.
- In sample grader inputs, every two consecutive tokens on a line are separated by a single space, unless another format is explicitly specified.
- When you test your code on your local machine, we recommend you to use scripts in the attachment packages. Please note that we use the `-std=gnu++17` compiler option.
- If you are unable to submit to CMS, you can use the `ioisubmit` tool to store your code for evaluation after the end of the contest.
  - Run `ioisubmit <task_shortname> <source_file>` in directory with `<source_file>`.
  - Ask a committee member to take a picture of the output of `ioisubmit`. Your submission will not be considered unless this step was done.
    - If you are competing online, ask your proctor to take a picture of the output of `ioisubmit` and send it to the organizers.

Convention

The task statements specify signatures using generic type names `void`, `bool`, `int`, `int[]` (array), and `union(bool, int[])`.

In C++, the graders use appropriate data types or implementations, as listed below

void	**bool**	**int**	**int[]**
`void`	`bool`	`int`	`std::vector<int>`

union(bool, int[])	**length of array a**
`std::variant<bool, std::vector<int>>`	`a.size()`

In C++, `std::variant` is defined in the `<variant>` header. A method with the return type `std::variant<bool, std::vector<int>>` can return either a `bool` or an `std::vector<int>`. The sample code below shows three working examples of functions returning `std::variant`.

```
std::variant<bool, std::vector<int>> foo(int N) {
  return N % 2 == 0;
}
std::variant<bool, std::vector<int>> goo(int N) {
  return std::vector<int>(N, 0);
}
std::variant<bool, std::vector<int>> hoo(int N) {
  if (N % 2 == 0) {
    return false;
  }
  return std::vector<int>(N, 0);
}
```