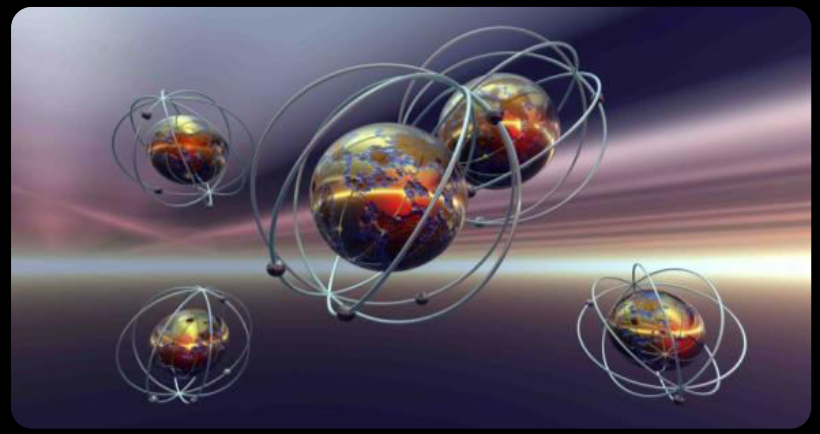
Methodology of Problem Solving

Efficiently Solving Computer Programming Problems

SoftUni Foundation



SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

Have a Question?

SoftUni Foundation

sli.do

#DsAlgo

SoftUni Foundation

Problems Solving

From Chaotic to Methodological Approach

Read and Analyze the Problems

SoftUni Foundation

- Consider you are at a computer programming exam or contest
  - You have 5 problems to solve in 6 hours
- First read carefully all problems and try to estimate how complex each of them is
  - Read the requirements, don't invent them!
- Start solving the easiest / fastest to solve problem first!
  - Leave the most complex / slow to solve problem last!
  - Approach the next problem when the previous is well tested

Analyzing the Problems

SoftUni Foundation

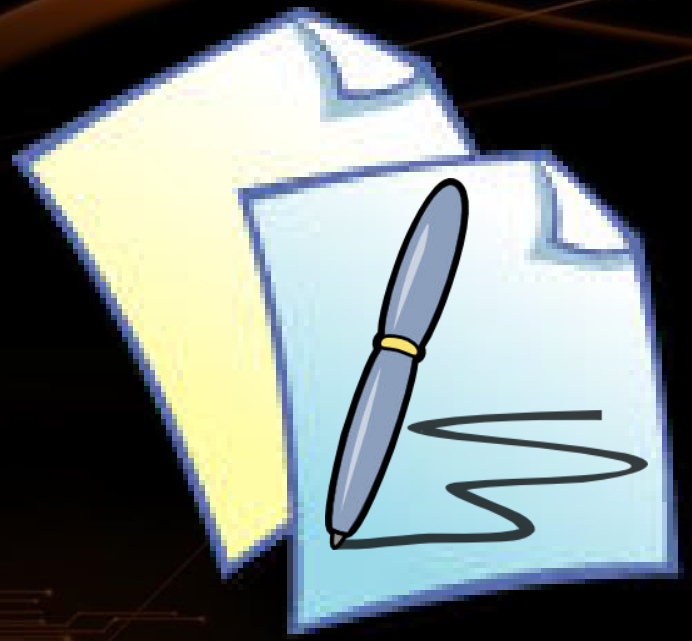
- Example: we are given 3 problems:
  - Cards Shuffle
    - Shuffle a deck of 52 cards in random order
  - Zig-Zag Matrix
    - Find the max-sum zig-zag path in a matrix
    - Start from the first column, go up, then down, then again up, them again down, …
  - Blocks
    - Generate all blocks 2 x 2 holding 4 of the first n Latin letters

Analyzing the Problems (2)

SoftUni Foundation

- Read carefully the problem descriptions
  - Think a bit about their possible solutions
- Order the problems from the easiest to the most complex:
  - Cards Shuffle
    - Trivial – randomize the elements of array
  - Blocks
    - Generate variations, rotate and check for duplicates
  - Zig-Zag Matrix
    - Needs summing, sorting and text file processing

SoftUni Foundation



Using a Paper and a Pen

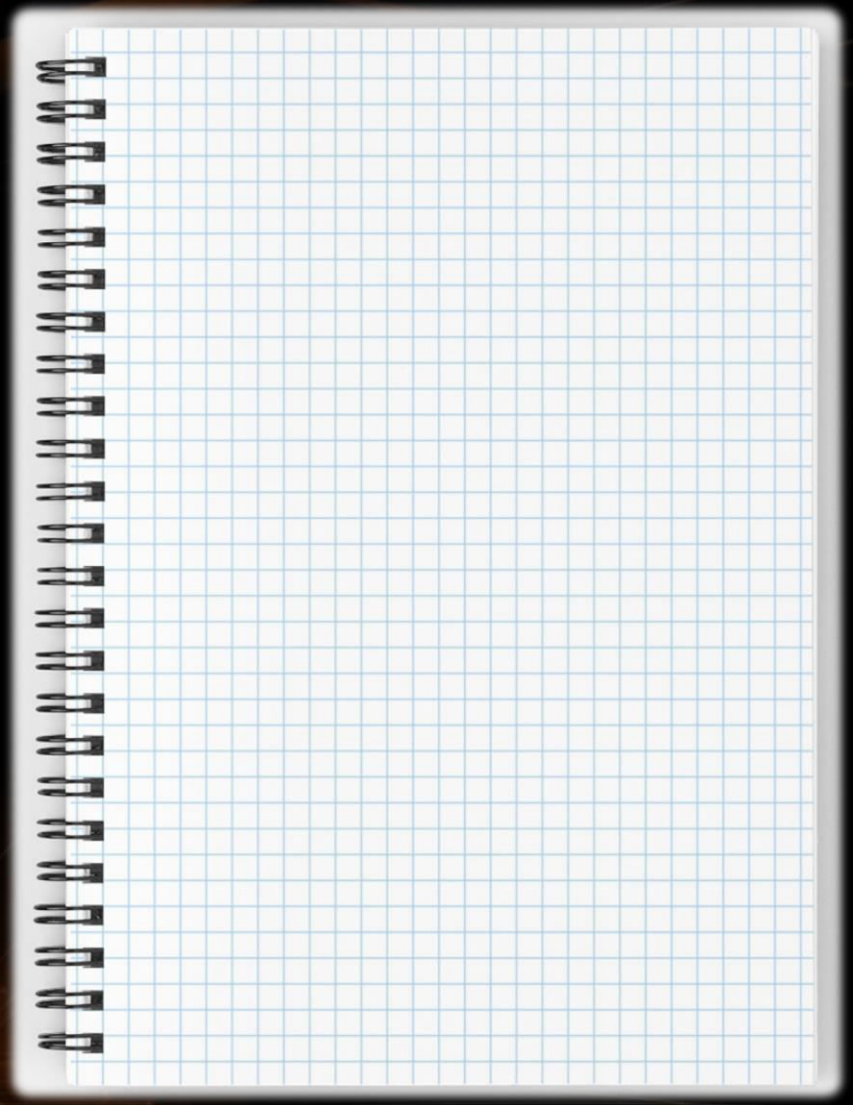
Visualizing and Sketching your Ideas

Use a Sheet of Paper and a Pen

SoftUni Foundation

- 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 efficiently
  - Paper works faster than keyboard / screen
  - Other visualization tools could also work well

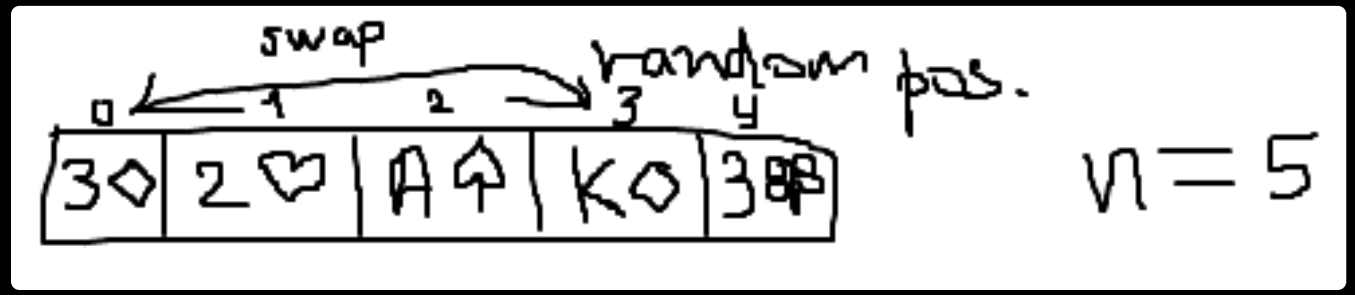
Prefer Squared Paper



- Squared paper works best for algorithmic problems
  - Easy to draw a table
  - Easy to draw a coordinate system with objects at it
  - Easy to calculate distances
  - Easy to sketch a problem and solution idea

Paper and Pen

- Consider the "Cards Shuffle" problem
  - We can sketch it to start thinking



  - Some ideas immediately come, e.g.
    - Split the deck into two parts and swap them a multiple times
    - Swap 2 random cards a random number of times
    - Swap each card with a random card

SoftUni Foundation

Invent Ideas

Think-up, Invent Ideas and Check Them

Think Up, Invent and Try Ideas

SoftUni Foundation

- First take an example of the problem
  - Sketch it on the sheet of paper
- Next try to invent some idea that works for your example
- Check if your idea will work for other examples
  - Try to find a case that breaks your idea
  - Try challenging examples and unusual cases
- If you find your idea incorrect, try to fix it
  - Or just invent a new idea

Invent and Try Ideas – Example

SoftUni Foundation

- Consider the "cards shuffle" problem
- Idea #1: random number of times split the deck into left and right part and swap them
  - How to represent the cards?
  - How to chose a random split point?
  - How to perform the exchange?
- Idea #2: swap each card with a random card
  - How many times to repeat this?
  - Is this fast enough?

Invent and Try Ideas – Example (2)

SoftUni Foundation

- Idea #3: swap 2 random cards a random number of times
  - How to choose two random cards?
  - How many times to repeat this?
- Idea #4: choose a random card and insert it in front of the deck
  - How to choose random card?
  - How many times to repeat this?
- Idea #5: do you have another idea?

SoftUni Foundation

Divide and Conquer

Decompose Problems into Manageable Pieces

Decompose the Problem

SoftUni Foundation

- Work decomposition is natural in engineering
  - It happens every day in the industry
  - Projects are decomposed into subprojects
- Complex problems could be decomposed into several smaller sub-problems
  - Technique known as "Divide and Conquer"
  - Small problems could easily be solved
  - Smaller sub-problems could be further decomposed as well

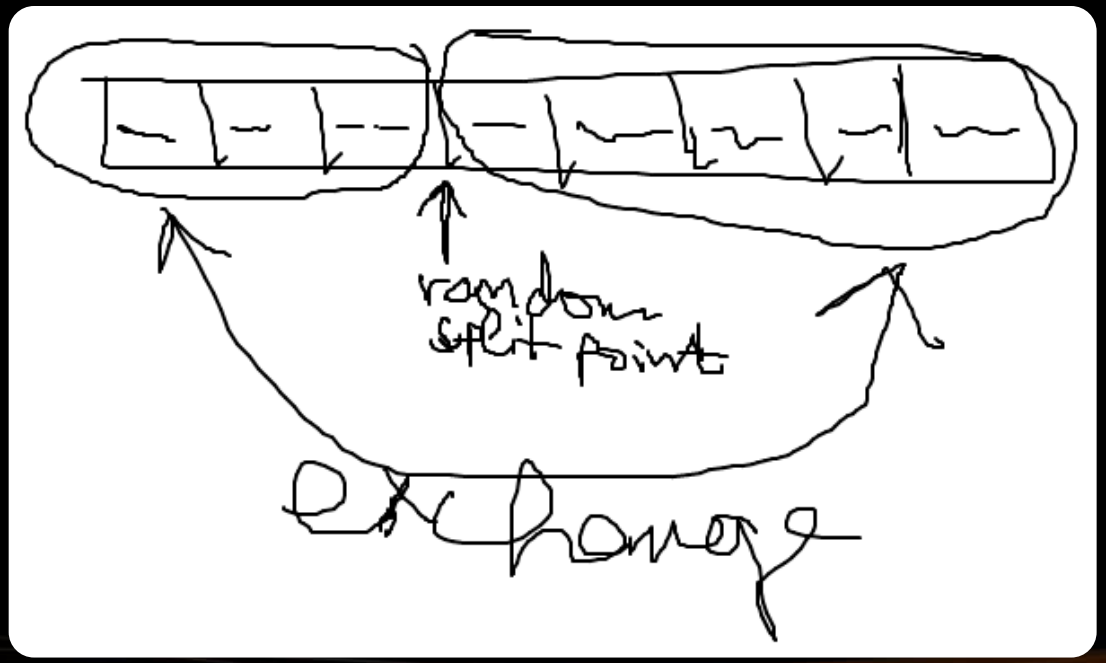
Divide and Conquer – Example

SoftUni Foundation

- Let's try idea #1:
  - Split the deck into left and right part and swap them (many times)
- Divide and conquer
  - Sub-problem #1 (single exchange) – split the deck into two random parts and exchange them
  - Sub-problem #2 – choosing a random split point
  - Sub-problem #3 – combining single exchanges
    - How many times to perform "single exchange"?

Sub-Problem #1 (Single Exchange)

- Split the deck into two parts at random split point
  - And exchange these 2 parts
- We visualize this by paper and pen:



Sub-Problem #1 (Single Exchange)

SoftUni Foundation

- Choosing a random split point
  - Need to understand the concept of pseudo-random numbers and how to use it
  - In Internet lots of examples are available, some of them incorrect
- The class `System.Random` can do the job
- Important detail is that the `Random` class should be instantiated only once
  - Not at each random number generation

Sub-Problem #1 (Single Exchange)

SoftUni Foundation

- Combining a sequence of single exchanges to solve the initial problem
  - How many times to perform single exchanges to reliably randomize the deck?
  - N times (where N is the number of the cards) seems enough
- We have an algorithm:
  - N times split at random position and exchange the left and right parts of the deck

SoftUni Foundation

Check-up Your Ideas

Don't go Ahead before Checking Your Ideas

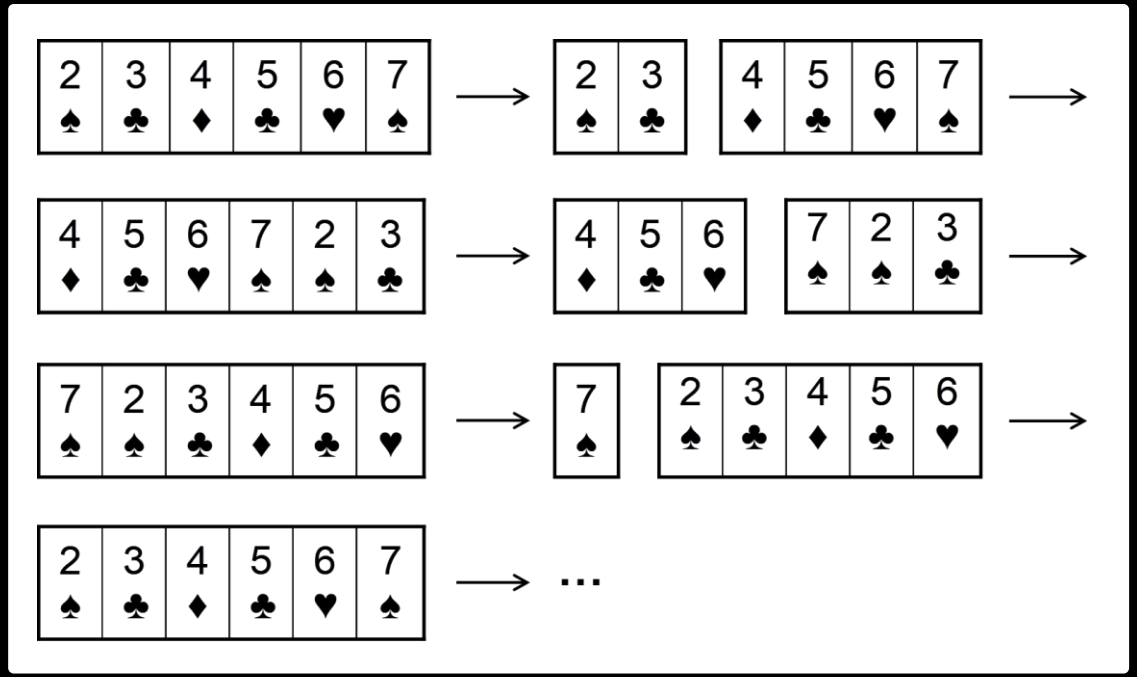
Check-up Your Ideas

SoftUni Foundation

- Check-up your ideas with examples
  - It is better to find a problem before the idea is implemented
  - When the code is written, changing your ideas radically costs a lot of time and effort
- Carefully select examples for check-up
  - Examples should be simple enough to be checked by hand in a minute
  - Examples should be complex enough to cover the most general case, not just an isolated case

Check-up Your Ideas – Example

- Let's check the idea:



- After 3 random splits and swaps we obtain the start position
  - Seems like a **serious problem**!

Invent New Idea If Needed

SoftUni Foundation

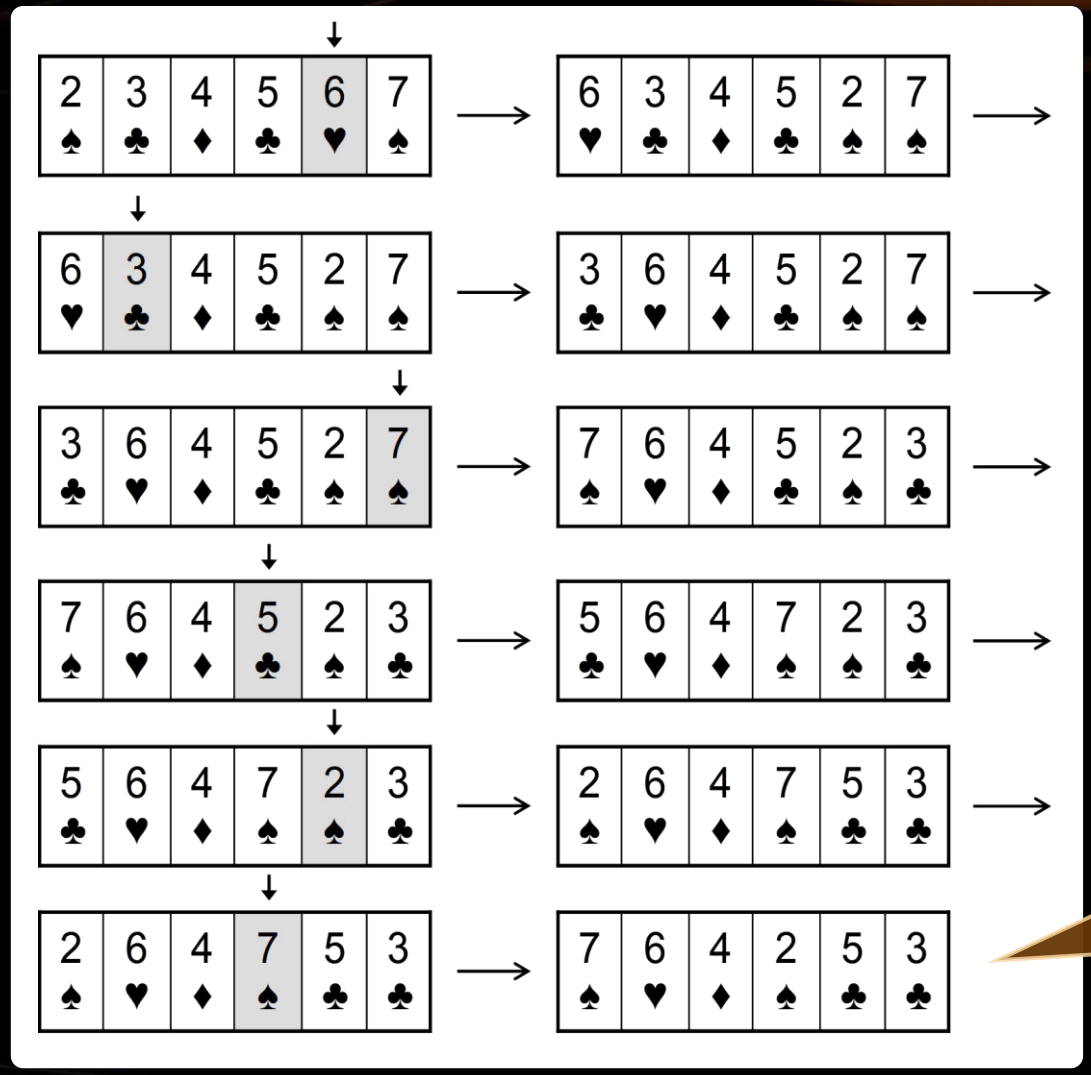
- What to do when you find your idea is not working in all cases?
  - Try to fix your idea
    - Sometimes a small change could fix the problem
  - Invent new idea and carefully check it
- Iterate
  - Usually your first idea is not the best
  - Invent ideas, check them, try various cases, find problems, fix them, invent better ideas, etc.

Invent New Ideas – Example

SoftUni Foundation

- Invent a few new ideas:
  - New idea #1 – multiple times select 2 random cards and exchange them
  - New idea #2 – multiple times select a random card and exchange it with the first card
  - New idea #3 – multiple times select a random card and move it to an external list
- Let's check the new idea #2
  - Is it correct?

Check-up the New Idea – Example



The result seems correct

SoftUni Foundation

Think about Data Structures

Select Data Structures that Will Work Well

Choosing Appropriate Data Structures

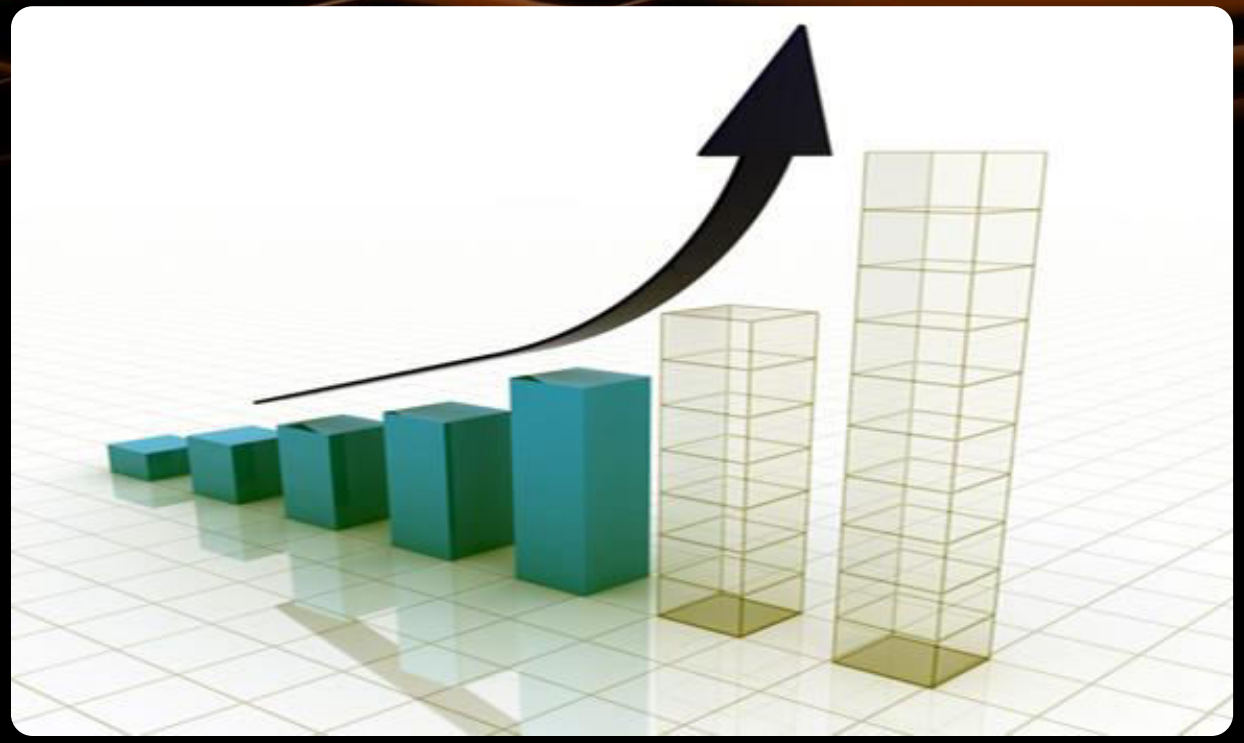
- Choose appropriate data structures before you start coding
  - Think how to represent input data
  - Think how to represent intermediate program states
  - Think how to represent the requested output
- You could find that your idea cannot be implemented efficiently
  - Or implementation will be very complex or inefficient

Choose Appropriate Data Structures – Example

SoftUni Foundation

- How to represent a single card?
  - The best idea is to create a structure / class **`Card`**
    - **`Face`** – could be string, int or enumeration
    - **`Suit`** – enumeration
- How to represent a deck of cards?
  - Array – **`Card[]`**
  - Indexed list – **`List<Card>`**
  - Set / Dictionary / Queue / Stack – not a fit

SoftUni Foundation



Efficiency and Performance

Is Your Algorithm Fast Enough?

Think About the Efficiency

SoftUni Foundation

- Think about efficiency before writing the first line of code
  - Estimate the running time (asymptotic complexity)
  - Check the requirements
    - Will your algorithm be fast enough to conform with them?
- You don't want to implement your algorithm and find that it is slow when testing
  - You will lose time

Efficiency is not Always Required

SoftUni Foundation

- Best solution is sometimes just not needed
- Read carefully your problem statement
- Sometimes ugly solutions could work for your requirements and it will cost you less time
- Example: if you need to sort n numbers, any algorithm will work well for $n \in [0..500]$
- Implement complex algorithms only when the problem really needs them

Efficiency – Example

SoftUni Foundation

- How many cards do we have?
  - In a typical deck we have 52 cards
    - No matter how fast the algorithm is – it will work fast enough
  - If we have N cards, we perform N swaps → the expected running time is O(N)
  - O(N) will work fast for millions of cards
- Conclusion:
  - The efficiency is not an issue in this problem

SoftUni Foundation

Implementation

Coding and Testing Step-by-Step

Start Coding: Check List

SoftUni Foundation

- Never start coding before you find a correct idea that will meet the requirements
  - What shall you write before you have a correct idea?
- Checklist to follow before start of coding:
  - Ensure you understand the requirements well
  - Ensure you have come up a good idea
  - Ensure your idea is correct
  - Ensure you know what data structures to use
  - Ensure the performance will be sufficient

Coding Check List – Example

SoftUni Foundation

- Checklist before you start coding:
  - Ensure you understand the requirements well
    - Yes, shuffle a given deck of cards
  - Ensure you have come up with a correct idea
    - Yes, the idea seems correct and is tested
  - Ensure you know what data structures to use
    - Class `Card`, enumeration `Suit` and `List<Card>`
  - Ensure the performance will be sufficient
    - Linear running time → good performance

Implement your Algorithm Step-by-Step

- "Step-by-step" approach is always better than "build all, then test"
  - Implement a piece of your program and test it
  - Then implement another piece of the program and test it
  - Finally put together all pieces and test it
- Small increments (steps) reveal errors early
  - "Big bang" integration takes more time

Step #1 – Class Card

SoftUni Foundation

```
class Card
{
  public string Face { get; set; }
  public Suit Suit { get; set; }

  public override string ToString()
  {
    string card = "(" + this.Face + " " + this.Suit +")";
    return card;
  }
}

enum Suit
{
  Club, Diamond, Heart, Spade
}
```

Step #1 – Test

SoftUni Foundation

- Testing the class **Card** to get feedback as early as possible:

```
static void Main()
{
  Card card = new Card() { Face="A", Suit=Suit.Club };
  Console.WriteLine(card);
}
```

- The result is as expected:

```
(A Club)
```

Step #2 – Create and Print a Deck of Cards

SoftUni Foundation

```
static void Main()
{
  var cards = new List<Card>();
  cards.Add(new Card() { Face = "7", Suit = Suit.Heart });
  cards.Add(new Card() { Face = "A", Suit = Suit.Spade });
  cards.Add(new Card() { Face = "10", Suit = Suit.Diamond });
  cards.Add(new Card() { Face = "2", Suit = Suit.Club });
  cards.Add(new Card() { Face = "6", Suit = Suit.Diamond });
  cards.Add(new Card() { Face = "J", Suit = Suit.Club });
  PrintCards(cards);
}

static void PrintCards(List<Card> cards)
{
  foreach (Card card in cards) Console.Write(card);
  Console.WriteLine();
}
```

Step #2 – Test

- Testing the deck of cards seems to be working correctly:

```
(7 Heart)(A Spade)(10 Diamond)(2 Club)(6 Diamond)(J Club)
```

Step #3 – Single Exchange

SoftUni Foundation

```
static void PerformSingleExchange(List<Card> cards)
{
  Random rand = new Random();
  int randomIndex = rand.Next(1, cards.Count - 1);
  Card firstCard = cards[1];
  Card randomCard = cards[randomIndex];
  cards[1] = randomCard;
  cards[randomIndex] = firstCard;
}
```

Step #3 – Test

SoftUni Foundation

- To test the single exchange we use the following code:

```
static void Main()
{
  List<Card> cards = new List<Card>();
  cards.Add(new Card() { Face = "2", Suit = Suit.Club });
  cards.Add(new Card() { Face = "3", Suit = Suit.Heart });
  cards.Add(new Card() { Face = "4", Suit = Suit.Spade });
  PerformSingleExchange(cards);
  PrintCards(cards);
}
```

- The result is unexpected:

```
(2 Club)(3 Heart)(4 Spade)
```

Step #3 – Fix Bug and Test

SoftUni Foundation

- The first element of the list is at index **0**, not **1**:

```
static void PerformSingleExchange(List<Card> cards)
{
  Random rand = new Random();
  int randomIndex = rand.Next(1, cards.Count - 1);
  Card firstCard = cards[0];
  Card randomCard = cards[randomIndex];
  cards[0] = randomCard;
  cards[randomIndex] = firstCard;
}
```

- The result is again incorrect (3 times the same):

```
(3 Heart)(2 Club)(4 Spade)
(3 Heart)(2 Club)(4 Spade)
(3 Heart)(2 Club)(4 Spade)
```

Step #3 – Fix Again and Test

SoftUni Foundation

- `Random.Next()` has exclusive end range:

```
static void PerformSingleExchange(List<Card> cards)
{
  Random rand = new Random();
  int randomIndex = rand.Next(1, cards.Count);
  Card firstCard = cards[0];
  Card randomCard = cards[randomIndex];
  cards[0] = randomCard;
  cards[randomIndex] = firstCard;
}
```

- The result now seems correct:

```
(3 Heart)(2 Club)(4 Spade)
(4 Spade)(3 Heart)(2 Club)
(4 Spade)(3 Heart)(2 Club)
```

Step #4 – Shuffle the Deck

SoftUni Foundation

- Shuffle the entire deck of cards:

```
static void ShuffleCards(List<Card> cards)
{
  for (int i = 1; i <= cards.Count; i++)
  {
    PerformSingleExchange(cards);
  }
}
```

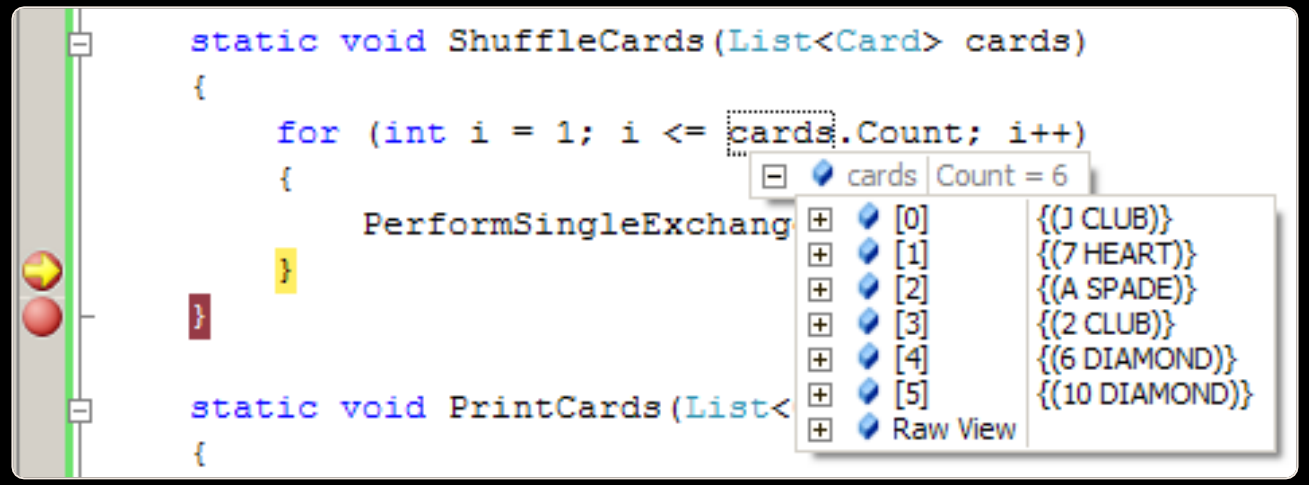
- The result is surprisingly incorrect:

```
Initial deck: (7 Heart)(A Spade)(10 Diamond)(2 Club)(6 Diamond)(J Club)
After shuffle: (7 Heart)(A Spade)(10 Diamond)(2 Club)(6 Diamond)(J Club)
```

Step #4 – Strange Bug

- When we step through the code with the debugger, the result seems correct:

```
Initial deck:  (7 Heart)(A Spade)(10 Diamond)(2 Club)(6 Diamond)(J Club)
After shuffle: (10 Diamond)(7 Heart)(A Spade)(J Club)(2 Club)(6 Diamond)
```



- Without the debugger the result is wrong!

Step #4 – Fix Again and Test

SoftUni Foundation

- **Random** should be instantiated only once:

```
private static Random rand = new Random();
static void PerformSingleExchange(List<Card> cards)
{
  int randomIndex = rand.Next(1, cards.Count);
  Card firstCard = cards[0];
  Card randomCard = cards[randomIndex];
  cards[0] = randomCard;
  cards[randomIndex] = firstCard;
}
```

- The result finally is correct with and without the debugger

SoftUni Foundation

Testing the Code

Thoroughly Test Your Solution

Thoroughly Test your Solution

SoftUni Foundation

- Wise software engineers say that:
  - Inventing a good idea and implementing it is half of the solution
  - Testing is the second half of the solution
- Always test thoroughly your solution
  - Invest in testing!
  - One 90-100% solved problem could be better than 2 or 3 partially solved
  - Testing an existing problem takes less time than solving another problem from scratch

How to Test?

SoftUni Foundation

- Testing could not certify absence of defects
  - It just reduces the defects' rates
  - Well tested solutions are more likely to be correct
- Start testing with a good representative of the general case
  - Not a small isolated case
    - A typical case
  - Large enough test case, but
    - Small enough to be easily checkable

How to Test? (2)

SoftUni Foundation

- Test the border cases
  - E.g. if **n** ∈ [0..500] → try **n**=0, **n**=1, **n**=2, **n**=499, **n**=500
- If a bug is found
  - Repeat all tests after fixing it to avoid regressions
- Run a performance test
  - How to be sure that your algorithm is fast enough to meet the requirements?
  - Use copy-pasting to generate large test data

Read the Problem Statement

SoftUni Foundation

- Read carefully the problem statement
  - Does your solution print exactly what is expected?
  - Does your output follow the requested format?
  - Did you remove your debug printouts?
- Be sure to solve the requested problem, not the problem you think is requested!
  - Example: "Write a program to print the number of permutations of n elements" means to print a single number, not a set of permutations!

Testing – Example

SoftUni Foundation

- Test with full deck of 52 cards
  - Serious error found → change the algorithm
- Change the algorithm
  - Exchange the first card with a random card → exchange cards 0, 1, …, N-1 with a random card
  - Test whether the new algorithm works
- Test with 1 card, 2 cards, 0 cards, 52 cards
- Performance test with 52 000 cards

Test with 52 Cards – Example

SoftUni Foundation

```
static void TestShuffle52Cards()
{
    List<Card> cards = new List<Card>();
    string[] allFaces = new string[] { "2", "3", "4", "5",
        "6", "7", "8", "9", "10", "J", "Q", "K", "A" };
    Suit[] allSuits = new Suit[] {
        Suit.Club, Suit.Diamond, Suit.Heart, Suit.Spade };
    foreach (var face in allFaces)
        foreach (var suit in allSuits)
        {
            Card card = new Card() { Face = face, Suit = suit };
            cards.Add(card);
        }
    ShuffleCards(cards);
    PrintCards(cards);
}
```

Test with 52 Cards – Example (2)

SoftUni Foundation

- The result is surprising:

```
(4 Diamond)(2 Diamond)(6 Heart)(2 Spade)(A Spade)(7 Spade)(3
Diamond)(3 Spade)(4 Spade)(4 Heart)(6 Club)(K Heart)(5 Club)(5
Diamond)(5 Heart)(A Heart)(9 Club)(10 Club)(A Club)(6 Spade)(7
Club)(7 Diamond)(3 Club)(9 Heart)(8 Club)(3 Heart)(9 Spade)(4
Club)(8 Heart)(9 Diamond)(5 Spade)(8 Diamond)(J Heart)(10
Diamond)(10 Heart)(10 Spade)(Q Heart)(2 Club)(J Club)(J Spade)(Q
Club)(7 Heart)(2 Heart)(Q Spade)(K Club)(J Diamond)(6 Diamond)(K
Spade)(8 Spade)(A Diamond)(Q Diamond)(K Diamond)
```

- Half of the cards keep their initial positions!
  - Serious problem: the randomization algorithm is not reliable

Fixing the Algorithm

SoftUni Foundation

- New idea that slightly changes the algorithm:
  - Exchange the first card with a random card → exchange cards 0, 1, …, N-1 with a random card

```
static void PerformSingleExchange(List<Card> cards, int index)
{
  int randomIndex = rand.Next(1, cards.Count);
  Card firstCard = cards[index];
  cards[index] = cards[randomIndex];
  cards[randomIndex] = firstCard;
}

static void ShuffleCards(List<Card> cards)
{
  for (int i = 0; i < cards.Count; i++)
    PerformSingleExchange(cards, i);
}
```

Test with 52 Cards (Again)

SoftUni Foundation

- The result now seems correct:

```
(9 Heart)(5 Club)(3 Club)(7 Spade)(6 Club)(5 Spade)(6 Heart)
(4 Club)(10 Club)(3 Spade)(K Diamond)(10 Heart)(8 Club)(A
Club)(J Diamond)(K Spade)(9 Spade)(7 Club)(10 Diamond)(9
Diamond)(8 Heart)(6 Diamond)(8 Spade)(5 Diamond)(4 Heart)
(10 Spade)(J Club)(Q Spade)(9 Club)(J Heart)(K Club)(2 Heart)
(7 Heart)(A Heart)(3 Diamond)(K Heart)(A Spade)(8 Diamond)(4
Spade)(3 Heart)(5 Heart)(Q Heart)(4 Diamond)(2 Spade)(A
Diamond)(2 Diamond)(J Spade)(7 Diamond)(Q Diamond)(2 Club)
(6 Spade)(Q Club)
```

- Cards are completely randomized

Test with 1 Card

SoftUni Foundation

- Create a method to test with 1 card:

```
static void TestShuffleOneCard()
{
  List<Card> cards = new List<Card>();
  cards.Add(new Card() { Face = "A", Suit = Suit.Club });
  CardsShuffle.ShuffleCards(cards);
  CardsShuffle.PrintCards(cards);
}
```

- We found a bug:

```
Unhandled Exception: System.ArgumentOutOfRangeException:
Index was out of range. Must be non-negative and less than
the size of the collection. Parameter name: index …
```

Test with 1 Card – Bug Fixing

SoftUni Foundation

- Isolate 1 card as a special case:

```
static void ShuffleCards(List<Card> cards)
{
  if (cards.Count > 1)
  {
    for (int i = 0; i < cards.Count; i++)
    {
      PerformSingleExchange(cards, i);
    }
  }
}
```

- Test shows that the problem is fixed

Test with 2 Cards

SoftUni Foundation

- Create a method to test with 2 cards:

```
static void TestShuffleTwoCards()
{
  List<Card> cards = new List<Card>();
  cards.Add(new Card() { Face = "A", Suit = Suit.Club });
  cards.Add(new Card() { Face = "3", Suit = Suit.Diamond });
  CardsShuffle.ShuffleCards(cards);
  CardsShuffle.PrintCards(cards);
}
```

- Bug: sequential executions get the same result:

```
(3 Diamond)(A Club)
```

- The problem: the first and the second cards always exchange each other exactly once

Test with 2 Cards – Bug Fixing

SoftUni Foundation

- We allow each card to be exchanged with any other random card, including itself

```
static void PerformSingleExchange(List<Card> cards, int index)
{
  int randomIndex = rand.Next(0, cards.Count);
  Card firstCard = cards[index];
  Card randomCard = cards[randomIndex];
  cards[index] = randomCard;
  cards[randomIndex] = firstCard;
}
```

- Test shows that the problem is fixed

Test with 0 Cards; Regression Tests

SoftUni Foundation

- Testing with 0 cards (empty list) generates an empty list → correct result
  - Seems like the cards shuffle algorithm works correctly after the last few fixes
- Needs a regression test
  - Assure the new changes do not break all previously working cases
  - Test with full deck of 52 cards; with 1 card; with 2 cards; with 0 cards → everything works

Performance Test – 52 000 Cards

SoftUni Foundation

- Finally we need a performance test with 52 000 cards:

```
static void TestShuffle52000Cards() {
  var cards = new List<Card>();
  string[] allFaces = new string[] {"2", "3", "4", "5",
    "6", "7", "8", "9", "10", "J", "Q", "K", "A"};
  Suit[] allSuits = new Suit[] {
    Suit.Club, Suit.Diamond, Suit.Heart, Suit.Spade};
  for (int i = 0; i < 1000; i++)
    foreach (string face in allFaces)
      foreach (Suit suit in allSuits)
        cards.Add(new Card() { Face = face, Suit = suit });
  ShuffleCards(cards);
  PrintCards(cards);
}
```

Summary

SoftUni Foundation

- Problem solving needs methodology:
  - Understanding and analyzing problems
  - Using a sheet of paper and a pen for sketching
  - Thinking up, inventing and trying ideas
  - Decomposing problems into sub-problems
  - Selecting appropriate data structures
  - Thinking about the efficiency and performance
  - Implementing step-by-step
  - Testing the nominal case, border cases and efficiency

Problem Solving Methodology

SoftUni Foundation

Questions?

https://softuni.bg/opencourses/algorithms

License

SoftUni Foundation

- This course (slides, examples, labs, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "Fundamentals of Computer Programming with C#" book by Svetlin Nakov & Co. under CC-BY-SA license
  - "Data Structures and Algorithms" course by Telerik Academy under CC-BY-NC-SA license

Trainings @ Software University (SoftUni)

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

Software University

SoftUni Foundation

FORUM