Technical Slide

1. Lesson 1: Programming competitions
    - Video 1.1: Introduction and course structure
    - Video 1.2: Specifics of programming competitions
    - Video 1.3: Problem example
    - Video 1.4: Steps in solving a problem
    - Video 1.5: Soft skills
    - Video 1.6: Competitions review

Why competitive programming?

- Write reliable and efficient programs

Why competitive programming?

- Write reliable and efficient programs
- Learn and practice algorithms

Why competitive programming?

- Write reliable and efficient programs
- Learn and practice algorithms
- Manage time when it's very limited

Why competitive programming?

- Write reliable and efficient programs
- Learn and practice algorithms
- Manage time when it's very limited
- Do well at job interviews

Why competitive programming?

- Write reliable and efficient programs
- Learn and practice algorithms
- Manage time when it's very limited
- Do well at job interviews
- Join the community of highly motivated and smart people

Why competitive programming?

- Write reliable and efficient programs
- Learn and practice algorithms
- Manage time when it's very limited
- Do well at job interviews
- Join the community of highly motivated and smart people
- Have fun :)

In this course

Basic skills and algorithmic ideas

In this course

Basic skills and algorithmic ideas

- Week 1: Programming competitions, testing

In this course

Basic skills and algorithmic ideas

- Week 1: Programming competitions, testing
- Week 2: Code correctness, brute force solutions, running time

In this course

Basic skills and algorithmic ideas

- Week 1: Programming competitions, testing
- Week 2: Code correctness, brute force solutions, running time
- Week 3: Struggles with numbers and how to get unstuck

In this course

Basic skills and algorithmic ideas

- Week 1: Programming competitions, testing
- Week 2: Code correctness, brute force solutions, running time
- Week 3: Struggles with numbers and how to get unstuck
- Week 4: Greedy algorithms, language specifics

In this course

Basic skills and algorithmic ideas

- Week 1: Programming competitions, testing
- Week 2: Code correctness, brute force solutions, running time
- Week 3: Struggles with numbers and how to get unstuck
- Week 4: Greedy algorithms, language specifics
- Weeks 5 and 6: Dynamic programming: edit distance, knapsack, and other common problems

In this course

Basic skills and algorithmic ideas

- Week 1: Programming competitions, testing
- Week 2: Code correctness, brute force solutions, running time
- Week 3: Struggles with numbers and how to get unstuck
- Week 4: Greedy algorithms, language specifics
- Weeks 5 and 6: Dynamic programming: edit distance, knapsack, and other common problems

Programming assignments — just like problems on real competitions

Technical Slide

1. Lesson 1: Programming competitions
   - Video 1.1: Introduction and course structure
   - Video 1.2: Specifics of programming competitions
   - Video 1.3: Problem example
   - Video 1.4: Steps in solving a problem
   - Video 1.5: Soft skills
   - Video 1.6: Competitions review

Competitions

Competitions

- Timed — 2–5 hours

Competitions

- Timed — 2–5 hours
- Individual or team

Competitions

- Timed — 2–5 hours
- Individual or team
- Several problems, solving each adds to the score

Competitions

- Timed — 2–5 hours
- Individual or team
- Several problems, solving each adds to the score
- Solutions are checked by an automated testing system

Algorithmic problems

- Precisely formulated, although often through some real-world legend

Algorithmic problems

- Precisely formulated, although often through some real-world legend
- Input/output exactly of certain format

Algorithmic problems

- Precisely formulated, although often through some real-world legend
- Input/output exactly of certain format
- Have tight time/memory limits

Algorithmic problems

- Precisely formulated, although often through some real-world legend
- Input/output exactly of certain format
- Have tight time/memory limits
- Efficiency is key

Algorithmic problems

- Precisely formulated, although often through some real-world legend
- Input/output exactly of certain format
- Have tight time/memory limits
- Efficiency is key
- Often require knowledge of some algorithms/ideas

Problem solution

- Program in one of the supported languages

Problem solution

- Program in one of the supported languages
- Usually short — a few dozen lines

Problem solution

- Program in one of the supported languages
- Usually short — a few dozen lines
- Reads data in a specific format from standard input/input file

Problem solution

- Program in one of the supported languages
- Usually short — a few dozen lines
- Reads data in a specific format from standard input/input file
- Outputs solution in a specific format to standard output/output file

Problem solution

- Program in one of the supported languages
- Usually short — a few dozen lines
- Reads data in a specific format from standard input/input file
- Outputs solution in a specific format to standard output/output file
- Is repeatedly run on a testing system against prepared test cases

Problem solution

- Program in one of the supported languages
- Usually short — a few dozen lines
- Reads data in a specific format from standard input/input file
- Outputs solution in a specific format to standard output/output file
- Is repeatedly run on a testing system against prepared test cases
- Must not use external libraries, create extra files, go to the network and so on

Testing system verdicts

Testing system verdicts

- CE (Compilation error) — the compiler could not compile your program

Testing system verdicts

- CE (Compilation error) — the compiler could not compile your program
- RE (Runtime error) — your program crashed during the execution

Testing system verdicts

- CE (Compilation error) — the compiler could not compile your program
- RE (Runtime error) — your program crashed during the execution
- TL (Time limit exceeded) — your program didn't exit in the alloted time

Testing system verdicts

- CE (Compilation error) — the compiler could not compile your program
- RE (Runtime error) — your program crashed during the execution
- TL (Time limit exceeded) — your program didn't exit in the alloted time
- ML (Memory limit exceeded) — your program tried to use too much memory

Testing system verdicts

- CE (Compilation error) — the compiler could not compile your program
- RE (Runtime error) — your program crashed during the execution
- TL (Time limit exceeded) — your program didn't exit in the alloted time
- ML (Memory limit exceeded) — your program tried to use too much memory
- PE (Presentation error) — your output was formatted incorrectly

Testing system verdicts

- CE (Compilation error) — the compiler could not compile your program
- RE (Runtime error) — your program crashed during the execution
- TL (Time limit exceeded) — your program didn't exit in the alloted time
- ML (Memory limit exceeded) — your program tried to use too much memory
- PE (Presentation error) — your output was formatted incorrectly
- WA (Wrong answer) — your output is not correct

Testing system verdicts

- CE (Compilation error) — the compiler could not compile your program
- RE (Runtime error) — your program crashed during the execution
- TL (Time limit exceeded) — your program didn't exit in the alloted time
- ML (Memory limit exceeded) — your program tried to use too much memory
- PE (Presentation error) — your output was formatted incorrectly
- WA (Wrong answer) — your output is not correct
- AC (Accepted) — your program passed all tests, congratulations!

Testing system verdicts

- CE (Compilation error) — the compiler could not compile your program
- RE (Runtime error) — your program crashed during the execution
- TL (Time limit exceeded) — your program didn't exit in the alloted time
- ML (Memory limit exceeded) — your program tried to use too much memory
- PE (Presentation error) — your output was formatted incorrectly
- WA (Wrong answer) — your output is not correct
- AC (Accepted) — your program passed all tests, congratulations!

Test cases

- Strictly formatted, no need to process typos, handle possible errors, and so on

Test cases

- Strictly formatted, no need to process typos, handle possible errors, and so on
- Range of possible values for each parameter is given in the statement

Test cases

- Strictly formatted, no need to process typos, handle possible errors, and so on
- Range of possible values for each parameter is given in the statement
- However, you can't assume *anything* about the data, except for what's explicitly stated

Test cases

- Strictly formatted, no need to process typos, handle possible errors, and so on
- Range of possible values for each parameter is given in the statement
- However, you can't assume *anything* about the data, except for what's explicitly stated
- Be sure that problem authors will put test cases of any possible type

  No matter how extreme or nonsensical — only compliance with the statement counts

Test cases

- Strictly formatted, no need to process typos, handle possible errors, and so on
- Range of possible values for each parameter is given in the statement
- However, you can't assume *anything* about the data, except for what's explicitly stated
- Be sure that problem authors will put test cases of any possible type

  No matter how extreme or nonsensical — only compliance with the statement counts
- Usually, to earn score you need to pass *all* tests

Technical Slide

1. Lesson 1: Programming competitions

    Video 1.1: Introduction and course structure

    Video 1.2: Specifics of programming competitions

    Video 1.3: Problem example

    Video 1.4: Steps in solving a problem

    Video 1.5: Soft skills

    Video 1.6: Competitions review

You are given a list of contact names. Order it alphabetically.

Input format

Sequence of contact names, each on a new line. Names are non-empty and consist only of lowercase english letters. Total length of names is no more than 10 000.

Output format

Output given names in alphabetical order — each name on a new line.

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

You are given a list of contact names. Order it alphabetically.

Input format

Sequence of contact names, each on a new line. Names are non-empty and consist only of lowercase english letters. Total length of names is no more than 10 000.

Output format

Output given names in alphabetical order — each name on a new line.

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

You are given a list of contact names. Order it alphabetically.

Input format

Sequence of contact names, each on a new line. Names are non-empty and consist only of lowercase english letters. Total length of names is no more than 10 000.

Output format

Output given names in alphabetical order — each name on a new line.

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

You are given a list of contact names. Order it alphabetically.

Input format

Sequence of contact names, each on a new line. Names are non-empty and consist only of lowercase english letters. Total length of names is no more than 10 000.

Output format

Output given names in alphabetical order — each name on a new line.

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

You are given a list of contact names. Order it alphabetically.

Input format

Sequence of contact names, each on a new line. Names are non-empty and consist only of lowercase english letters. Total length of names is no more than 10 000.

Output format

Output given names in alphabetical order — each name on a new line.

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

Legend

You are given a list of contact names. Order it alphabetically.

- Problem formulation/motivation

Legend

You are given a list of contact names. Order it alphabetically.

- Problem formulation/motivation
- Often long

Legend

You are given a list of contact names. Order it alphabetically.

- Problem formulation/motivation
- Often long
- Look for formal conditions/constraints

Input format

Sequence of contact names, each on a new line. Names are non-empty and consist only of lowercase english letters. Total length of names is no more than 10 000.

- Formal description of test case format

Input format

Sequence of contact names, each on a new line. Names are non-empty and consist only of lowercase english letters. Total length of names is no more than 10 000.

- Formal description of test case format
- Constraints/limits

Input format

Sequence of contact names, each on a new line. Names are non-empty and consist only of lowercase english letters. Total length of names is no more than 10 000.

- Formal description of test case format
- Constraints/limits
- You may assume about tests *only* what's explicitly stated

Input format

Sequence of contact names, each on a new line. Names consist only of lowercase english letters. Total length of names is no more than 10 000.

- Intuition — names are short, real-looking, distinct

Input format

Sequence of contact names, each on a new line. Names consist only of lowercase english letters. Total length of names is no more than 10 000.

- Intuition — names are short, real-looking, distinct
- It's *not* stated that they belong to real people

```
abcdefg
aaaaa
```

Input format

Sequence of contact names, each on a new line. Names consist only of lowercase english letters. Total length of names is no more than 10 000.

- Intuition — names are short, real-looking, distinct
- It's *not* stated that they belong to real people

  ```
  abcdefg
  aaaaa
  ```

- It's *not* stated that names have particular length

  ```
  aaa...aa (letter 'a' 10000 times)
  ```

Input format

Sequence of contact names, each on a new line. Names consist only of lowercase english letters. Total length of names is no more than 10 000.

- It's *not* stated that they are distinct

  ```
  a
  a
  ... (line 'a' 10000 times)
  ```

Input format

Sequence of contact names, each on a new line. Names consist only of lowercase english letters. Total length of names is no more than 10 000.

- It's *not* stated that they are distinct

  ```
  a
  a
  ... (line 'a' 10000 times)
  ```

- Length is *not* necessarily similar

  ```
  aaa..aa (letter 'a' 5000 times)
  a
  a
  ...(line 'a' 5000 times)
  ```

Output format

Output given names in alphabetical order — each name on a new line.

- Your output is tested by a special program — the *checker*

Output format

Output given names in alphabetical order — each name on a new line.

- Your output is tested by a special program — the *checker*
- So you must follow output format closely — otherwise the checker couldn't understand it and wouldn't accept it

Samples

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

- Verify your understanding of the statement with them

Samples

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

- Verify your understanding of the statement with them
- If something doesn’t tie up — reread the statement

Samples

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

- Verify your understanding of the statement with them
- If something doesn't tie up — reread the statement
- And later, check the correctness of your program

Samples

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

- Verify your understanding of the statement with them
- If something doesn't tie up — reread the statement
- And later, check the correctness of your program
- But use other test cases, too!

Samples

Sample input

```
turing
dijkstra
knuth
```

Sample output

```
dijkstra
knuth
turing
```

- Verify your understanding of the statement with them
- If something doesn't tie up — reread the statement
- And later, check the correctness of your program
- But use other test cases, too!
- Samples are usually tested first when you submit

Technical Slide

1. Lesson 1: Programming competitions
   - Video 1.1: Introduction and course structure
   - Video 1.2: Specifics of programming competitions
   - Video 1.3: Problem example
   - Video 1.4: Steps in solving a problem
   - Video 1.5: Soft skills
   - Video 1.6: Competitions review

Steps in solving a problem

Steps in solving a problem

1. Read the statement

Steps in solving a problem

1. Read the statement
2. Formalize it

Steps in solving a problem

1. Read the statement
2. Formalize it
3. Invent a solution

Steps in solving a problem

1. Read the statement
2. Formalize it
3. Invent a solution
4. Prove it

Steps in solving a problem

1. Read the statement
2. Formalize it
3. Invent a solution
4. Prove it
5. Implement it

Steps in solving a problem

1. Read the statement
2. Formalize it
3. Invent a solution
4. Prove it
5. Implement it
6. Test your implementation

Steps in solving a problem

1. Read the statement
2. Formalize it
3. Invent a solution
4. Prove it
5. Implement it
6. Test your implementation
7. Debug if not working

Steps in solving a problem

1. Read the statement
2. Formalize it
3. Invent a solution
4. Prove it
5. Implement it
6. Test your implementation
7. Debug if not working
8. Submit and get AC (hopefully)

What is proving

- Why not just implement an “obviously correct” solution?

What is proving

- Why not just implement an “obviously correct” solution?
- Often solutions base on wrong assumptions

What is proving

- Why not just implement an “obviously correct” solution?
- Often solutions base on wrong assumptions
- Both correctness and efficiency could depend on it

What is proving

- Why not just implement an “obviously correct” solution?
- Often solutions base on wrong assumptions
- Both correctness and efficiency could depend on it
- So if you assume anything, it must be either written in the statement or proven

What is proving

- Why not just implement an "obviously correct" solution?
- Often solutions base on wrong assumptions
- Both correctness and efficiency could depend on it
- So if you assume anything, it must be either written in the statement or proven
- Proving correctness of greedy algorithms and bounds on runnning time in general — later in the course

Fixing a bug

- Say you've found a test case your program isn't working on

Fixing a bug

- Say you've found a test case your program isn't working on
- An error could be on any step

Fixing a bug

- Say you've found a test case your program isn't working on
- An error could be on any step
- So you need to check all of them

Fixing a bug

- Say you've found a test case your program isn't working on
- An error could be on any step
- So you need to check all of them
- If you've found and fixed an error on some step — fix it and then all the following steps one by one

Fixing a bug

- Say you've found a test case your program isn't working on
- An error could be on any step
- So you need to check all of them
- If you've found and fixed an error on some step — fix it and then all the following steps one by one
- Starting from the wrong step could be disastrous

```
...
if n == 5:
    print(42)
...
```

Technical Slide

1. Lesson 1: Programming competitions
   - Video 1.1: Introduction and course structure
   - Video 1.2: Specifics of programming competitions
   - Video 1.3: Problem example
   - Video 1.4: Steps in solving a problem
   - Video 1.5: Soft skills
   - Video 1.6: Competitions review

How to ask for help

- If you're stuck with some problem — you could ask the community for help

How to ask for help

- If you're stuck with some problem — you could ask the community for help
- Where — forum here, forums on popular competitive programming resources

How to ask for help

- If you're stuck with some problem — you could ask the community for help
- Where — forum here, forums on popular competitive programming resources
- Respect competition rules — do not discuss problems from ongoing contest

How to ask for help

- If you're stuck with some problem — you could ask the community for help
- Where — forum here, forums on popular competitive programming resources
- Respect competition rules — do not discuss problems from ongoing contest
- Ask questions well

How to ask for help

- If you're stuck with some problem — you could ask the community for help
- Where — forum here, forums on popular competitive programming resources
- Respect competition rules — do not discuss problems from ongoing contest
- Ask questions well
    - Summarize the issue in the title as good as possible

How to ask for help

- If you're stuck with some problem — you could ask the community for help
- Where — forum here, forums on popular competitive programming resources
- Respect competition rules — do not discuss problems from ongoing contest
- Ask questions well
    - Summarize the issue in the title as good as possible
    - Format your question and code in it well

How to ask for help

- If you're stuck with some problem — you could ask the community for help
- Where — forum here, forums on popular competitive programming resources
- Respect competition rules — do not discuss problems from ongoing contest
- Ask questions well
    - Summarize the issue in the title as good as possible
    - Format your question and code in it well
    - Include just enough code to reproduce the problem

Clarifications

- On competitions, if you don't understand something in the statement, you could ask the jury for a clarification

Clarifications

- On competitions, if you don't understand something in the statement, you could ask the jury for a clarification
- That is, send a specific question about the problem statement, assuming a Yes/No answer

Clarifications

- On competitions, if you don't understand something in the statement, you could ask the jury for a clarification
- That is, send a specific question about the problem statement, assuming a Yes/No answer
- Most probably, the answer is already in the statement

Clarifications

- On competitions, if you don't understand something in the statement, you could ask the jury for a clarification
- That is, send a specific question about the problem statement, assuming a Yes/No answer
- Most probably, the answer is already in the statement
- Questions must be about the problem, not your solution or other ones

Establishing workflow

As time is everything, find a way to do routine things faster/before the contest

Establishing workflow

As time is everything, find a way to do routine things faster/before the contest

- Learn to use specific IDE/text editor, preferably lightweight one — save time on creating new projects, opening new files, debugging

Establishing workflow

As time is everything, find a way to do routine things faster/before the contest

- Learn to use specific IDE/text editor, preferably lightweight one — save time on creating new projects, opening new files, debugging
- Prepare a template code with common includes and so on — to not start from scratch each time

Establishing workflow

As time is everything, find a way to do routine things faster/before the contest

- Learn to use specific IDE/text editor, preferably lightweight one — save time on creating new projects, opening new files, debugging
- Prepare a template code with common includes and so on — to not start from scratch each time
- Backup code versions and tests

Technical Slide

1. Lesson 1: Programming competitions
    - Video 1.1: Introduction and course structure
    - Video 1.2: Specifics of programming competitions
    - Video 1.3: Problem example
    - Video 1.4: Steps in solving a problem
    - Video 1.5: Soft skills
    - Video 1.6: Competitions review

What to do besides this course

- Participate in competitions

What to do besides this course

- Participate in competitions
- Practice solving problems from archives

What to do besides this course

- Participate in competitions
- Practice solving problems from archives
- Learn additional algorithms when you feel the need

ACM ICPC

International Collegiate Programming Contest

ACM ICPC

International Collegiate Programming Contest

- 5 hour contests, teams of three, one computer per team

ACM ICPC

International Collegiate Programming Contest

- 5 hour contests, teams of three, one computer per team
- Result right after submission

ACM ICPC

International Collegiate Programming Contest

- 5 hour contests, teams of three, one computer per team
- Result right after submission
- Ranking by number of accepted problems, on equality — by total *penalty time*

$$\begin{aligned}\text{penalty time} = {} & \text{minute when got AC} \\ & + \text{incorrect attempts} \cdot 20\end{aligned}$$

ACM ICPC

International Collegiate Programming Contest

- 5 hour contests, teams of three, one computer per team
- Result right after submission
- Ranking by number of accepted problems, on equality — by total *penalty time*

$$\begin{aligned}\text{penalty time} = {} & \text{minute when got AC} \\ & + \text{incorrect attempts} \cdot 20\end{aligned}$$

- You must be enrolled in a degree program to participate

ACM ICPC

International Collegiate Programming Contest

- 5 hour contests, teams of three, one computer per team
- Result right after submission
- Ranking by number of accepted problems, on equality — by total *penalty time*

$$\begin{aligned}\text{penalty time} = {} & \text{minute when got AC} \\ & + \text{incorrect attempts} \cdot 20\end{aligned}$$

- You must be enrolled in a degree program to participate
- Onsite contests, multi-tiered: Subregionals, Regionals, World Finals

GCJ, FHC

Google Code Jam
Facebook Hacker Cup

GCJ, FHC

Google Code Jam
Facebook Hacker Cup

- Individual, open for all

GCJ, FHC

Google Code Jam
Facebook Hacker Cup

- Individual, open for all
- Annual, begin with Qualification Round, usually in April (GCJ) and January (FHC)

GCJ, FHC

Google Code Jam
Facebook Hacker Cup

- Individual, open for all
- Annual, begin with Qualification Round, usually in April (GCJ) and January (FHC)
- All rounds except final are online

GCJ, FHC

Google Code Jam
Facebook Hacker Cup

- Individual, open for all
- Annual, begin with Qualification Round, usually in April (GCJ) and January (FHC)
- All rounds except final are online
- You solve the problem, request input and in several minutes need to send output, so not solution itself

GCJ, FHC

Google Code Jam
Facebook Hacker Cup

- Individual, open for all
- Annual, begin with Qualification Round, usually in April (GCJ) and January (FHC)
- All rounds except final are online
- You solve the problem, request input and in several minutes need to send output, so not solution itself
- Used for recruiting

TopCoder

- Regular rounds — Single Round Matches (SRMs)

TopCoder

- Regular rounds — Single Round Matches (SRMs)
- Rating system — after each round your rating changes

TopCoder

- Regular rounds — Single Round Matches (SRMs)
- Rating system — after each round your rating changes
- Challenge phase — you need to come up with a test to fail other people's solutions

TopCoder

- Regular rounds — Single Round Matches (SRMs)
- Rating system — after each round your rating changes
- Challenge phase — you need to come up with a test to fail other people's solutions
- Annual TopCoder Open — multi-tiered championship

Codeforces

- Also regular rounds and rating system

Codeforces

- Also regular rounds and rating system
- Prize rounds with job opportunities by technological companies

Codeforces

- Also regular rounds and rating system
- Prize rounds with job opportunities by technological companies
- Vibrant community — many useful blog posts about competive programming, and a place to ask for help

Other resources

- Codechef — regular rounds and practice problems
- Hackerrank — rounds and challenges, strongly aimed to help companies in recruiting
- Sphere Online Judge — vast problems archive
- CSAcademy — poised for learning
- And many others