Problem Solving

How to Design Algorithms and Solve Exam Problems?

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Questions?

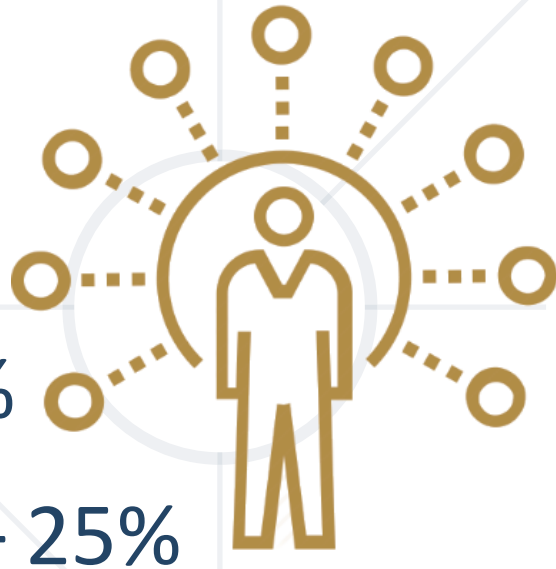
Software University

sli.do

#fund-common

Fundamental Skills of Software Engineers

Skills of the Software Engineers



- 4 main **groups of tech skills**:
  - **Coding** skills – 20%
  - Algorithmic thinking and **problem solving** – 30%
  - Fundamental software development **concepts** – 25%
  - Programming languages and software **technologies** – 25%

Algorithmic Thinking

- **Algorithmic** (engineering, mathematical) **thinking**
  - The ability to analyze problems and find solutions
  - Breaking the problem down to steps (algorithm)
- How to develop algorithmic thinking?
  - Solve 1000+ programming problems
  - It takes 6 to 12 months of coding every day
- Courses in **SoftUni**: Programming Basics, Fundamentals and Advanced Modules
- The programming language doesn't matter!

Tech Problems: Definition and Problem Solving

What is a Tech Problem?

- **Definition** – an **assignment** to design and implement a program, app or software system
  - **Input** data + state, **output** data + state, behavior
- **Goals** – functionality you wish to **implement**
  - Calculate the **output** / implement the behavior
- **Technical difficulties** – barriers, obstacles and limitations to implement the app
  - Technical knowledge, skills and experience

Solving a Problem

- Requires a logical thinking (**algorithmic thinking**)
  - **Define** the problem (software requirements)
  - **Analyse** and understand the problem
  - **Identify** potential solutions (ideas)
  - **Evaluate** and choose a solution (try and test)
  - **Plan** actions (algorithm design)
  - **Implement** the algorithm (coding)
  - **Review** the results (testing)

Tech Problem-Solving Skills

Software University

- Software developers have strong **problem-solving skills**
  - The ability to **think logically** and solve tech problems
  - Math thinking / engineering thinking
  - The ability **analyze** problems and propose **solutions**
  - To design **algorithms** and to implement them
    - **Algorithm** == steps to achieve something
- **Problem-solving** is essential for programming!
  - Solving math / physics problems at school requires similar problem-solving skills

Solving Exam Problems

Tips and Best Practices

Read and Analyze the Problems

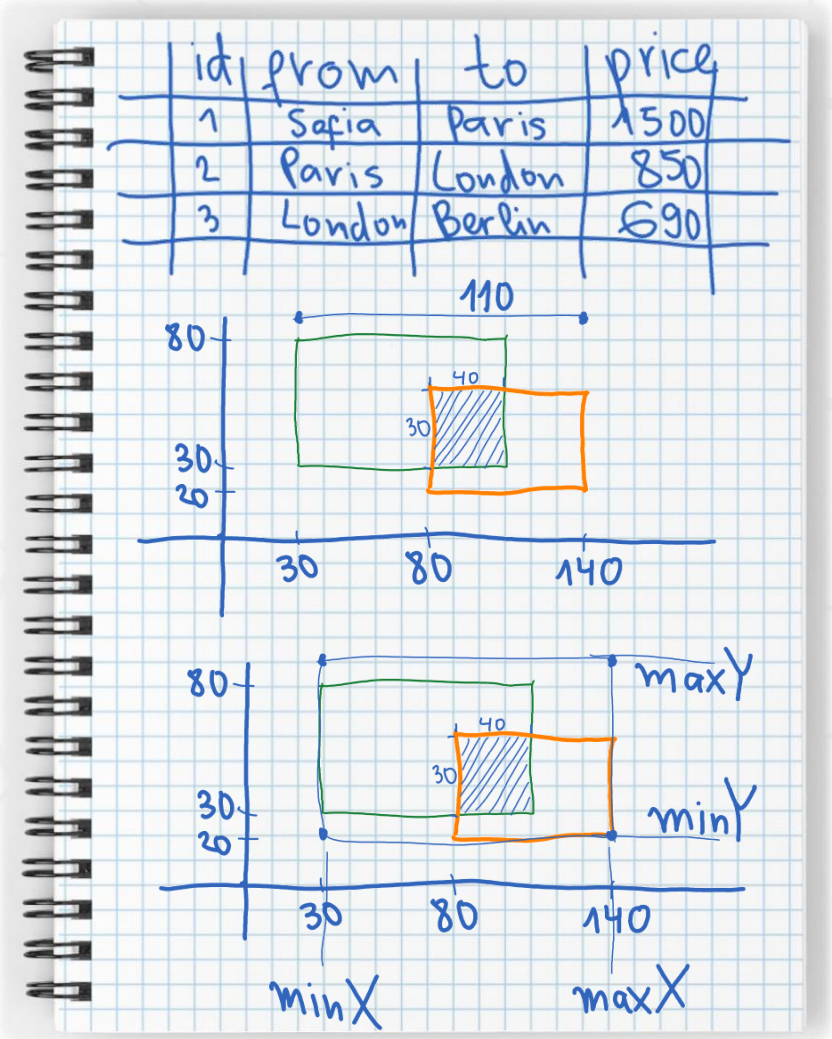
- You are at your "Programming Fundamentals" practical exam
  - You have **3 problems** to solve in **4 hours**
- First **read all the problems** carefully and try to estimate how difficult each one of them is (from your perspective)
  - Read the **requirements**, don't invent them!
- Start solving the **easiest / fastest** to solve problem **first**!
  - Leave the most **difficult / slowest** to solve problem **last**!
  - Approach the next problem when the previous is well tested

Use a Sheet of Paper and a Pen

- Never start solving a problem without a **sheet of paper + a pen**
  - You need to **sketch your ideas**
  - Paper and pen is the best **visualization tool**
    - Allows your brain to **think visually**
  - **Paper works faster** than keyboard / screen
  - Other visualization tools could also work well

Prefer Squared Paper

- Squared paper works **best for algorithmic problems**
  - Easy to draw a table
  - Easy to draw a coordinate system with objects in it
  - Easy to calculate distances
  - Easy to sketch a problem and solution ideas
- Use pens of different colors



Managing Your Time

- At the exam you have **limited time**!
  - Start with the problem, which will take you the **least time**
  - Then, again the problem, which will take you the **least time**
- When you achieve a result of 80/100 or 90/100
  - Think carefully for the **edge cases** → try to handle them
  - After you spend **10-15 minutes** on the last few tests, stop!
- **Don't spend hours** for the last 10% of the tests!
  - Achieving a score of 80-90% of 3 problems is better than 100% of just 1 problem

Typical Mistakes at the Exam

- **Wrong approach #1**: start coding at the first 5 minutes
  - These students have not read the problems (and will fail)
  - They don't start with the easiest problem, but with the first one
- **Wrong approach #2**: don't use pen + paper
  - These students try to invent solutions in their minds
  - Instead of visualizing their ideas on a sheet of paper
- **Wrong approach #3**: debugging in your mind
  - Trying to find the bugs by reading the code
  - This is wrong: you have a debugger in your IDE!

Typical Mistakes at the Exam (2)

- **Wrong approach #4**: spend all the time at the first problem
  - Some students spend 4 hours at the first problem
  - This is wrong: when you spend 1 hour at certain problem, without a significant progress → go to the next problem!
  - You can go back to the first problem, after you solve the others
- **Wrong approach #5**: spend hours trying to fix a bug
  - Some students spend hours to move from 90% to 100% for the first problem and never start the next problem
  - Move on the next problem shortly after you reach 70%-90% of the score!

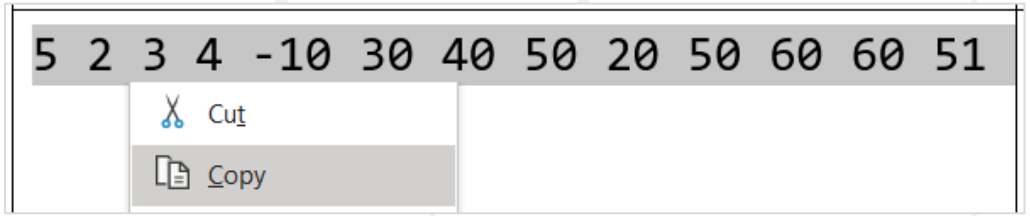
Typical Mistakes at the Exam (3)

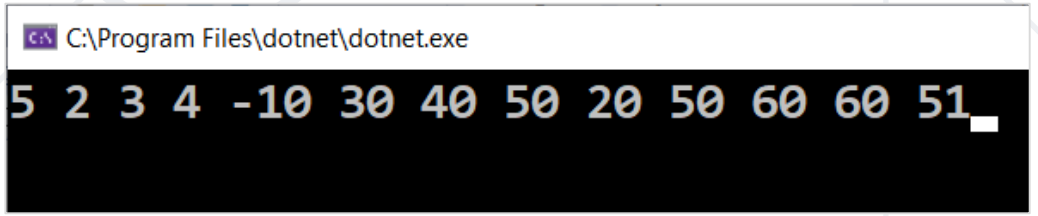
- **Wrong approach #6**: don't take a break, when you block
  - Everyone can block, get nervous, or become distracted
  - Take a short break, go outside, breathe, calm down
- **Wrong approach #7**: come to the exam unprepared
  - Prepare yourself, study hard, practice a lot, solve sample exams
  - You are ready, when you can solve any previous exam for 1-2 hours
- **Wrong approach #8**: trying to cheat
  - Many students try to cheat (e.g. use help from friends)
  - Cheating is bad for you! Who will do your future job?

Typical Mistakes at the Exam (4)

- **Wrong approach #9**: working without a mouse
  - Use the mouse, not a touchpad!
  - Mouse works more precisely
  - Mouse saves time and effort

- **Wrong approach #10**: typing the sample input examples by hand
  - Use copy/paste for the input examples!



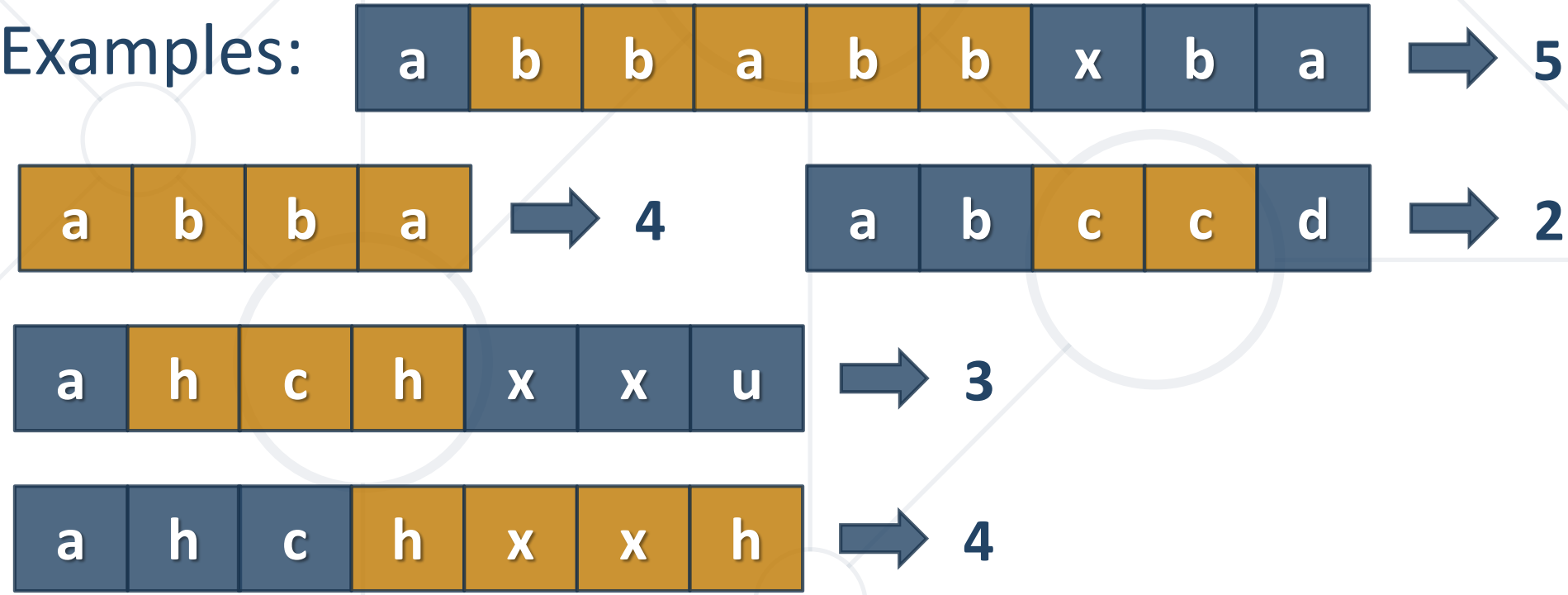


Sample Exam Problems

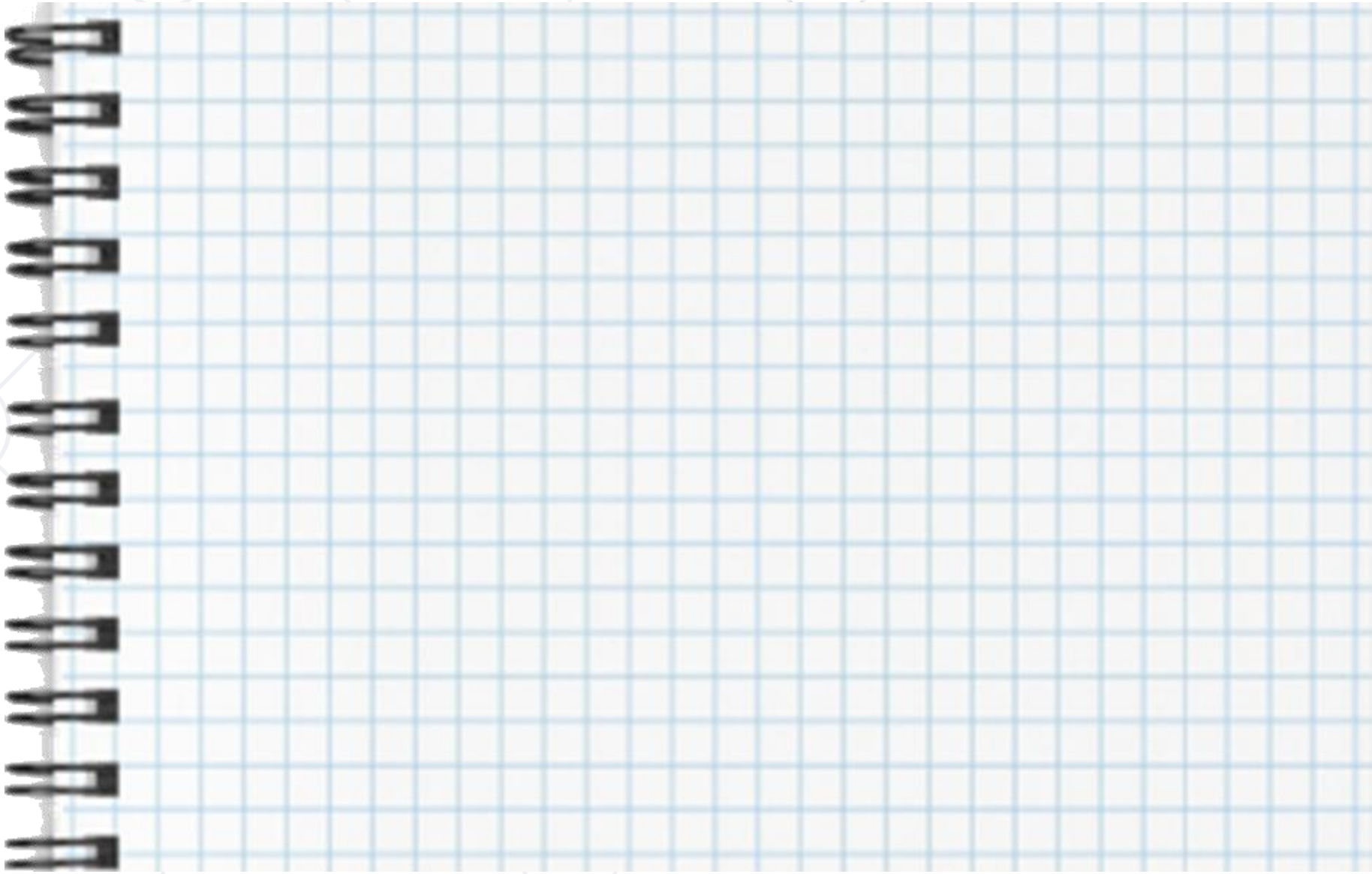
Following the Best Practices

Tech Problem: Longest Palindrome Sub-List

- We are given a **list of letters**
  - We want to find the longest sub-list, which is a **palindrome** (reads the same backward as forward)
- Examples:



Take a Pen and Paper and Visualize Ideas



Longest Palindrome Sub-List: Analysis

- Problem analysis: 2 types of palindromes
  - **Odd-length** (single letter at the center)

  - **Even-length** (two letters at the center)

Largest Palindrome Sub-List: Solutions

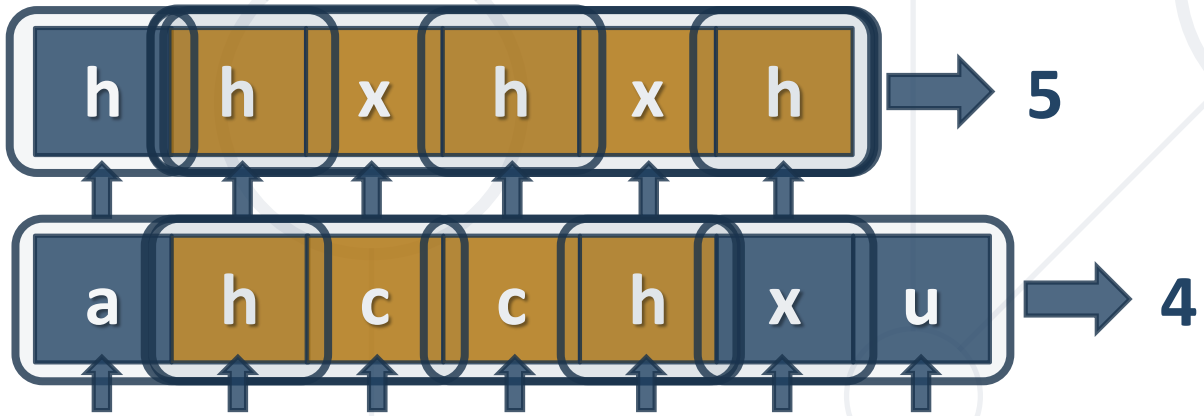
- **Potential solutions**:

  1. Find **all possible start + end positions** and check for palindrome
  2. Find **all possible single central points** and **double central points** and check for palindrome around them
  3. Find all sub-lists of size **n** (the length of the input list), then of size **n-1**, **n-2**, …, **1** and check for palindrome each sub-list

- Can we find the length of the longest palindrome without checking all palindromes in the list? → Yes, solution #3
- Which is the **most efficient solution**? → solution #2

Largest Palindrome Sub-List: Algorithm

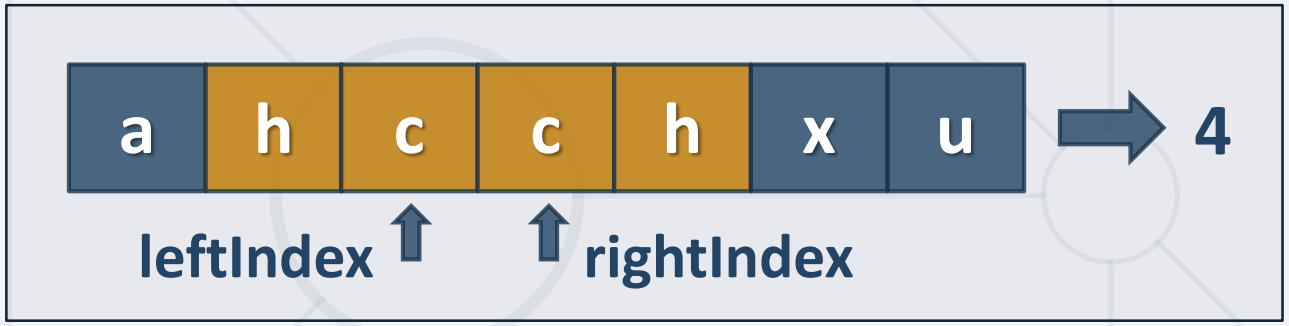
- **Algorithm** (sequence of steps) for solution #2:
  - Choose **each letter as central point** and count how many letters around it form a palindrome
  - Choose **each two consecutive equal letters as central point** and count how many letters around them form a palindrome
  - Choose **the longest** among all palindromes found



Largest Palindrome Sub-List: Implementation

Software University

```
int PalindromeLen(int leftIndex, int rightIndex)
{
  while (leftIndex > 0 && rightIndex < letters.Length
      && letters[leftIndex] == letters[rightIndex])
  {
    leftIndex--;
    rightIndex++;
  }
  return rightIndex - leftIndex - 1;
}
```

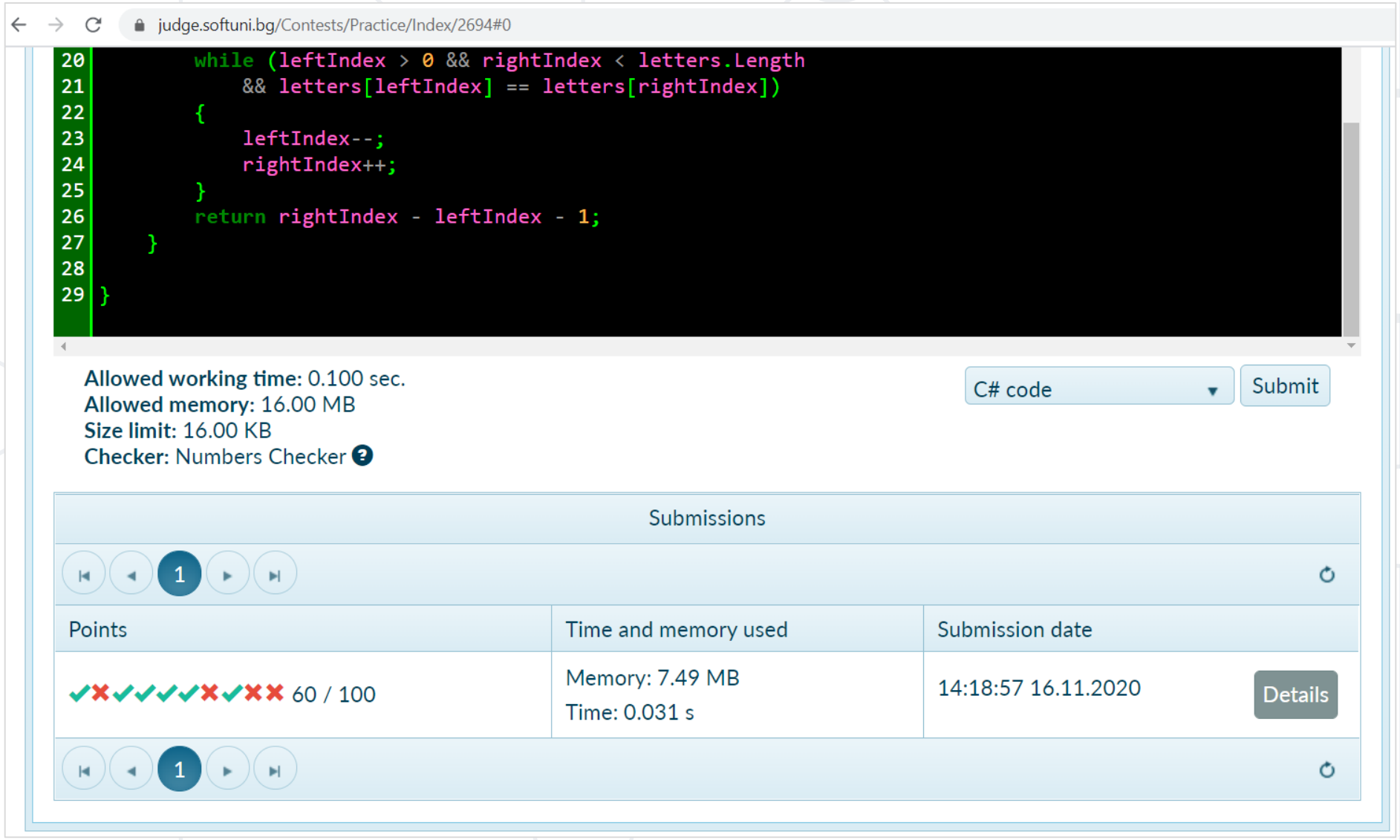


Largest Palindrome Sub-List: Implementation

Software University

```
string letters = Console.ReadLine();
int maxLen = 0;
// Check all single letter central points
for (var c = 1; c < letters.Length; c++)
  maxLen = Math.Max(maxLen, PalindromeLen(c, c));
// Check all double letter central points
for (var c = 1; c < letters.Length-1; c++)
  maxLen = Math.Max(maxLen, PalindromeLen(c, c+1));
Console.WriteLine(maxLen);
```

Submit to Judge



Use the Debugger

- Use the debugger, with good input

Program.cs

ConsoleAppTest

ConsoleAppTest

```
static int PalindromeLen(string letters, int leftIndex, int rightIndex)
{
    while (leftIndex > 0 && rightIndex < letters.Length
        && letters[leftIndex] == letters[rightIndex])
    {
        leftIndex--;
        rightIndex++;
    }
    return rightIndex - leftIndex - 1;
}

}
```

Bug Fix

```
int PalindromeLen(int leftIndex, int rightIndex)
{
  while (leftIndex >= 0 && rightIndex < letters.Length
      && letters[leftIndex] == letters[rightIndex])
  {
    leftIndex--;
    rightIndex++;
  }
  return rightIndex - leftIndex - 1;
}
```

90/100 → Go Ahead or Debug More?

Software University

judge.softuni.bg/Contests/Practice/Index/2694#0

```
20          while (leftIndex >= 0 && rightIndex < letters.Length
21              && letters[leftIndex] == letters[rightIndex])
22          {
23              leftIndex--;
24              rightIndex++;
25          }
26          return rightIndex - leftIndex - 1;
27      }
28
29  }
```

Allowed working time: 0.100 sec.
Allowed memory: 16.00 MB
Size limit: 16.00 KB
Checker: Numbers Checker

C# code | Submit

Submissions

1

Points	Time and memory used	Submission date	
✓✓✓✓✓✓✓✓✗✓ 90 / 100	Memory: 7.49 MB Time: 0.031 s	14:21:29 16.11.2020	Details
✓✗✓✓✓✓✗✓✗✗ 60 / 100	Memory: 7.49 MB Time: 0.031 s	14:18:57 16.11.2020	Details

1

Another Bug Fix

```
string letters = Console.ReadLine();
int maxLen = 0;
// Check all single letter central points
for (var c = 0; c < letters.Length; c++)
  maxLen = Math.Max(maxLen, PalindromeLen(c, c));
// Check all double letter central points
for (var c = 0; c < letters.Length-1; c++)
  maxLen = Math.Max(maxLen, PalindromeLen(c, c+1));
Console.WriteLine(maxLen);
```

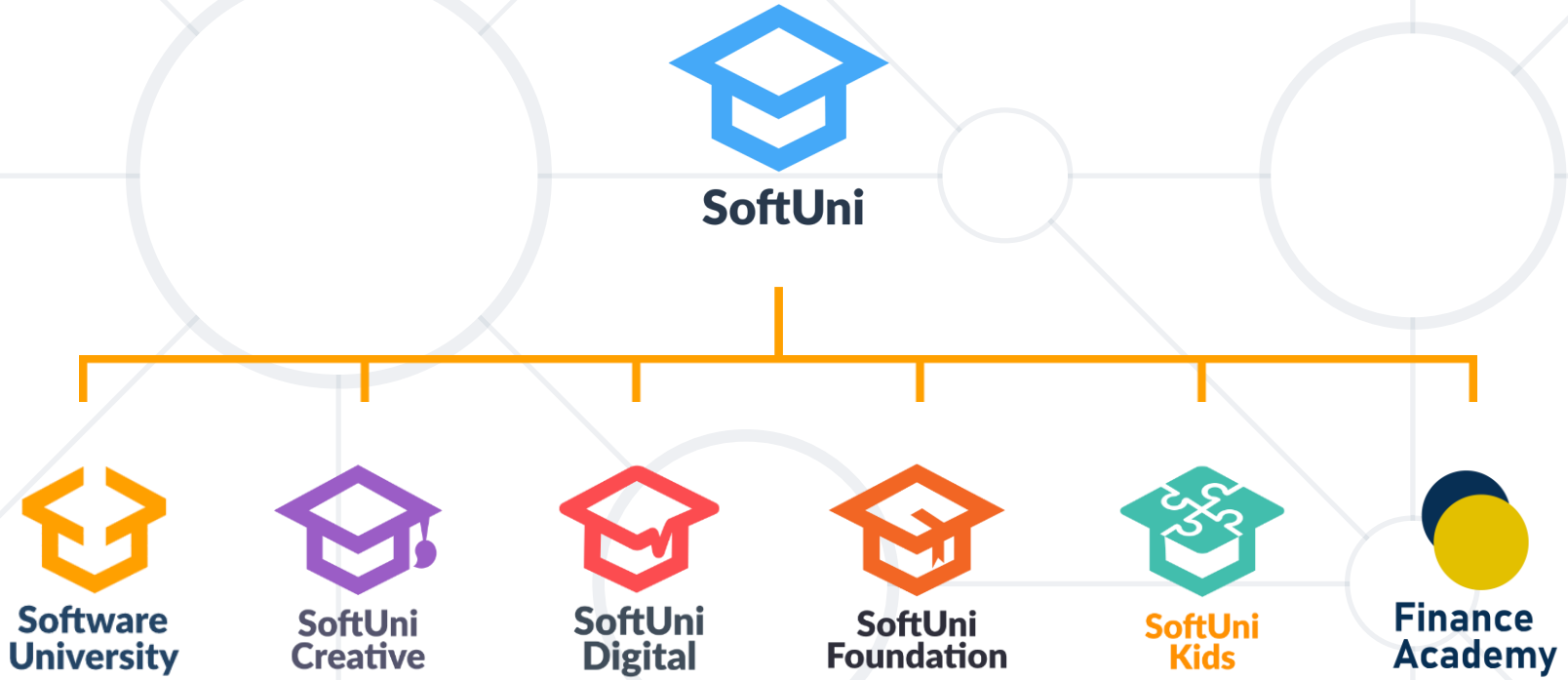
Largest Palindrome Sub-List: Review

- Review the results
  - Does this solution work well for all cases? Any edge cases?
- Tests, covering the edge cases:
  - abc, abcd, ab, a → 1
  - aa, aa0, 0aa, 0aa1, xxaazz, 01aa234 → 2
  - aaa, aaa0, 0aaa, 0aaa1, 012aaa34 → 3
  - aaaa, abba, 0abba, xxxx0, 0abba1 → 4
- Can we solve this problem better?

Summary

- The stages of problem solving:
  - Define the problem (requirements)
  - Analyze the problem (deep understand)
  - Identify potential solutions (ideas)
  - Evaluate and choose the best solution
  - Algorithm design (action plan)
  - Implement and review results

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

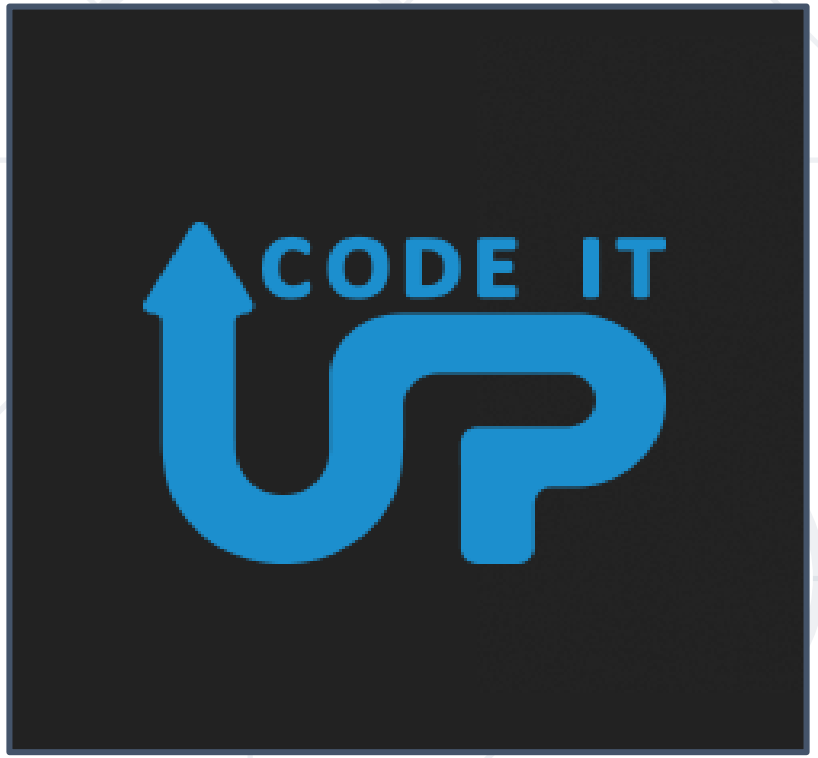
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

FORUM