Technical Slide

1 Lesson 1: Testing

Video 1.1: Testing, sample tests, min/max tests

Video 1.2: Custom cases and testing workflow

Video 1.3: Stress-testing

■ Run your program locally on some inputs

- Run your program locally on some inputs
- Incorrect attempts are penalized

- Run your program locally on some inputs
- Incorrect attempts are penalized
- You need a test for debug

- Run your program locally on some inputs
- Incorrect attempts are penalized
- You need a test for debug
- In this lesson:
 - Common types of test cases
 - Testing workflow
 - Stress-testing

■ Correctness: compare your output with the correct answer

- Correctness: compare your output with the correct answer
- Need to know the answer get it manually or otherwise

- Correctness: compare your output with the correct answer
- Need to know the answer get it manually or otherwise
- Reliability: make sure that your program doesn't crash

- Correctness: compare your output with the correct answer
- Need to know the answer get it manually or otherwise
- Reliability: make sure that your program doesn't crash
- Asserts help check invariants without correct answer

- Correctness: compare your output with the correct answer
- Need to know the answer get it manually or otherwise
- Reliability: make sure that your program doesn't crash
- Asserts help check invariants without correct answer
- Limits: check working time and memory on large inputs

- Correctness: compare your output with the correct answer
- Need to know the answer get it manually or otherwise
- Reliability: make sure that your program doesn't crash
- Asserts help check invariants without correct answer
- Limits: check working time and memory on large inputs
- Locally detailed information on perfomance

■ Always are given, with the answer

- Always are given, with the answer
- Test your understanding of the statement

- Always are given, with the answer
- Test your understanding of the statement
- You could've gotten it wrong

- Always are given, with the answer
- Test your understanding of the statement
- You could've gotten it wrong
- Test your solution before implementing

- Always are given, with the answer
- Test your understanding of the statement
- You could've gotten it wrong
- Test your solution before implementing
- Save time by realizing you're wrong earlier

- Always are given, with the answer
- Test your understanding of the statement
- You could've gotten it wrong
- Test your solution before implementing
- Save time by realizing you're wrong earlier
- Samples check general correctness and sometimes special cases

- Always are given, with the answer
- Test your understanding of the statement
- You could've gotten it wrong
- Test your solution before implementing
- Save time by realizing you're wrong earlier
- Samples check general correctness and sometimes special cases
- Do not rely on samples only!

■ Test of minimal size/minimal input values

- Test of minimal size/minimal input values
- Given: integer N ($1 \le N \le 10^6$), then a sequence of N nonnegative integers, each not greater than 10^9

- Test of minimal size/minimal input values
- Given: integer N ($1 \le N \le 10^6$), then a sequence of N nonnegative integers, each not greater than 10^9

 - U

- Test of minimal size/minimal input values
- Given: integer N ($1 \le N \le 10^6$), then a sequence of N nonnegative integers, each not greater than 10^9

Often is "special"

- Test of minimal size/minimal input values
- Given: integer N ($1 \le N \le 10^6$), then a sequence of N nonnegative integers, each not greater than 10^9
 - Λ
 - 0

- Often is "special"
- Easy to construct

- Test of minimal size/minimal input values
- Given: integer N ($1 \le N \le 10^6$), then a sequence of N nonnegative integers, each not greater than 10^9
 - 1
 - 0

- Often is "special"
- Easy to construct
- Something else could be minimized, e.g. answer size

■ Maximal size/maximal input values

- Maximal size/maximal input values
- Given: integer N ($1 \le N \le 10^6$), then a sequence of N nonnegative integers, each not greater than 10^9
 - 1000000
 - 1000000000 1000000000 1000000000 ...

- Maximal size/maximal input values

Hard to compute the answer

- Maximal size/maximal input values

- Hard to compute the answer
- Checks crashes (e.g. array sizes)

- Maximal size/maximal input values
- Hard to compute the answer
- Checks crashes (e.g. array sizes)
- TL/ML but max time not always on any max size test

- Maximal size/maximal input values
- Given: integer N ($1 \le N \le 10^6$), then a sequence of N nonnegative integers, each not greater than 10^9 1000000 100000000 1000000000 . . .
- Hard to compute the answer
- Checks crashes (e.g. array sizes)
- TL/ML but max time not always on any max size test
- Integer overflow if negative answer when should be nonnegative

How to obtain max test

How to obtain max test

Generate by another program

```
1 int n = 1000000;
2 cout << n << '\n';
3 for (int i = 0; i < n; ++i) {
4   cout << int(1e9) << 'u';
5 }
```

How to obtain max test

Generate by another program

```
int n = 1000000;
cout << n << '\n';
for (int i = 0; i < n; ++i) {
   cout << int(1e9) << 'u';
}</pre>
```

Plug in inside your code

```
1 int n;
2 //cin >> n;
3 n = 1000000;
4 for (int i = 0; i < n; ++i) {
5     //cin >> a[i];
6     a[i] = int(1e9);
7 }
```

```
    Better to have special function for reading data,

   to replace it as a whole
   void readInput() {
        cin >> n;
3
        for (int i = 0; i < n; ++i) {
            cin >> a[i];
5
6
   void setInput() {
8
        n = 1000000;
9
        for (int i = 0; i < n; ++i) {
10
            a[i] = int(1e9);
11
12
13
        main() {
   int
14
        //readInput();
        setInput();
15
16
```

Technical Slide

1 Lesson 1: Testing

Video 1.1: Testing, sample tests, min/max tests

Video 1.2: Custom cases and testing workflow

Video 1.3: Stress-testing

Specific problem types

String problems aaaaaa abcdef

Specific problem types

- String problems aaaaaa abcdef
- Problems about divisibility prime numbers, numbers with many divisors
 2, 3, 11, 31, 997, 10⁹ + 7 are prime
 48 has 10 divisors, 931 170 240 has 1344

Specific problem types

- String problems aaaaaa abcdef
- Problems about divisibility prime numbers, numbers with many divisors
 2, 3, 11, 31, 997, 10⁹ + 7 are prime
 48 has 10 divisors, 931 170 240 has 1344
- Graphs, geometry, . . .

Program structure

Test all branches in your code

```
1 if (condition) {
2     ...
3 } else {
4     ...
5 }
```

Include test with condition true, and condition false

Program structure

Test all branches in your code

```
1 if (condition) {
2     ...
3 } else {
4     ...
5 }
```

Include test with condition true, and condition false

■ Different answer types (YES/NO, -1 for there is no answer, etc)

Program structure

Test all branches in your code

Include test with condition true, and condition false

- Different answer types (YES/NO, -1 for there is no answer, etc)
- Test different parts separately, each right after it's finished

Custom tests

■ Make "interesting" tests — but note that they are not necessarily interesting for your solution

Custom tests

- Make "interesting" tests but note that they are not necessarily interesting for your solution
- Test different run patterns, special cases, pathological cases — depends on the solution and its proof

Custom tests

- Make "interesting" tests but note that they are not necessarily interesting for your solution
- Test different run patterns, special cases, pathological cases — depends on the solution and its proof
- Combine all of the above

Testing stages

Testing stages

■ Before submission — to not waste attempts

Testing stages

- Before submission to not waste attempts
- 2 After submission to find a test case for debugging

■ Time is limited, so there is always a trade-off between "test well" and "test fast"

- Time is limited, so there is always a trade-off between "test well" and "test fast"
- Depends on the rules

- Time is limited, so there is always a trade-off between "test well" and "test fast"
- Depends on the rules
- Depends on complexity and how sure you are in your solution

- Time is limited, so there is always a trade-off between "test well" and "test fast"
- Depends on the rules
- Depends on complexity and how sure you are in your solution
- Always check on samples that your program works at all, and that the format is correct

- Time is limited, so there is always a trade-off between "test well" and "test fast"
- Depends on the rules
- Depends on complexity and how sure you are in your solution
- Always check on samples that your program works at all, and that the format is correct
- Nearly always test on cases other than samples

 After fixing a bug, start testing from the beginning

- After fixing a bug, start testing from the beginning
- Save all tests you've come up with so you don't need to invent them again

- After fixing a bug, start testing from the beginning
- Save all tests you've come up with so you don't need to invent them again
- Check on all your tests on one run

- After fixing a bug, start testing from the beginning
- Save all tests you've come up with so you don't need to invent them again
- Check on all your tests on one run
 - All tests are saved in one file one after another, and your program solves input cases repeatedly until the end of file, not just one test

- After fixing a bug, start testing from the beginning
- Save all tests you've come up with so you don't need to invent them again
- Check on all your tests on one run
 - All tests are saved in one file one after another, and your program solves input cases repeatedly until the end of file, not just one test
 - Tests are saved with special extension (e.g. 01.in, 02.in, ...), and you have a script to run your program on all files with it (like *.in)

- After fixing a bug, start testing from the beginning
- Save all tests you've come up with so you don't need to invent them again
- Check on all your tests on one run
 - All tests are saved in one file one after another, and your program solves input cases repeatedly until the end of file, not just one test
 - Tests are saved with special extension (e.g. 01.in, 02.in, ...), and you have a script to run your program on all files with it (like *.in)
 - Use some unit-testing software to manage tests, like JUnit

Technical Slide

1 Lesson 1: Testing

Video 1.1: Testing, sample tests, min/max tests

Video 1.3: Stress-testing

■ You can make the computer invent tests for you!

- You can make the computer invent tests for you!
- Write a generator program, which outputs some random input

- You can make the computer invent tests for you!
- Write a generator program, which outputs some random input
- Repeatedly:
 - Generate a random input
 - Run your solution on it
 - Check if the output is correct
 - If not stop and output the test case

- You can make the computer invent tests for you!
- Write a generator program, which outputs some random input
- Repeatedly:
 - Generate a random input
 - Run your solution on it
 - Check if the output is correct
 - If not stop and output the test case
- Fully automated, thousands tests per second!

No need if only interested in crashes, so utilize asserts

- No need if only interested in crashes, so utilize asserts
- *Trivial* solution as simple as possible

- No need if only interested in crashes, so utilize asserts
- *Trivial* solution as simple as possible
- It's correct, and maybe slow so small tests

- No need if only interested in crashes, so utilize asserts
- *Trivial* solution as simple as possible
- It's correct, and maybe slow so small tests
- Obtain the correct answer via the trivial solution

- No need if only interested in crashes, so utilize asserts
- *Trivial* solution as simple as possible
- It's correct, and maybe slow so small tests
- Obtain the correct answer via the trivial solution
- Tests outputs for equality or use custom *checker*

- No need if only interested in crashes, so utilize asserts
- *Trivial* solution as simple as possible
- It's correct, and maybe slow so small tests
- Obtain the correct answer via the trivial solution
- Tests outputs for equality or use custom *checker*
- In total a small version of a testing system

Generators

■ Generate small tests — faster (esp. for trivial solution), easier debuging

- Generate small tests faster (esp. for trivial solution), easier debuging
- Make parameters to easily tweak test size

- Generate small tests faster (esp. for trivial solution), easier debuging
- Make parameters to easily tweak test size
- Example:
 - Fails only on aaaaa, zzz, ...

- Generate small tests faster (esp. for trivial solution), easier debuging
- Make parameters to easily tweak test size
- Example:
 - Fails only on aaaaa, zzz, ...
 - Random 'a'-'z' strings of length 10: probability of $26^{-9} \simeq 2 \cdot 10^{-13}$

- Generate small tests faster (esp. for trivial solution), easier debuging
- Make parameters to easily tweak test size
- Example:
 - Fails only on aaaaa, zzz, ...
 - Random 'a'-'z' strings of length 10: probability of $26^{-9} \simeq 2 \cdot 10^{-13}$
 - Only 'a', 'b', 'c' and length 5: $3^{-4} \simeq 0.01$

- Generate small tests faster (esp. for trivial solution), easier debuging
- Make parameters to easily tweak test size
- Example:
 - Fails only on aaaaa, zzz, ...
 - Random 'a'-'z' strings of length 10: probability of $26^{-9} \simeq 2 \cdot 10^{-13}$
 - Only 'a', 'b', 'c' and length 5: $3^{-4} \simeq 0.01$
- Do not lose generality
 Strings of 'a' far less interesting than strings of 'a' and 'b'

- Generate small tests faster (esp. for trivial solution), easier debuging
- Make parameters to easily tweak test size
- Example:
 - Fails only on aaaaa, zzz, ...
 - Random 'a'-'z' strings of length 10: probability of $26^{-9} \simeq 2 \cdot 10^{-13}$
 - Only 'a', 'b', 'c' and length 5: $3^{-4} \simeq 0.01$
- Do not lose generality
 Strings of 'a' far less interesting than strings of 'a' and 'b'
- Correctly initialize random to get different tests

Stress-test for crashes

```
for ((test=1; test++))
   do
3
       echo Test $test
4
5
       ./generate > in
       ./solution < in > out
6
       if [ $? -ne 0 ]
       then
8
            echo Runtime error
9
            break
       fi
10
   done
```

Terminates on error, so error test is in the in file afterwards

Stress-test for correctness

```
for ((test=1; test++))
   do
3
       echo Test $test
4
5
6
7
8
9
        ./generate > in
        ./solution < in > out
        ./solution trivial < in > ans
        diff out ans
        if [ $? —ne 0 ]
        then
10
            echo Wrong answer
            break
11
        fi
12
13
   done
```

Stress-test after manual testing

- Stress-test after manual testing
- No point if generator/trivial solution/checker is too complex

- Stress-test after manual testing
- No point if generator/trivial solution/checker is too complex
- Start with very small test sizes

- Stress-test after manual testing
- No point if generator/trivial solution/checker is too complex
- Start with very small test sizes
- Couple of minutes running is usually enough

- Stress-test after manual testing
- No point if generator/trivial solution/checker is too complex
- Start with very small test sizes
- Couple of minutes running is usually enough
- While running do something else useful

- Stress-test after manual testing
- No point if generator/trivial solution/checker is too complex
- Start with very small test sizes
- Couple of minutes running is usually enough
- While running do something else useful
- If nothing is found, generate larger tests
 Or rethink the generator

■ Test your solution before and after submitting

- Test your solution before and after submitting
- Start with samples

- Test your solution before and after submitting
- Start with samples
- "Interesting" manual cases min/max, problem type specific, and anything you could imagine

- Test your solution before and after submitting
- Start with samples
- "Interesting" manual cases min/max, problem type specific, and anything you could imagine
- Test different parts separately

- Test your solution before and after submitting
- Start with samples
- "Interesting" manual cases min/max, problem type specific, and anything you could imagine
- Test different parts separately
- If everything else fails, run a stress-test

- Test your solution before and after submitting
- Start with samples
- "Interesting" manual cases min/max, problem type specific, and anything you could imagine
- Test different parts separately
- If everything else fails, run a stress-test
- Watch out for the generator
 Generate small tests