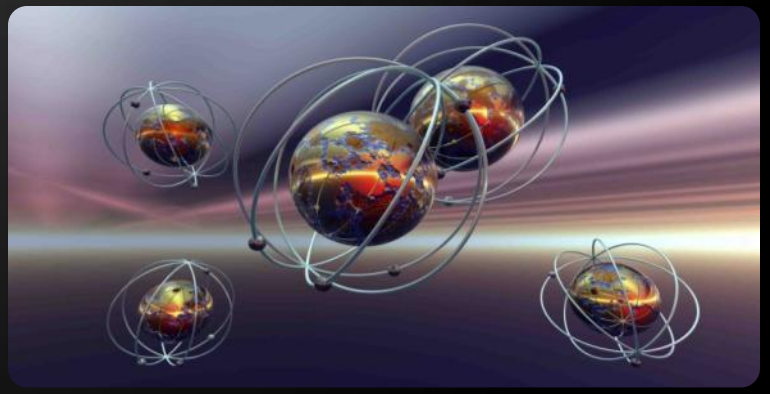
Telerik Academy

Methodology of Problem Solving

Efficiently Solving Computer Programming Problems



Data Structures and Algorithms

Telerik Software Academy

http://academy.telerik.com

Telerik Academy

Table of Contents

- Problem solving
  1. Read and Analyze the Problems
  2. Use a sheet of paper and a pen for sketching
  3. Think up, invent and try ideas
  4. Break the problem into subproblems
  5. Check up your ideas
  6. Choose appropriate data structures

Telerik Academy

Table of Contents (2)

7. Think about the efficiency
8. Implement your algorithm step-by-step
9. Thoroughly test your solution

- How to search in Google

Telerik Academy

Problems Solving

From Chaotic to Methodological Approach

Telerik Academy

Understanding the Requirements

Telerik Academy

Read and Analyze the Problems

- Consider you are at traditional computer programming exam or contest
  - You have 5 problems to solve in 8 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 most easy problem first!
- Leave the most complex problem last!
- Approach the next problem when the previous is completely solved and well tested

Analyzing the Problems

- Example: we are given 3 problems:
  1. Shuffle-cards
     - Shuffle a deck of cards in random order
  2. Students
     - Read a set of students and their marks and print top 10 students with the best results (by averaging their marks)
  3. Sorting numbers
     - Sort a set of numbers in increasing order

Analyzing the Problems (2)

- Read carefully the problems and think a bit about their possible solutions
- Order the problems from the easiest to the most complex:
  1. Sorting numbers
     - Trivial – we can use the built-in sorting in .NET
  2. Shuffle-cards
     - Need to randomize the elements of array
  3. Students
     - Needs summing, sorting and text file processing

Telerik Academy

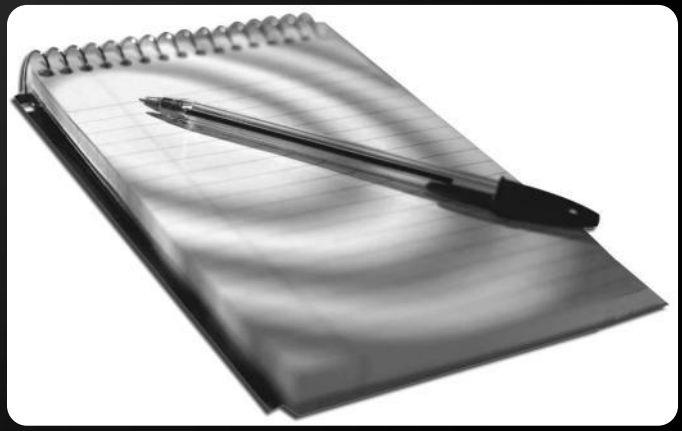
Using a Paper and a Pen

Visualizing and Sketching your Ideas

Telerik Academy

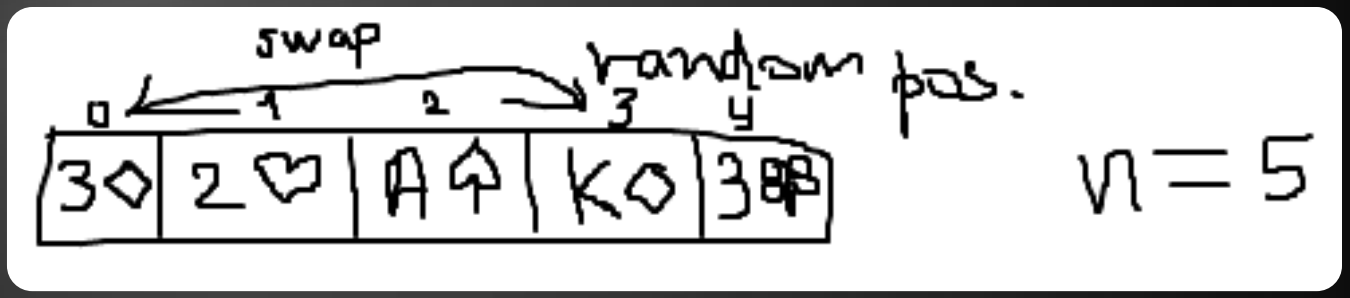
Use a Sheet of Paper and a Pen

- Never start solving a problem without a sheet of paper and 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 tool could also work well



Paper and Pen

- Consider the "cards shuffle" problem
  - We can sketch it to start thinking

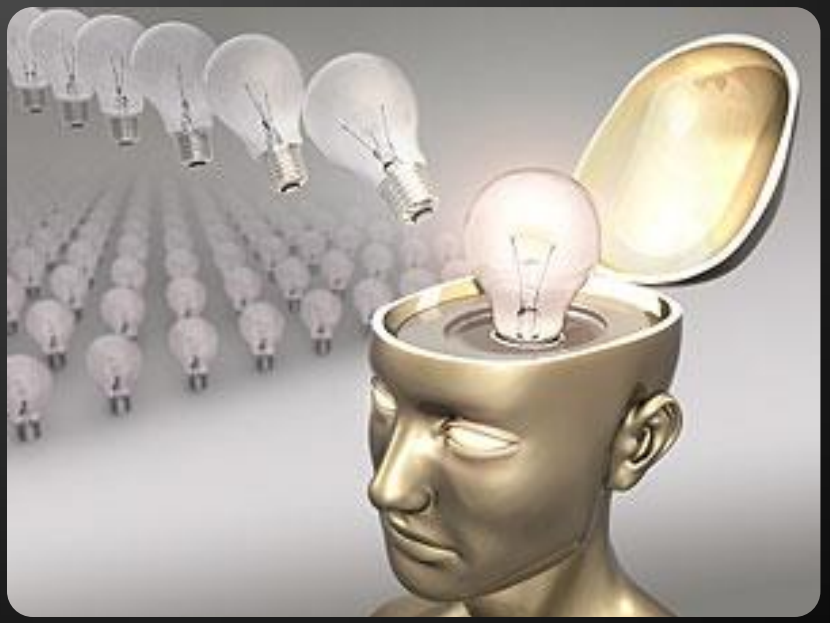


  - Some ideas immediately come, e.g.
    - Split the deck into two parts and swap them a multiple times
    - Swap 2 random cards a random number of times
    - Swap each card with a random card

Telerik Academy

Invent Ideas

Think-up, Invent Ideas and Check Them



Telerik Academy

Think up, Invent and Try Ideas

- 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

- 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)

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

Telerik Academy

Divide and Conquer

Decompose Problems into Manageable Pieces

Decompose the Problem into Subproblems

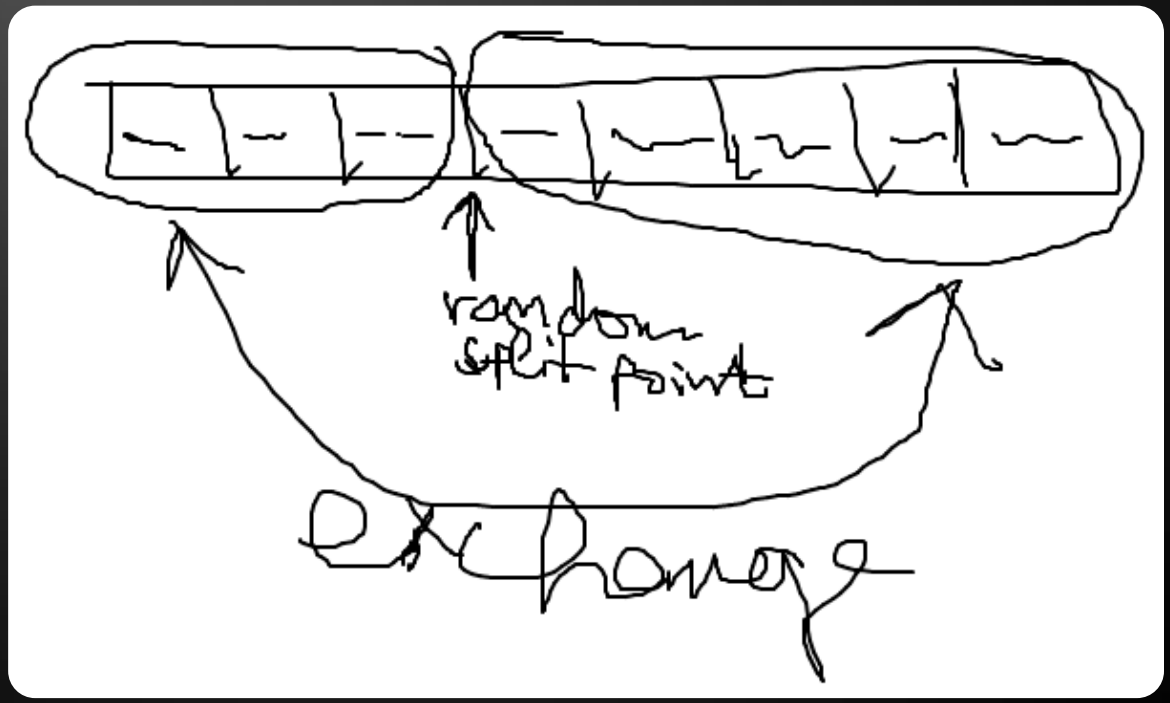
- Work decomposition is natural in engineering
  - It happens every day in the industry
  - Projects are decomposed into subprojects
- Complex problems could be decomposed into several smaller subproblems
  - Technique known as "Divide and Conquer"
  - Small problems could easily be solved
  - Smaller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

Divide and Conquer – Example

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

Subproblem #1 (Single Exchange)

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



Subproblem #2 (Random Split Point)

- Choosing a random split point
  - Needs 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

Telerik Academy

Subproblem #3 (Combining Single Exchanges)

- 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

Telerik Academy

Check-up Your Ideas

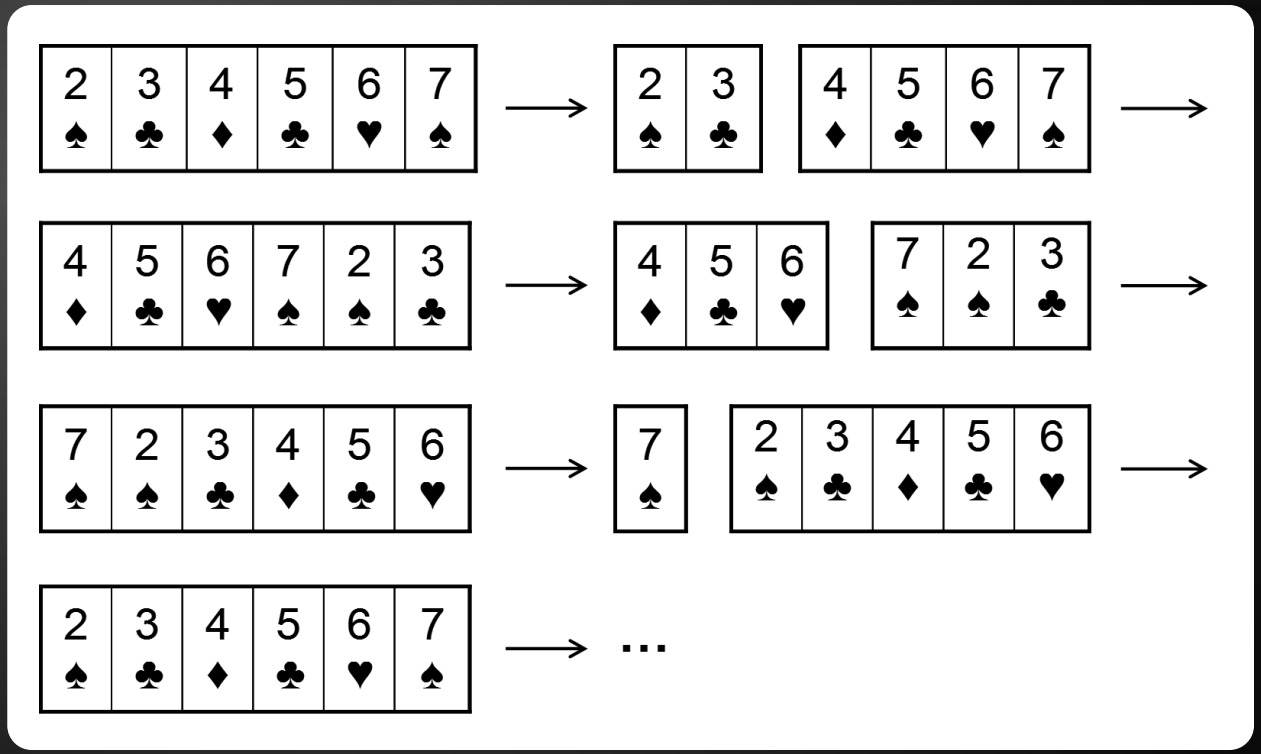
Don't go Ahead before Checking Your Ideas

Check-up Your Ideas

- Check-up your ideas with examples
  - It is better to find a problem before the idea is implemented
  - When the code is written, changing radically your ideas 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 bug!

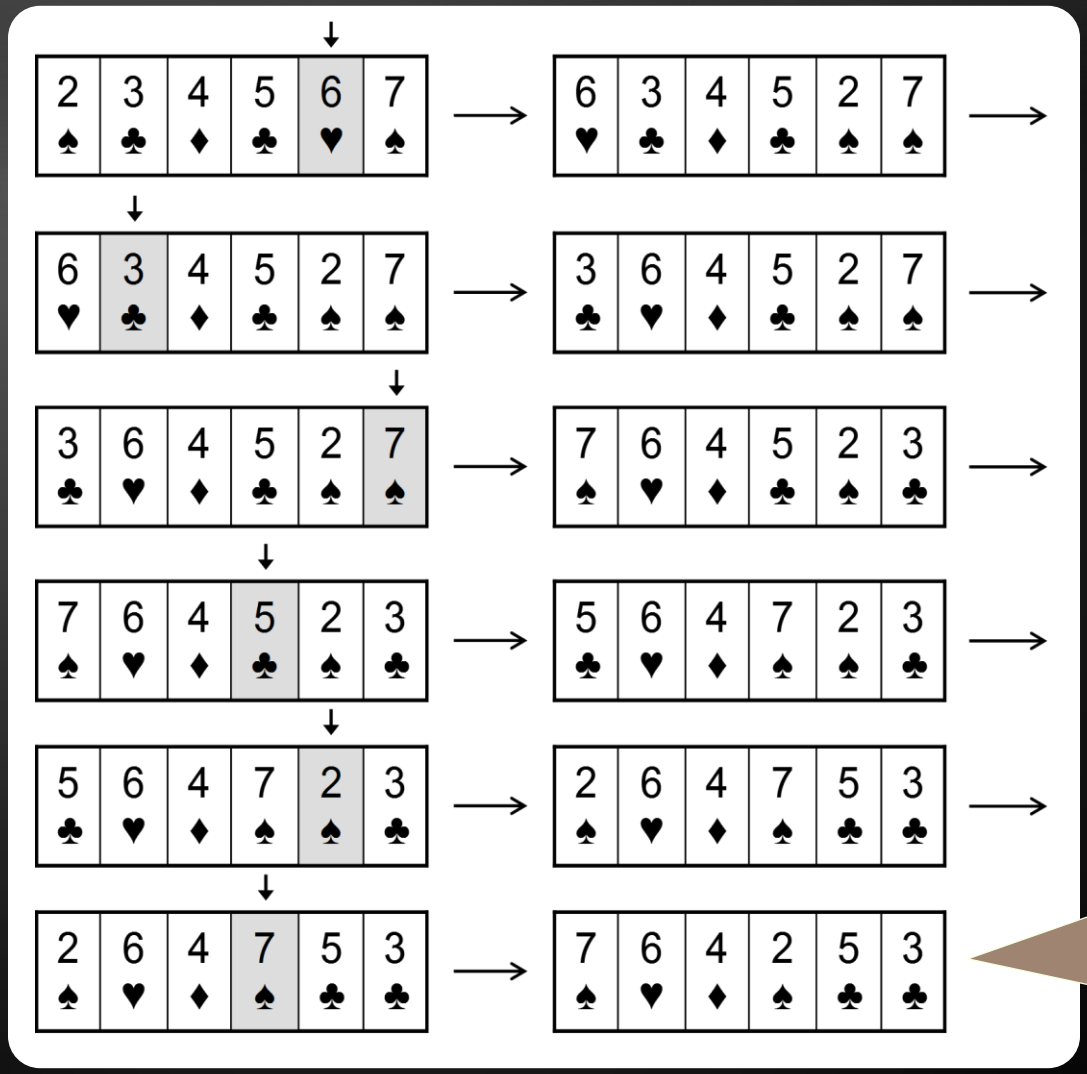
Invent New Idea If Needed

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

Invent New Ideas – Example

- Invent 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

Telerik Academy

Think about Data Structures

Select Data Structures that Will Work Well

Choosing Appropriate Data Structures

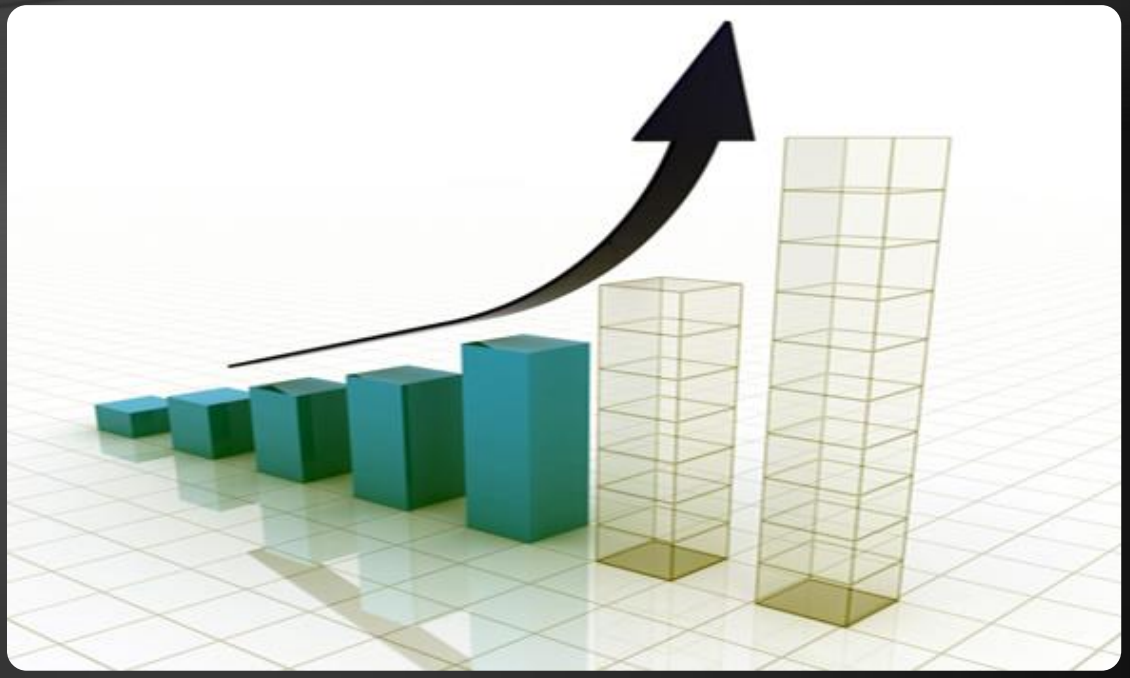
- Choose appropriate data structures before the start of coding
  - Think how to represent input data
  - Think how to represent intermediate program state
  - Think how to represent the requested output
- You could find that your idea cannot be implemented efficiently
  - Or implementation will be very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

Telerik Academy

Choose Appropriate Data Structures – Example

- How to represent a single card?
  - The best idea is to create a structure `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

Telerik Academy



Efficiency and Performance

Is Your Algorithm Fast Enough?

Think About the Efficiency

- 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 your time

Telerik Academy

Efficiency is not Always Required

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

Efficiency – Example

- How much cards 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

Telerik Academy

Implementation

Coding and Testing Step-by-Step

Start Coding: Check List

- Never start coding before you find correct idea that will meet the requirements
  - What you will write before you invent a correct idea to solve the problem?
- Checklist to follow before start of coding:
  - Ensure you understand well the requirements
  - Ensure you have invented 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

- Checklist before start of coding:
  - Ensure you understand well the requirements
    - Yes, shuffle given deck of cards
  - Ensure you have invented 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

```
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

- 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

```
static void Main()
{
  List<Card> 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

```
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

- 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

- The first element of 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

- 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

- Shuffle the entire deck of cards:

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

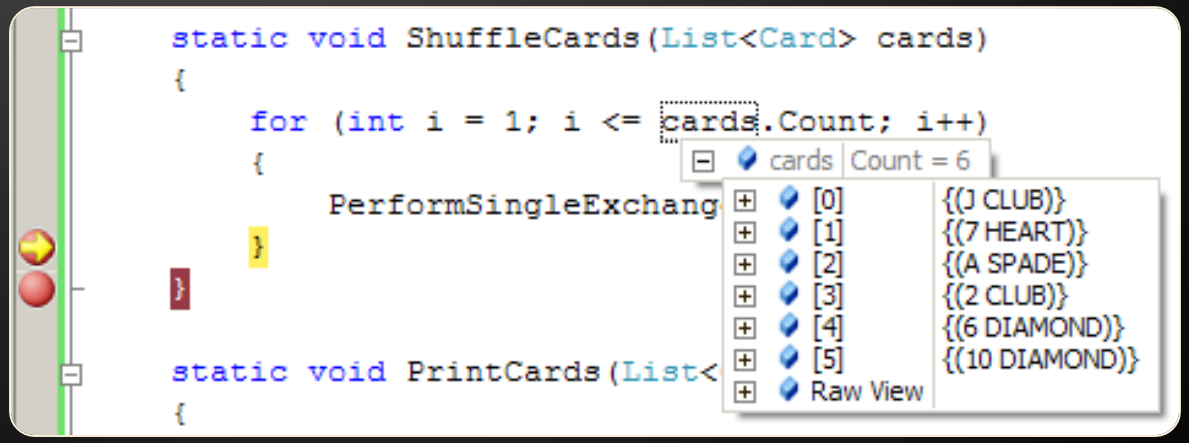
- The result is surprisingly incorrect:

```
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

- Random should be instantiated only once:

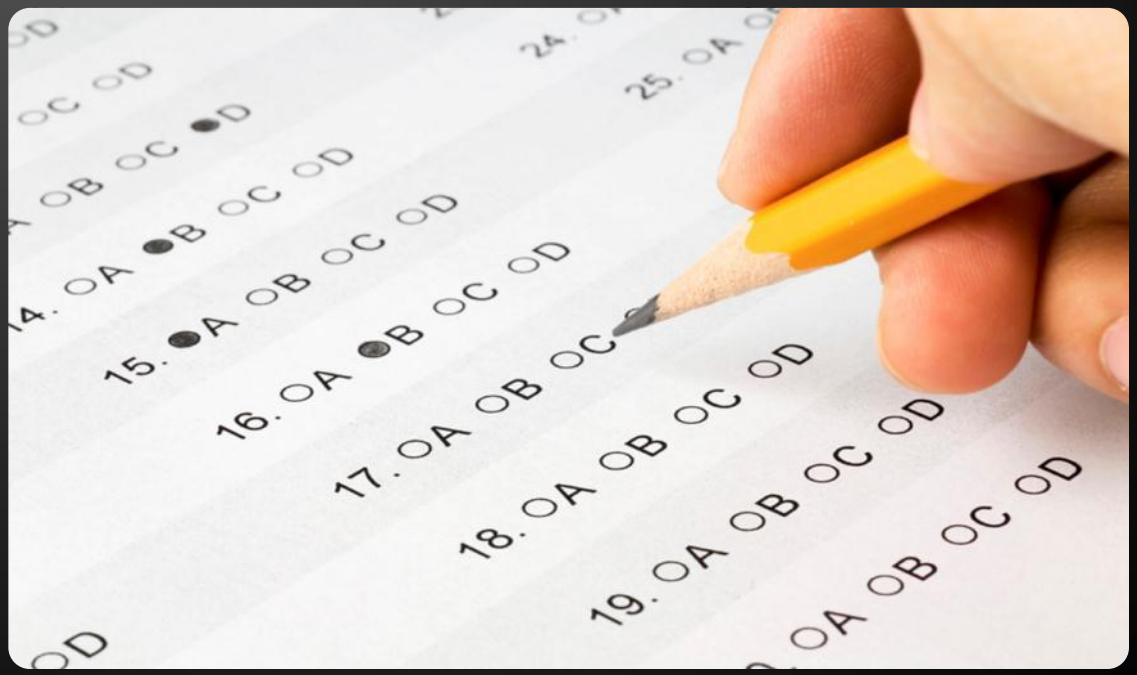
```
private static Random rand = new Random();
static void PerformSingleExchange(List<Card> cards)
{
  int randomIndex = rand.Next(1, cards.Count);
  Card firstCard = cards[0];
  Card randomCard = cards[randomIndex];
  cards[0] = randomCard;
  cards[randomIndex] = firstCard;
}
```

- The result finally is correct with and without the debugger

Telerik Academy

Testing

Thoroughly Test Your Solution



Telerik Academy

Thoroughly Test your Solution

- 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 is better than 2 or 3 partially solved
  - Testing existing problem takes less time than solving another problem from scratch

How to Test?

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

How to Test? (2)

- 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 load 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

- 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 on n elements" means to print a single number, not a set of permutations!

Testing – Example

- 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
- Test with 2 cards
- Test with 0 cards
- Load test with 52 000 cards

Test with 52 Cards – Example

```
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 (string face in allFaces)
   {
      foreach (Suit suit in allSuits)
      {
         Card card = new Card() { Face = face, Suit = suit };
         cards.Add(card);
      }
   }
   ShuffleCards(cards);
   PrintCards(cards);
}
```

Telerik Academy

Test with 52 Cards – Example (2)

- 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
  - We have serious problem – the randomization algorithm is not reliable

Fixing the Algorithm

- 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)

- 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

- 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

- We take 1 card are 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

- 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

- 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

Telerik Academy

Test with 0 Cards; Regression Tests

- 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
  - Test again that new changes did not break all previously working cases
- Test with full deck of 52 cards; with 1 card; with 2 cards with 0 cards → everything works

Load Test – 52 000 Cards

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

```
static void TestShuffle52000Cards()
{
  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};
  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);
}
```

Telerik Academy

How to Search in Google?

Some Advices for Successful Google Searching during Problem Solving

Search in Google Laws

- Keep it simple
  - Most queries do not require advanced operators or unusual syntax → simple is good
- Think what the page you are looking for is likely to contain → use the words that are most likely to appear on the page
  - A search engine is not a human, it is a program that matches the words you specify
  - For example, instead of saying "my head hurts", say "headache", because that's the term a medical page will use

Search in Google Laws (2)

- Describe what you need with as less terms
  - Since all words are used, each additional word limits the results
  - If you limit too much, you will miss a lot of useful information
- Choose descriptive words
  - The more unique the word is the more likely you are to get relevant results
  - Even if the word is correct, most people may use other word for the same concept

Search in Google Rules

- Search is always case insensitive
- Generally, punctuation is ignored, including @#$%^&*()=+[]\ and other special characters
- Functional words like 'the', 'a', 'and', and 'for' are usually ignored
- Synonyms might replace some words in your original query
- A particular word might not appear on a page in your results if there is sufficient other evidence that the page is relevant

Search in Google Tips

- Explicit phrase
  - Example: `"path in a graph"`
- Exclude words
  - Example: `path graph -tree`
- Site specific search
  - Search a specific website for content that matches a certain phrase
  - Example: `graph site:msdn.microsoft.com`

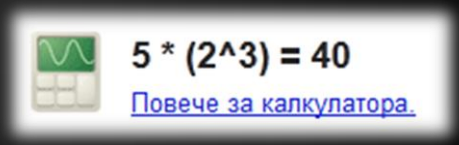
Search in Google Tips (2)

- Similar words and synonyms
  - If you want to include a word in your search, but want to include results that contain similar words or synonyms
  - Example: `"dijkstra" ~example`
- Specific Document Types
  - Example: `"dijkstra" filetype:cs`
- This OR That
  - Example: `"shortest path" graph OR tree`

Telerik Academy

Search in Google Tips (3)

- Numeric ranges
  - Example: `"prime numbers" 50..100`
- Units converter
  - Example: `10 radians in degrees`
- Calculator
  - Example: `5 * 2^3`



Telerik Academy

Search in Google Tips (4)

- Fill in the blanks (*)
  - It tells Google to try to treat the star as a placeholder for any unknown term(s) and then find the best matches
  - Example: `"* path in graph"`
    - Results: *shortest*, *longest*, *Hamiltonian*, *etc.*
- If you want to search C# code you can just add `"using System;"` to the search query
- www.google.com/advanced_search

Summary

- Problems 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 subproblems
  - Selecting appropriate data structures
  - Thinking about the efficiency and performance
  - Implementing step-by-step
  - Testing the nominal case, border cases and efficiency

Telerik Academy

Methodology of Problem Solving

http://academy.telerik.com

Free Trainings @ Telerik Academy

- C# Programming @ Telerik Academy
  - csharpfundamentals.telerik.com

- Telerik Software Academy
  - academy.telerik.com

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com