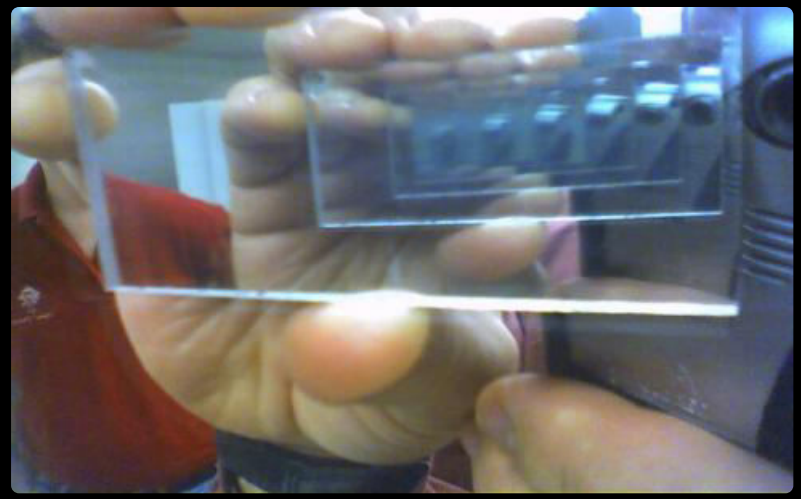
Recursion

Recursive Algorithms and Backtracking



SoftUni Team

Technical Trainers

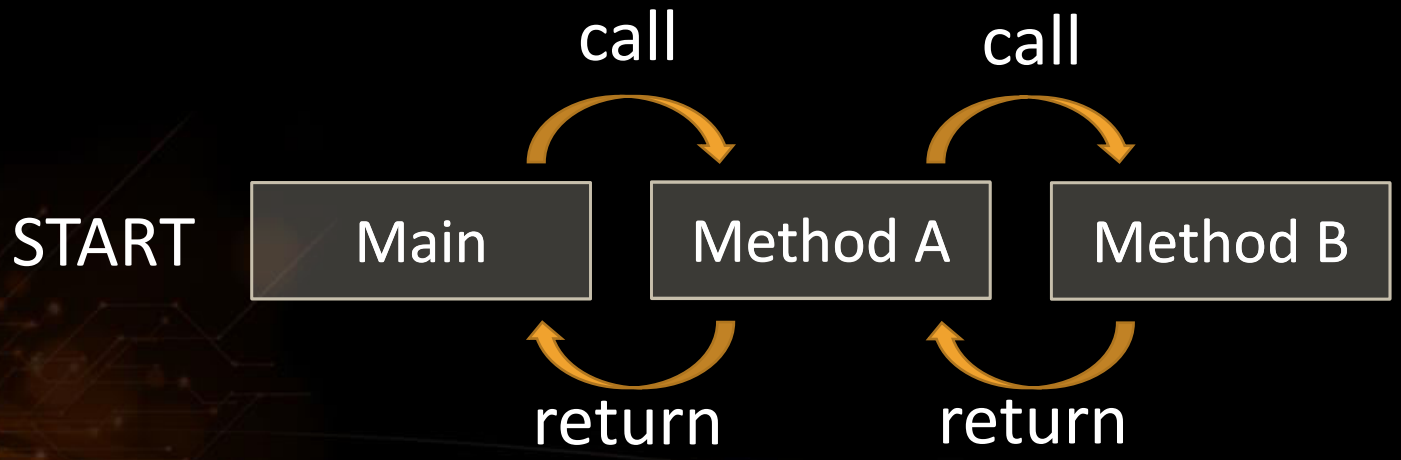
Software University

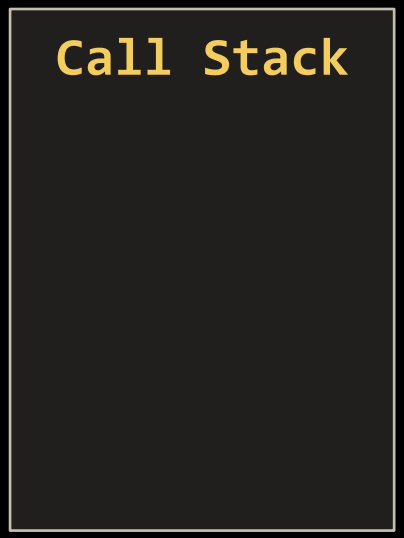
http://softuni.bg

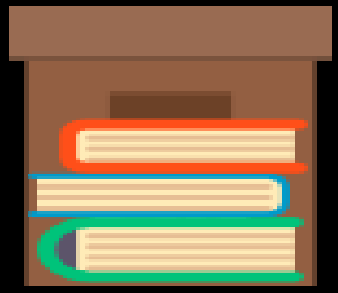
Table of Contents

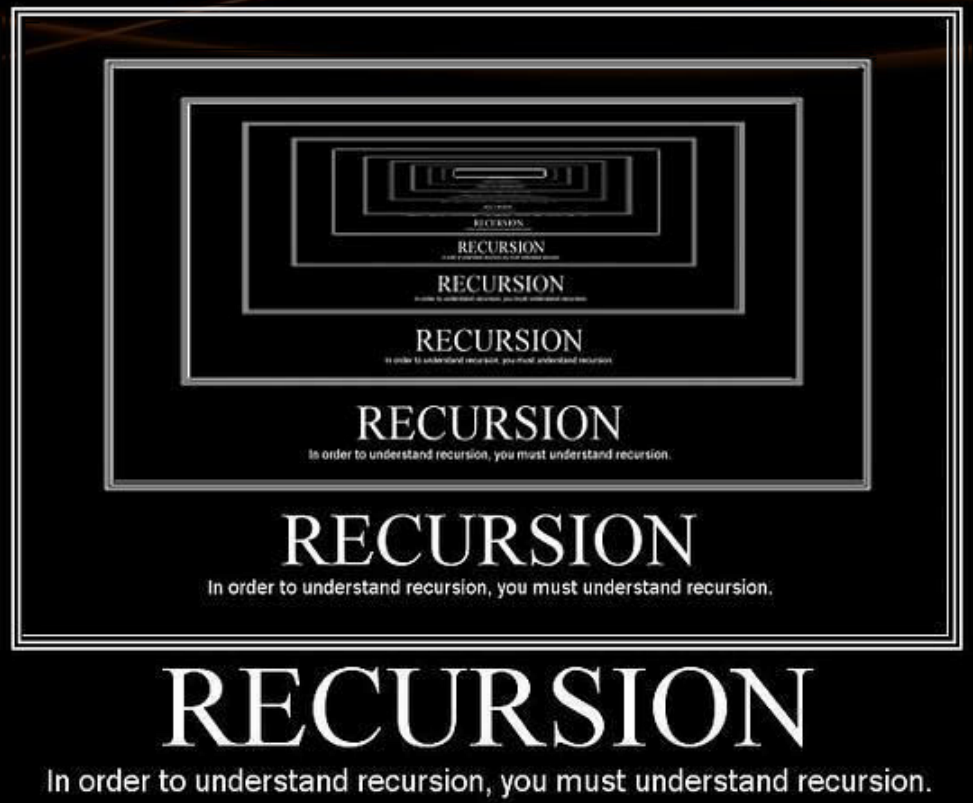
The Stack

- "The stack" is a small **fixed-size** chunk of memory (e.g. 1MB)
- Keeps track of **the point** to which each active subroutine should **return control** when it **finishes executing**





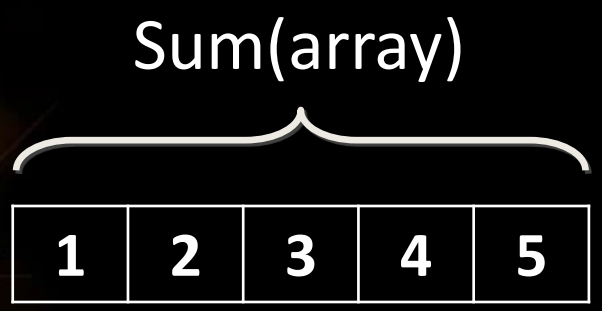


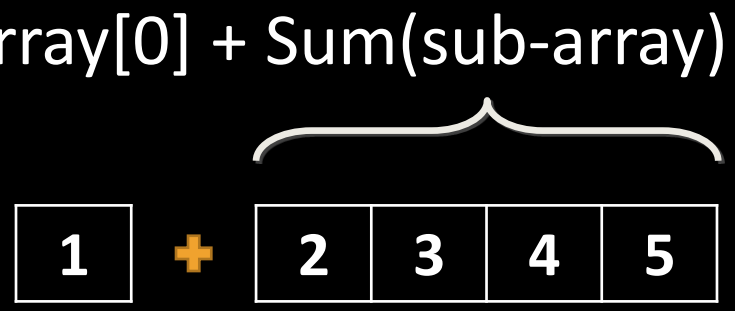


What is Recursion?

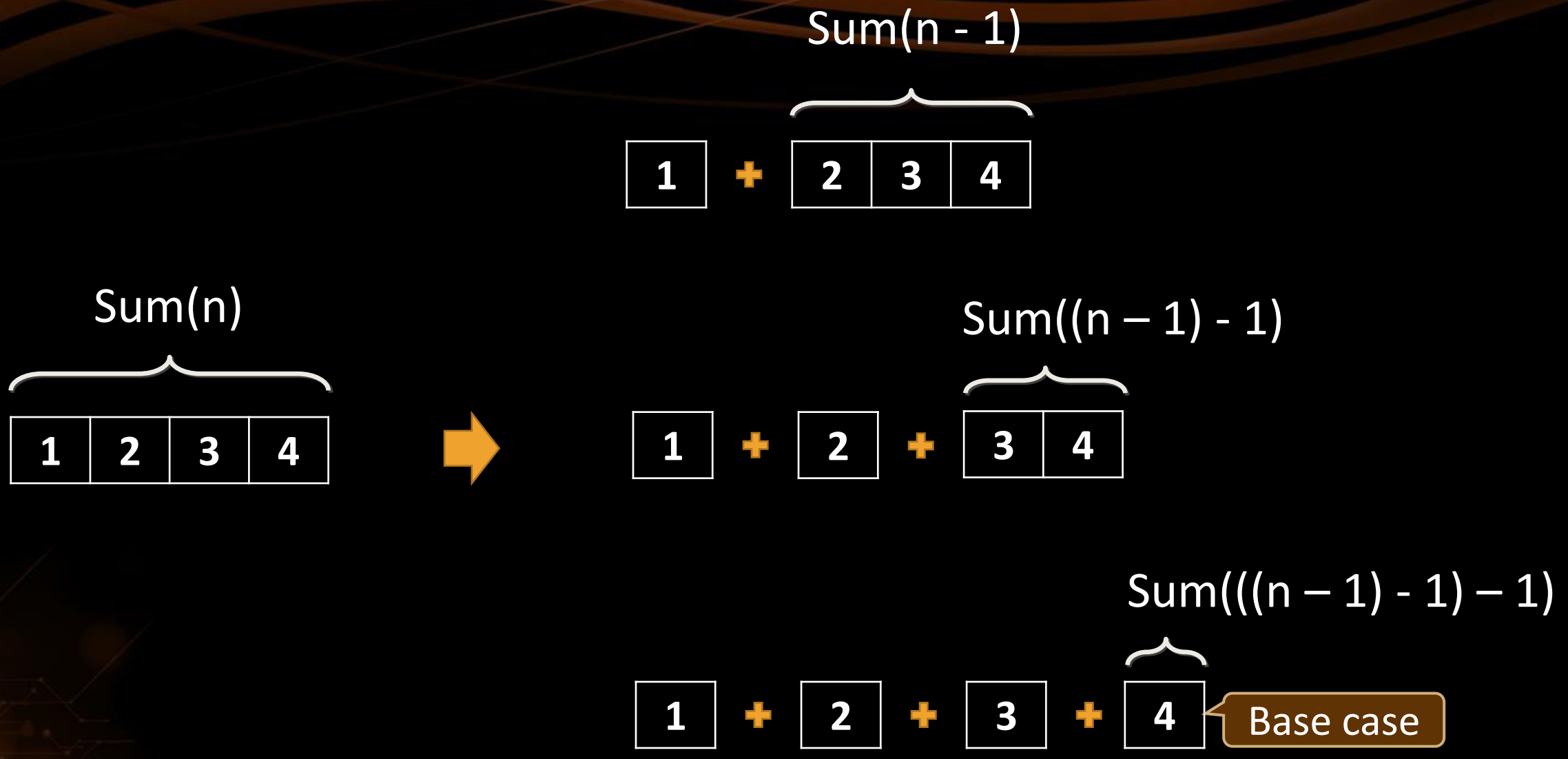
What is Recursion?

- Problem solving technique (In CS)
- Divides a problem into **subproblems of the same type**
  - Involves a **function calling itself**
  - The function should have a **base case**
  - **Each step** of the recursion should **move towards** the **base case**



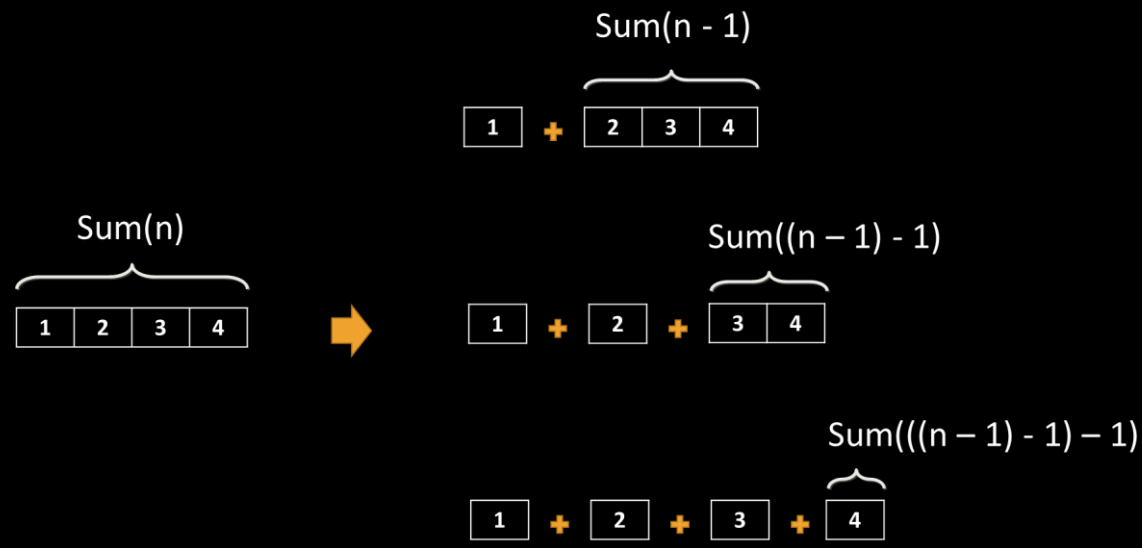


Array Sum – Example



Problem: Array Sum

- Create a **recursive method** that
  - Finds the sum of all numbers stored in an `int[] array`
  - Read numbers from the console



Solution: Array Sum

```
static int Sum(int[] array, int index)
{
  if (index == array.Length - 1)
  {
    return array[index];
  }

  return array[index] + Sum(array, index + 1);
}
```

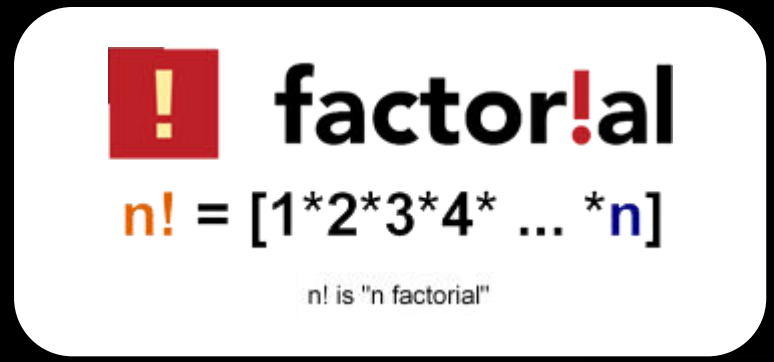
Base case

Recursive Factorial – Example

- Recursive definition of **n!** (n factorial):

```
n! = n * (n-1)! for n > 0
0! = 1
```

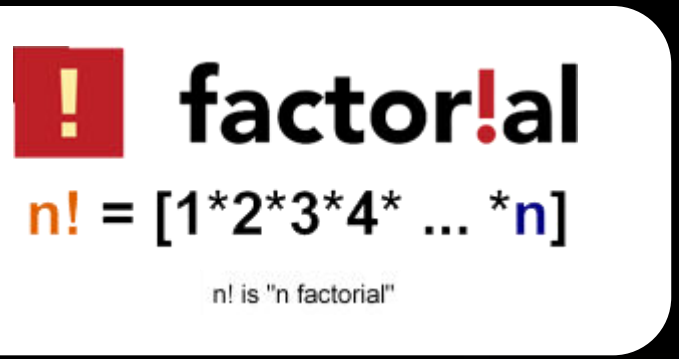
  - 5! = 5 * 4!
    - 4! = 4 * 3!
      - 3! = 3 * 2!
        - 2! = 2 * 1!
          - 1! = 1 * 0!
  - 0! = 1



Problem: Recursive Factorial

- Create a **recursive method** that calculates **n!**
  - Read n from the console

`5`	→	`120`
`10`	→	`3628800`

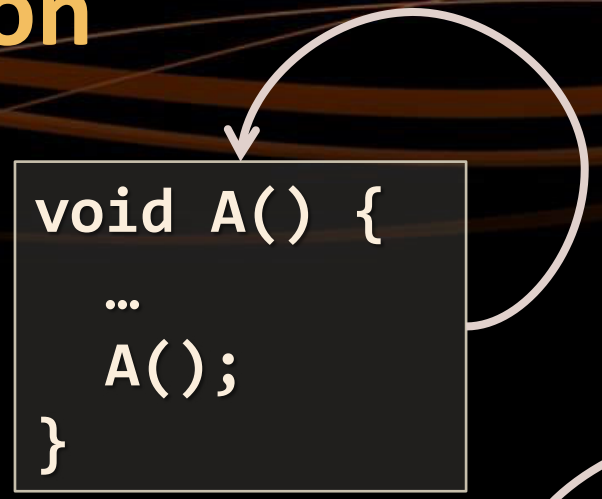


Solution: Recursive Factorial

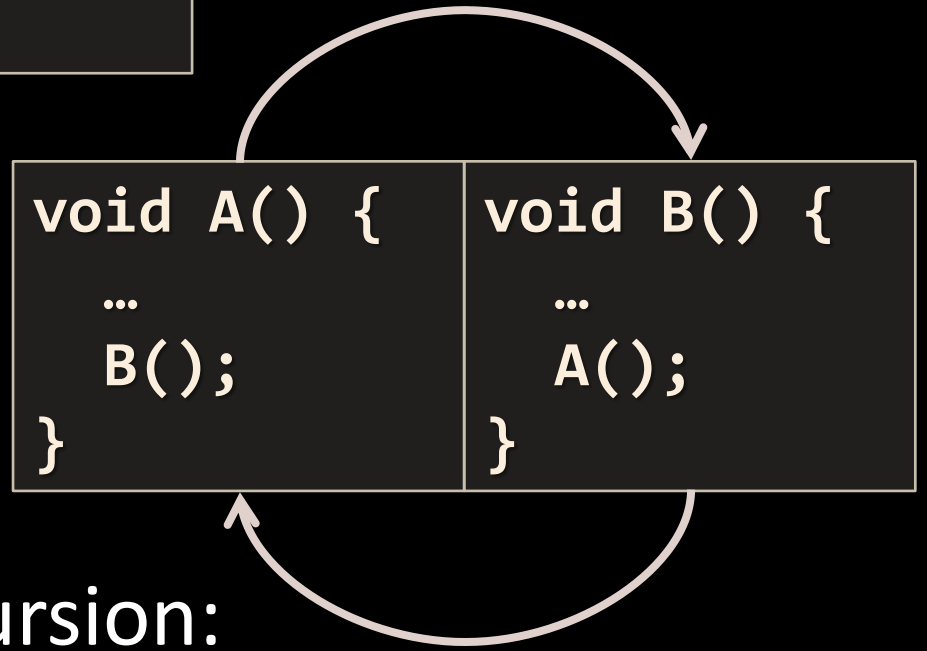
```
static long Factorial(int num)
{
  if (num == 0)
  {
    return 1;
  }

  return num * Factorial(num - 1);
}
```

Base case

Direct and Indirect Recursion



- Direct recursion
  - A method directly calls itself
- Indirect recursion
  - Method A calls B, method B calls A
  - Or even A → B → C → A



- Funny example of infinite indirect recursion:
  - http://www.nakov.com/blog/2013/01/23/indirect-recursion/

Recursion Pre-Actions and Post-Actions

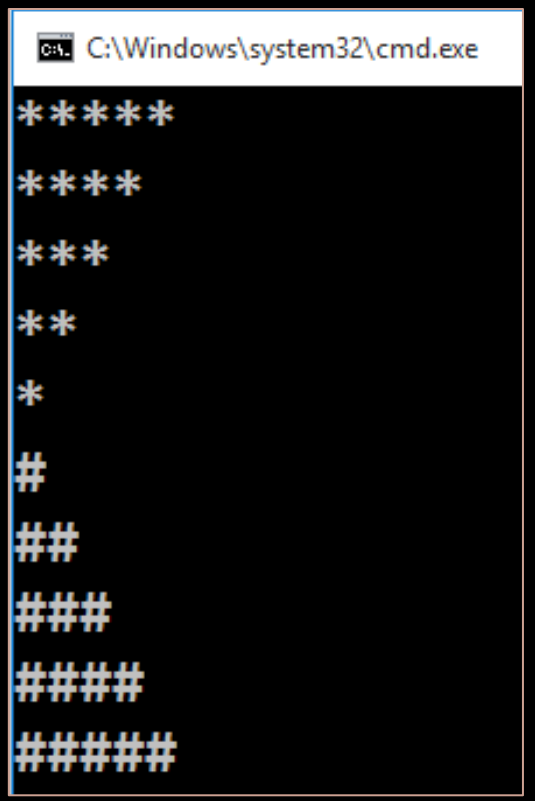
- Recursive methods have 3 parts:
  - Pre-actions (before calling the recursion)
  - Recursive calls (step-in)
  - Post-actions (after returning from recursion)

```
static void Recursion()
{
  // Pre-actions
  Recursion();
  // Post-actions
}
```

Problem: Recursive Drawing

- Create a **recursive method** that draws the following figure

5



Pre-Actions and Post-Actions – Example


```
static void PrintFigure(int n)
{
    if (n == 0) // Bottom of the recursion
      return;

    // Pre-action: print n asterisks
    Console.WriteLine(new string('*', n));

    // Recursive call: print figure of size n-1
    PrintFigure(n - 1);

    // Post-action: print n hashtags
    Console.WriteLine(new string('#', n));
}
```

SOFTWARE UNIVERSITY FOUNDATION

Recursion

Live Exercises in Class (Lab)

SOFTWARE UNIVERSITY FOUNDATION

Generating Combinations

Recursive Algorithm

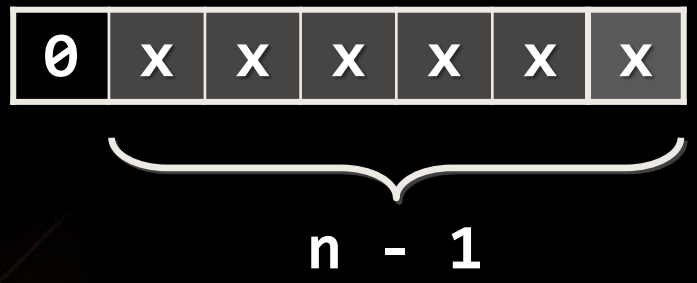
Generating 0/1 Vectors

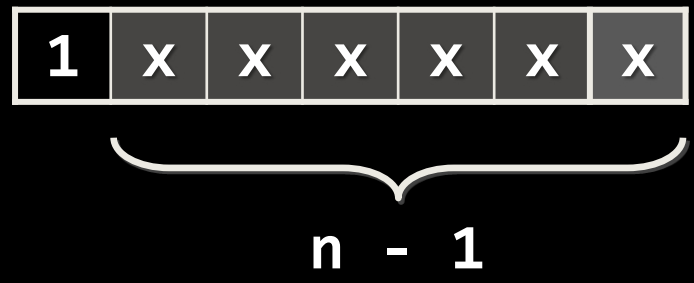
- How to generate all 8-bit vectors recursively?
  - 0 0 0 0 0 0 0 0
  - 0 0 0 0 0 0 0 1
  - …
  - 0 1 1 1 1 1 1 1
  - 1 0 0 0 0 0 0 0
  - …
  - 1 1 1 1 1 1 1 0
  - 1 1 1 1 1 1 1 1

Generating 0/1 Vectors

- Start with a **blank vector**

- Choose the **first position** and **loop through all possibilities**

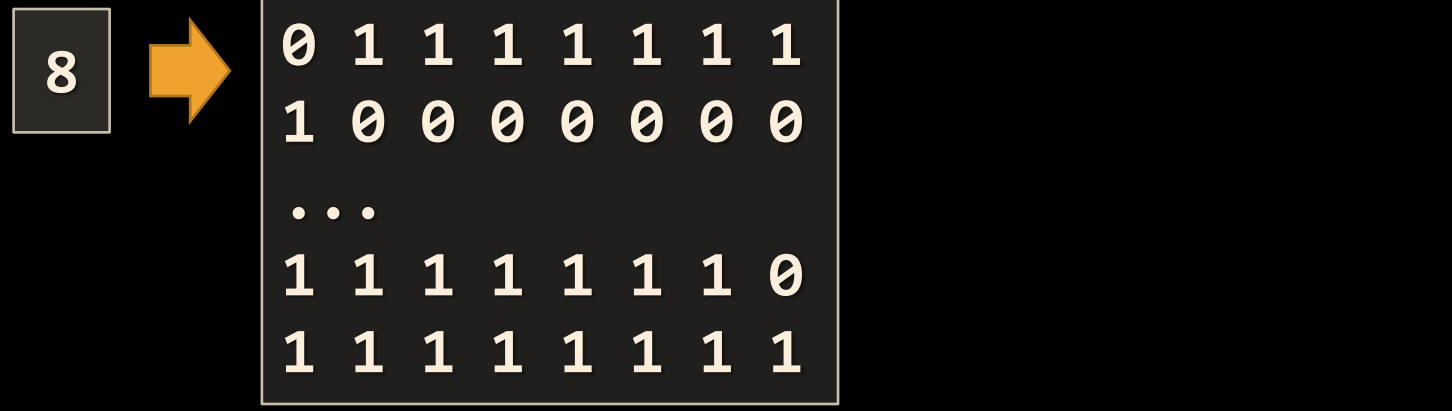




- For each possibility, generate all **(n – 1)-bit** vectors

Problem: Generate n-bit Vectors

- Generate all **n-bit** vectors as shown below
- Read n from the console

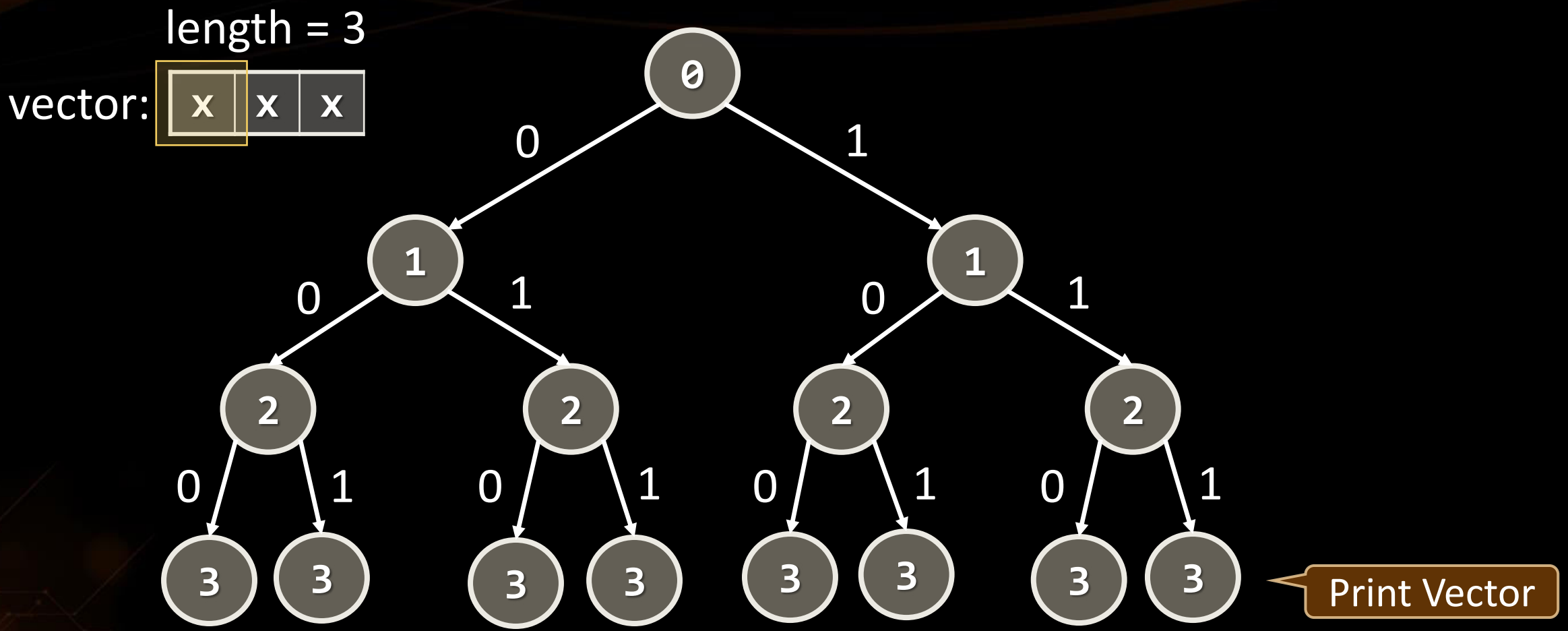


Solution: Generate n-bit Vectors

```
static void Gen01(int index, int[] vector)
{
  if (index >= vector.Length)
    Print(vector);
  else
    for (int i = 0; i <= 1; i++)
    {
      vector[index] = i;
      Gen01(index + 1, vector);
    }
}
```

```
static void Main()
{
  int n = 8;
  int[] vector = new int[n];
  Gen01(0, vector);
}
```

Generating 3-bit Vectors Recursion Tree



Generating 3-bit Vectors Trace

	vector	index	
Gen()	0 0 0	2	
Gen()	0 0 0	1	
Gen()	0 0 0	0	
Gen()	0 0 0	-1	Print Vector
Gen()	0 0 1	0	
Gen()	0 0 1	-1	Print Vector
Gen()	0 0 1	0	
Gen()	0 1 0	1	
Gen()	0 1 0	0	Print Vector
Gen()	0 1 0	-1	
Gen()	0 1 1	0	Print Vector
Gen()	0 1 1	-1	
...	...	...	

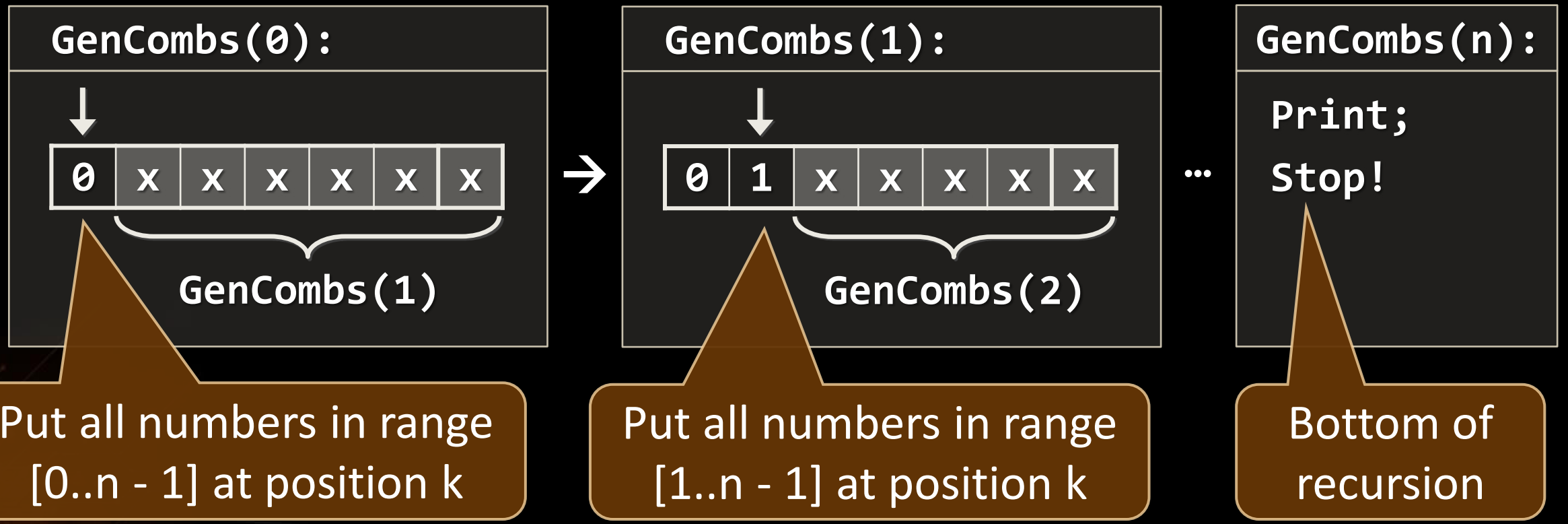
Generating Combinations

- Combinations - all the ways to extract a subset from a set
  - Select **k** members from a set of **n** elements
- Example:
  - **2** different elements from **{1, 2, 3, 4}** → 6 different ways:

(1, 2) (1, 3) (1, 4) (2, 3) (2, 4) (3, 4)

Generating Combinations (2)

- Algorithm `GenCombs(k)`: put the numbers `[0 … n]` at position `k` and call `GenCombs(k+1)` recursively for the rest of the elements:

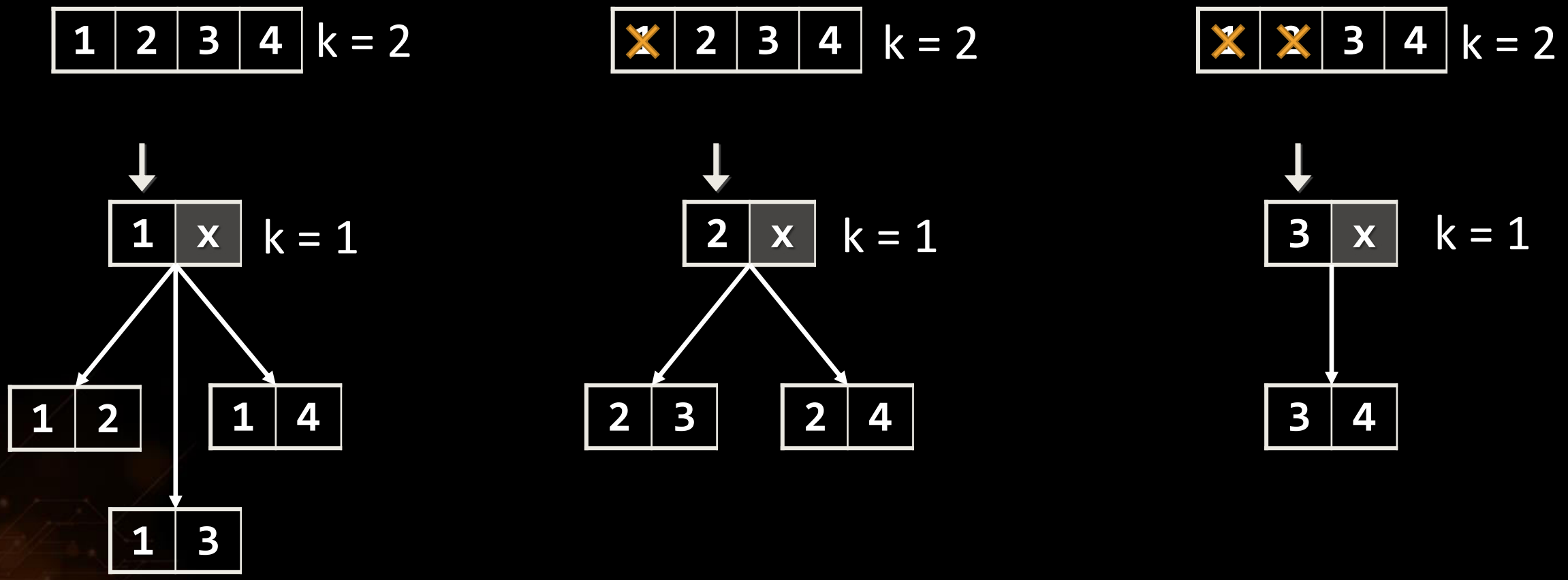


Generating Combinations (3)

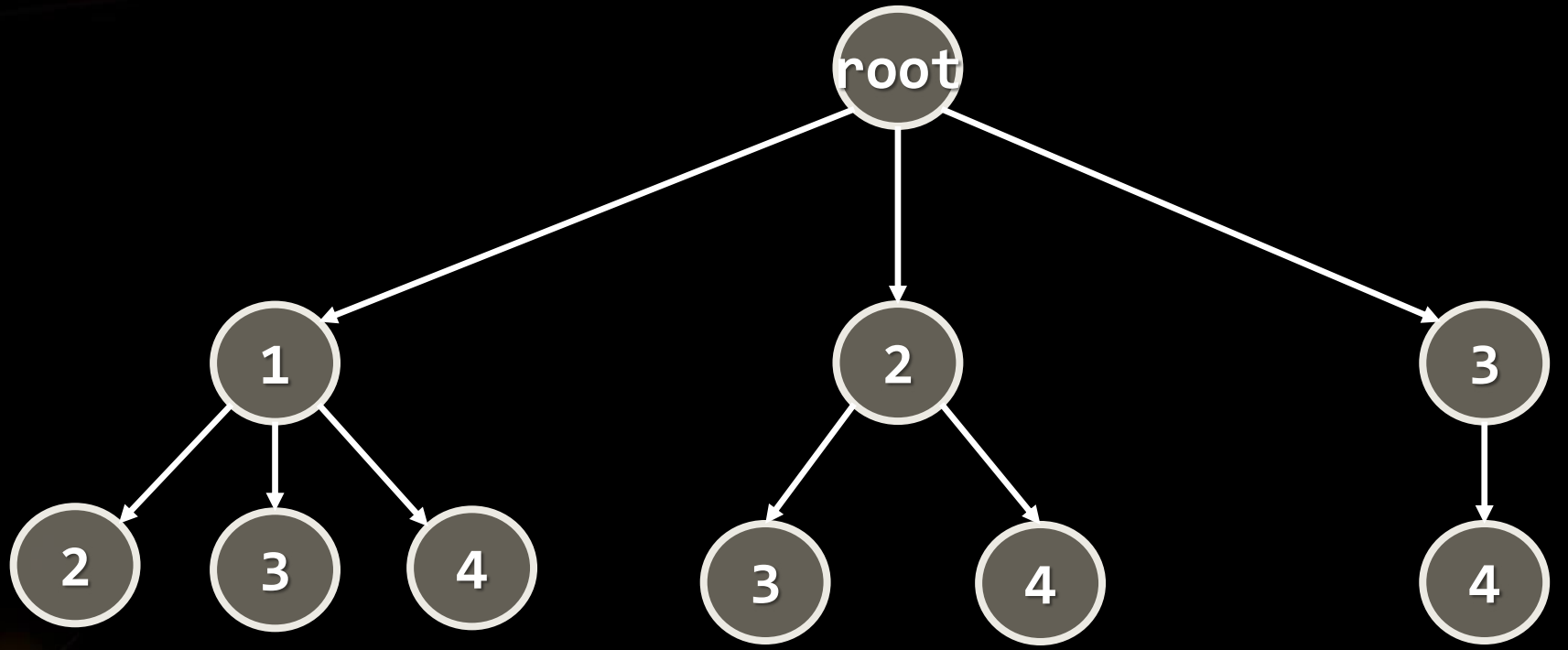
SOFTWARE UNIVERSITY FOUNDATION

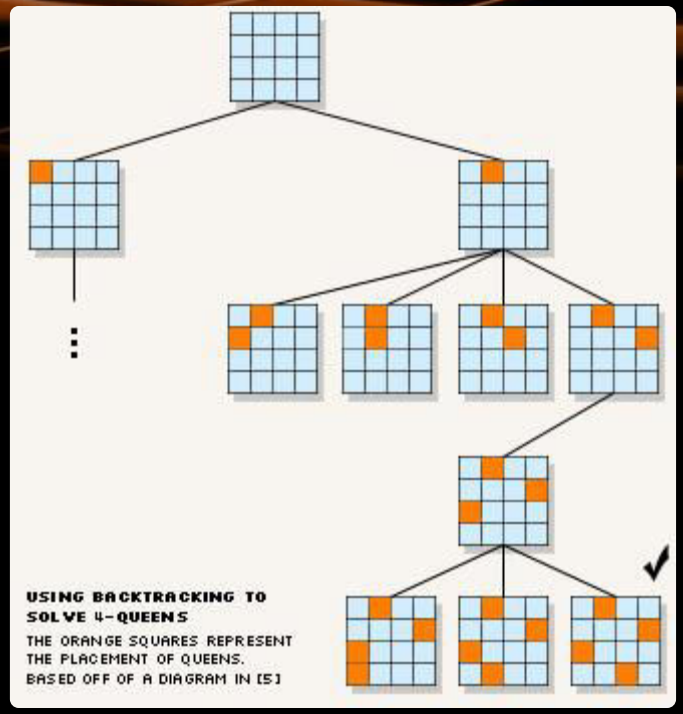
```
private static void GenCombs(
        int[] set, int[] vector, int index, int border)
{
  if (index >= vector.Length)
    // Print vector
  else
    for (int i = border + 1; i < set.Length; i++)
    {
      vector[index] = set[i];
      GenCombs(set, vector, index + 1, i);
    }
  }
}
```

Generating Combinations (3)



Generating {1, 2, 3, 4}, k = 2



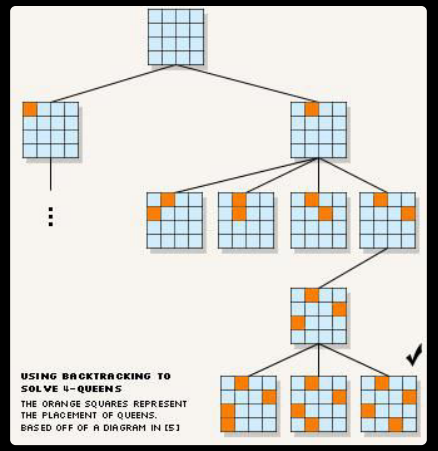


Backtracking

Generating All Candidates

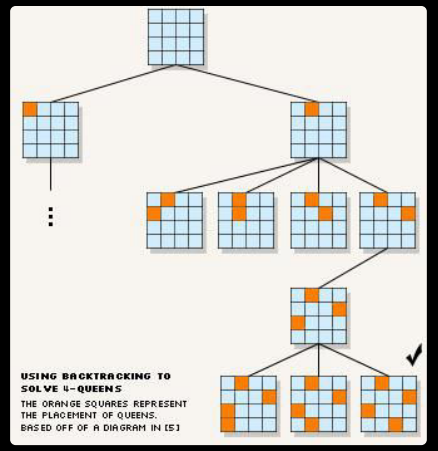
Backtracking

- What is **backtracking**?
  - Class of algorithms for **finding all solutions** to some combinatorial problem
    - E.g. find all paths from Sofia to Varna



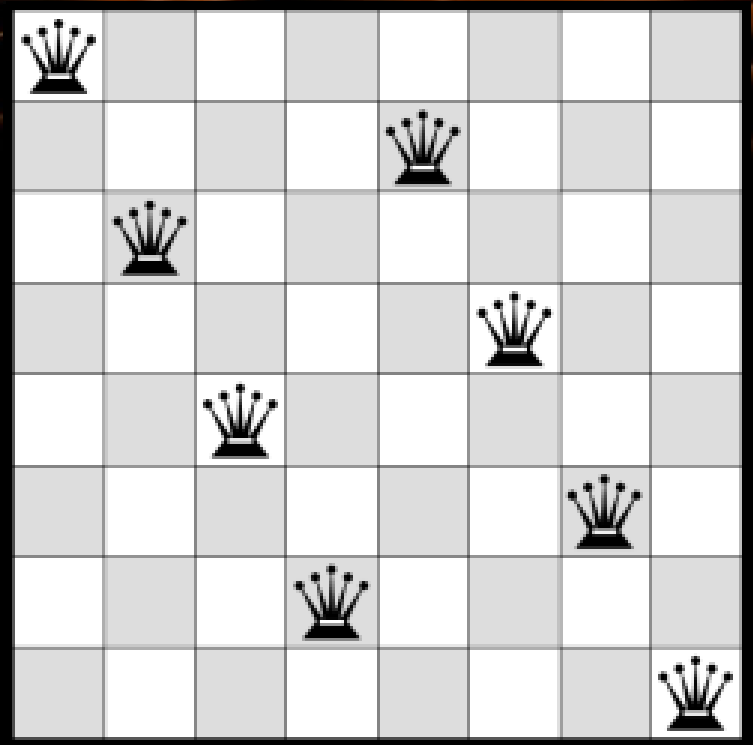
Backtracking

- How does backtracking work?
  - At each step **tries all perspective possibilities** recursively
  - **Drop** all **non-perspective possibilities** as early as possible
- Backtracking has **exponential running time**!



Backtracking Algorithm (Pseudocode)


```
void Backtracking(Node node)
{
  if (node is solution)
    PrintSolution(node);
  else
    for each child c of node
      if (c is perspective candidate)
      {
        MarkPositionVisited(c);
        Backtracking(c);
        UnmarkPositionVisited(c);
      }
}
```

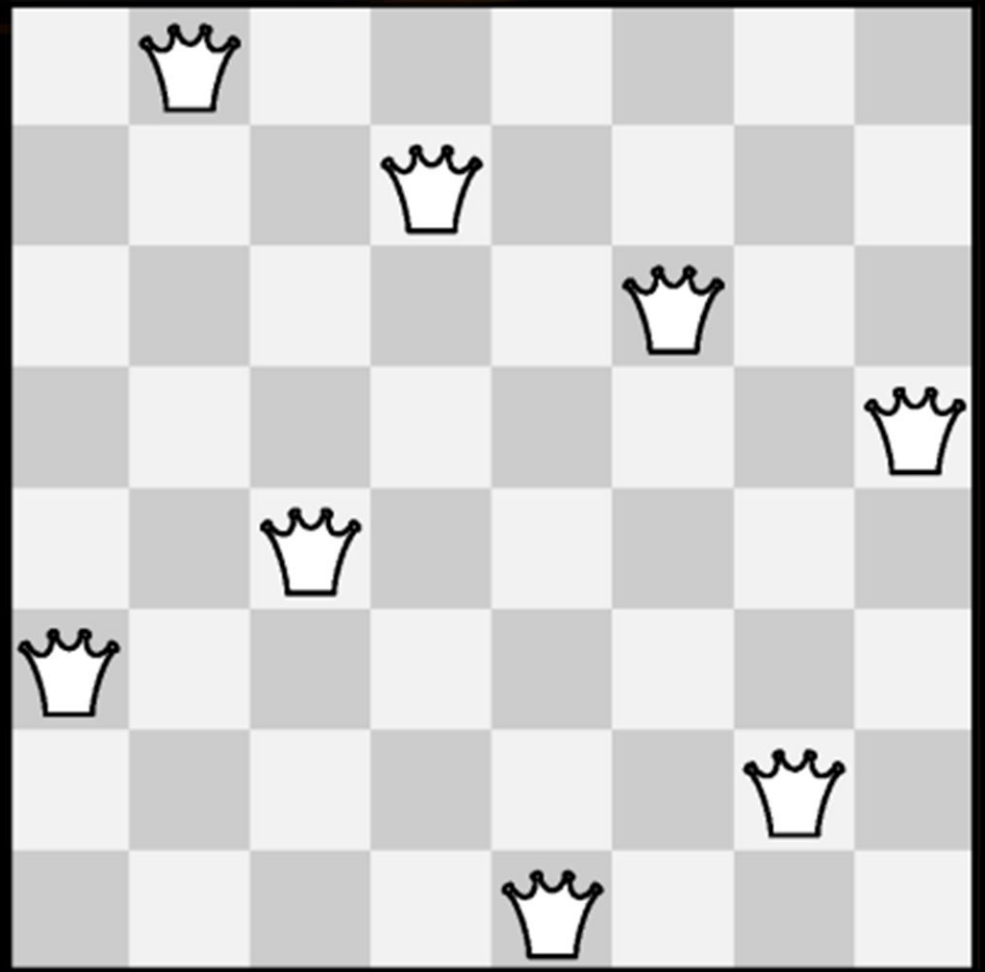


The "8 Queens" Puzzle

Backtracking in Practice

The "8 Queens" Puzzle

- Write a program to find all possible placements of
  - 8 queens on a chessboard
  - So that no two queens can attack each other
  - http://en.wikipedia.org/wiki/Eight_queens_puzzle

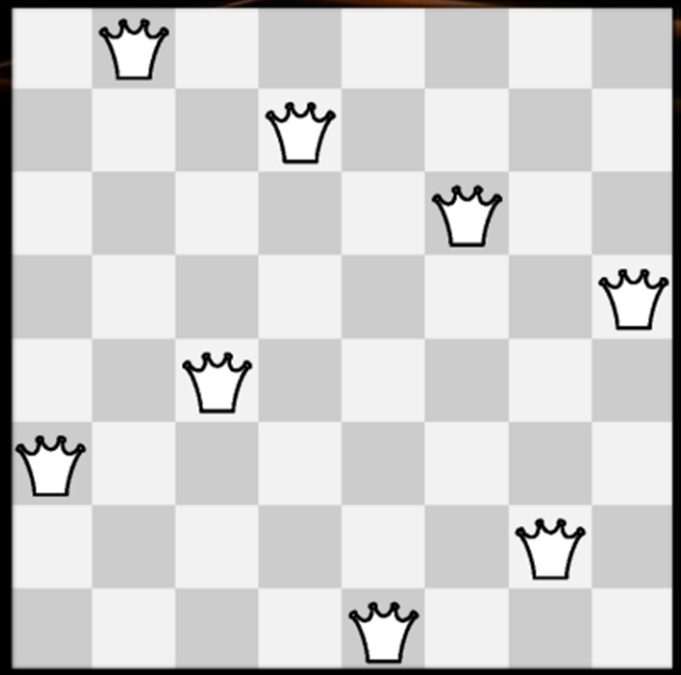


Solving The "8 Queens" Puzzle

- Find all solutions to "8 Queens Puzzle"
- At each step:
  - Put a queen at free position
  - Recursive call
  - Remove the queen

```
void PutQueens(row)
{
  if (row == 8)
    PrintSolution();
  else
    for (col = 0 … 7)
      if (CanPlaceQueen(row, col))
      {
        MarkAllAttackedPositions(row, col);
        PutQueens(row + 1);
        UnmarkAllAttackedPositions(row, col);
      }
}
```

SOFTWARE UNIVERSITY FOUNDATION

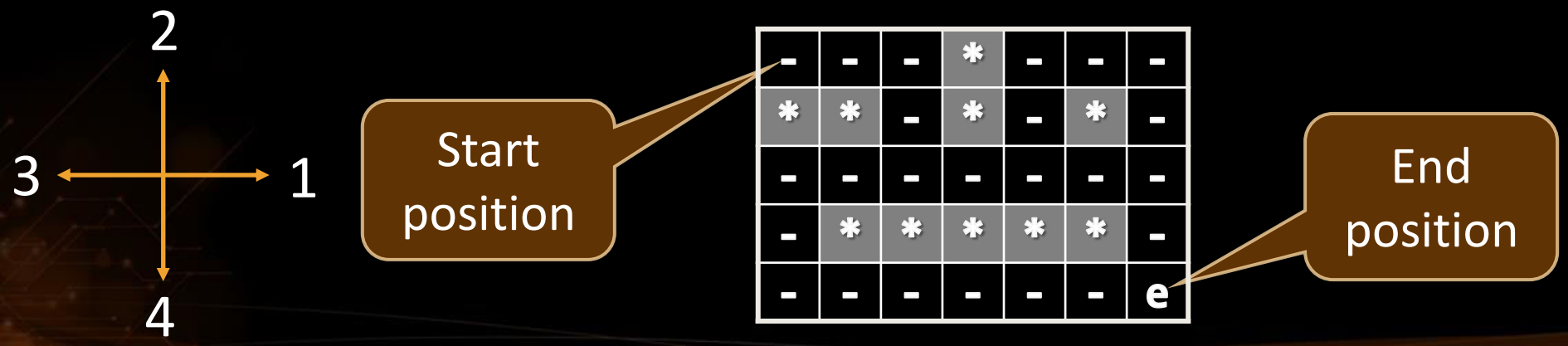


The "8 Queens" Puzzle

In-Class Exercise (Lab)

Finding All Paths in a Labyrinth

- We are given a **labyrinth**
  - Represented as matrix of cells of size M x N
  - Empty cells (-) are passable, the others (*) are not
- We **start from the top left** corner and **can move in all 4 directions**
- We want to **find all paths to the exit**, marked '**e**'



Finding All Paths in a Labyrinth (2)

- There are **3 different paths** from the top left corner to the bottom right corner:

1)

0	1	2	*			
*	*	3	*		*	
6	5	4				
7	*	*	*	*	*	
8	9	10	11	12	13	14

RRDDLLDDRRRRRR

2)

0	1	2	*	8	9	10
*	*	3	*	7	*	11
		4	5	6		12
	*	*	*	*	*	13
						14

RRDDRRUURRDDDD

3)

0	1	2	*			
*	*	3	*		*	
		4	5	6	7	8
	*	*	*	*	*	9
						10

RRDDRRRRDD

Find All Paths: Algorithm

- DS → matrix of characters:

```
static char[,] lab =
{
    {'-', '-', '-', '*', '-', '-', '-'},
    {'*', '*', '-', '*', '-', '*', '-'},
    {'-', '-', '-', '-', '-', '-', '-'},
    {'-', '*', '*', '*', '*', '*', '-'},
    {'-', '-', '-', '-', '-', '-', 'e'},
};
```

- Dashes '**-**' are **passable** cells
- Asterisks '*****' are **not passable**
- The symbol '**e**' is the **exit** (can occur multiple times)

Find All Paths: Algorithm (2)

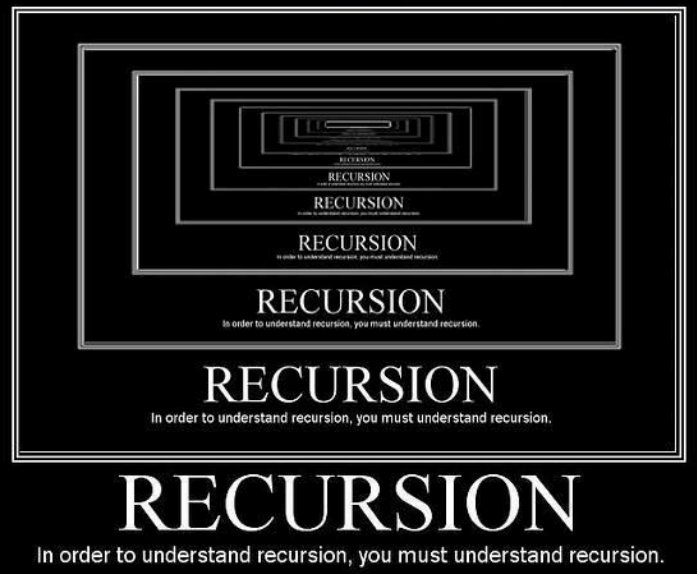

```
private static void FindPath(int row, int col)
{
  if (!IsInBounds(row, col)) { return; }

  if (IsExit(row, col)) { Console.WriteLine("Path found!"); }
  else if (!IsVisited(row, col) && IsPassable(row, col))
  {
    Mark(row, col);
    FindPath(row, col + 1); // Right
    FindPath(row + 1, col); // Down
    FindPath(row, col - 1); // Left
    FindPath(row - 1, col); // Up
    Unmark(row, col);
  }
}
```

Find All Paths and Print Them

- Create a `List<char>` that will store the path
- Pass a direction at each recursive call (**L**, **R**, **U** or **D**)
- At the start of each recursive call → **Add direction**
- At the end of each recursive call → **Remove last direction**

Find All Paths and Print Them (2)


```
private static void FindPath(int row, int col, char direction)
{
  if (!IsInBounds(row, col)) { return; }

  path.Add(direction);

  if (IsExit(row, col)) { PrintPath(); }
  else if (!IsVisited(row, col) && IsFree(row, col))
  {
    Mark(row, col);
    FindPath(row, col + 1, 'R');
    FindPath(row + 1, col, 'D');
    FindPath(row, col - 1, 'L');
    FindPath(row - 1, col, 'U');
    Unmark(row, col);
  }

  path.RemoveAt(path.Count - 1);
}
```



Recursion or Iteration?

When to Use and When to Avoid Recursion?

Performance: Recursion vs. Iteration

- Recursive calls are slightly slower than iteration
  - Parameters and return values travel through the stack at each step
  - Prefer iteration for linear calculations (without branched calls)

Recursive factorial:

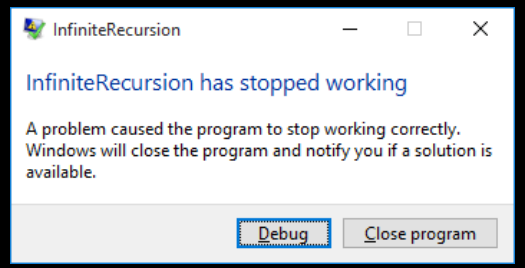
```
static long RecurFact(int n)
{
  if (n == 0)
    return 1;
  else
    return n * Fact(n - 1);
}
```

Iterative factorial:

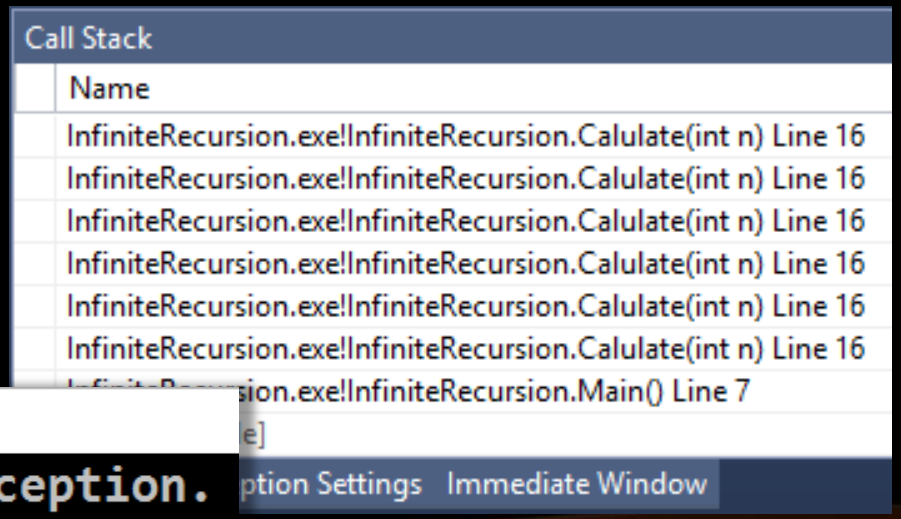
```
static long IterFact(int num)
{
  long result = 1;
  for (int i = 1; i <= n; i++)
    result *= i;
  return result;
}
```

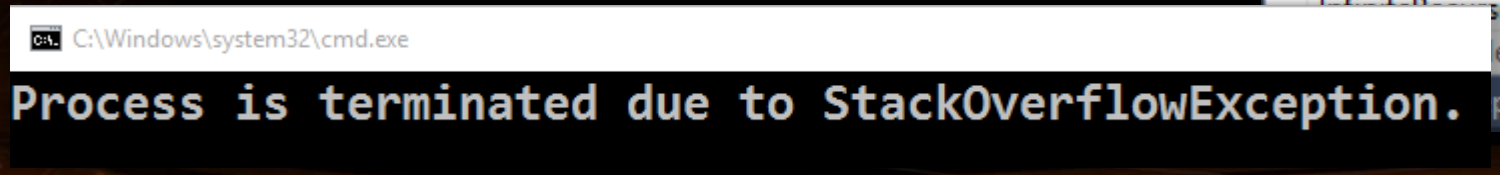
Infinite Recursion

- Infinite recursion == a method calls itself infinitely
  - Typically, infinite recursion == bug in the program
  - The bottom of the recursion is missing or wrong
  - In C# / Java / C++ causes "stack overflow" error




```
static long Calulate(int n)
{
    return Calulate(n + 1);
}
```





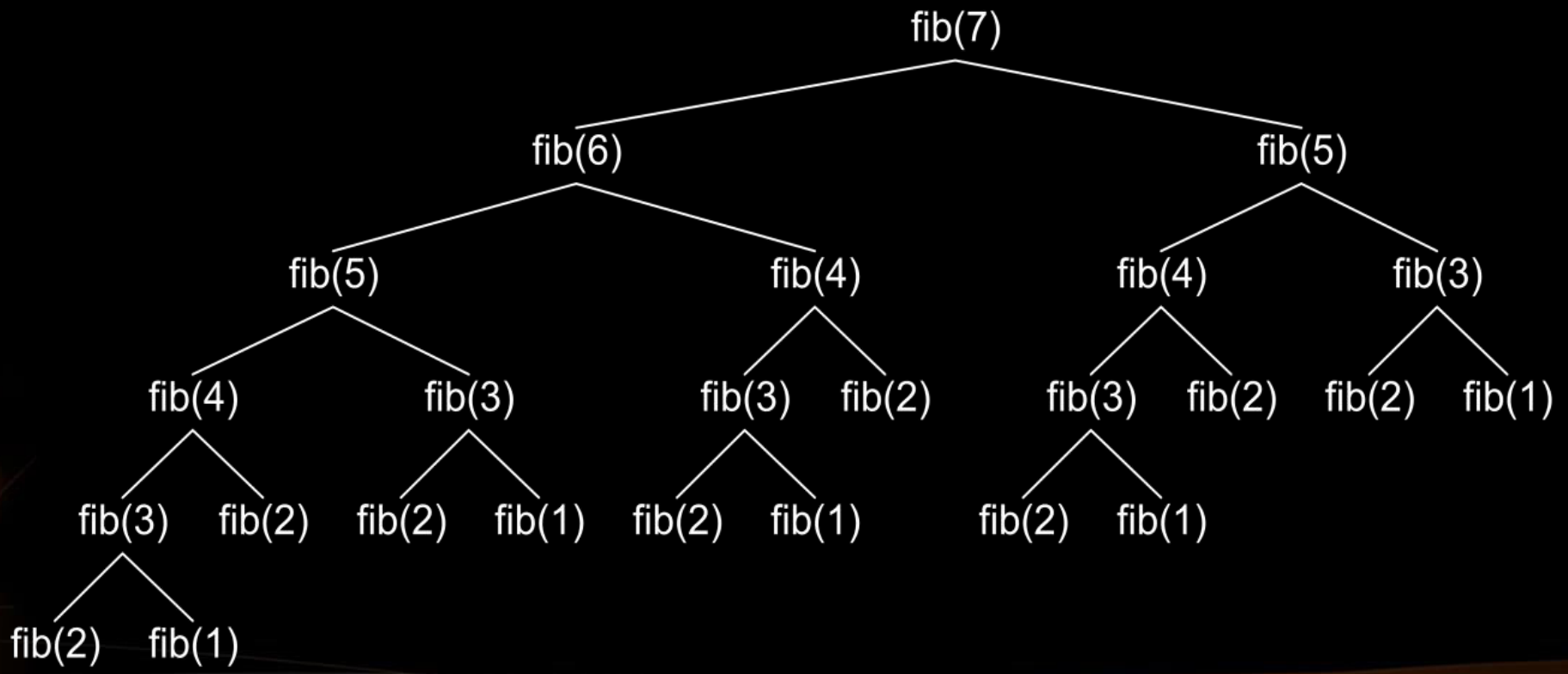
Recursion Can be Harmful!

- When used incorrectly recursion could take too much memory and computing power

```
static decimal Fibonacci(int n)
{
    if ((n == 1) || (n == 2))
        return 1;
    else
        return Fibonacci(n - 1) + Fibonacci(n - 2);
}

static void Main()
{
    Console.WriteLine(Fibonacci(10)); // 89
    Console.WriteLine(Fibonacci(50)); // This will hang!
}
```

How the Recursive Fibonacci Calculation Works?

- `fib(n)` makes about `fib(n)` recursive calls
- The same value is calculated many, many times!



When to Use Recursion?

- Avoid recursion when an obvious iterative algorithm exists
  - Examples: factorial, Fibonacci numbers
- Use recursion for combinatorial algorithms where
  - At each step you need to recursively explore more than one possible continuation, i.e. branched recursive algorithms

Summary

- Recursion means to call a method from itself
  - It should always have a bottom at which the recursive calls stop
  - Very powerful technique for implementing combinatorial algorithms
  - Examples: generating combinatorial configurations like vectors, permutations, combinations, variations, etc.
- Backtracking finds all solutions / optimal solution of combinatorial problem by generating all possibilities
  - Without non-perspective candidates
- Recursion can be harmful when not used correctly

Recursion

Questions?

https://softuni.bg/trainings/1688/algorithms-july-2017

License

- 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

Free Trainings @ Software University

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