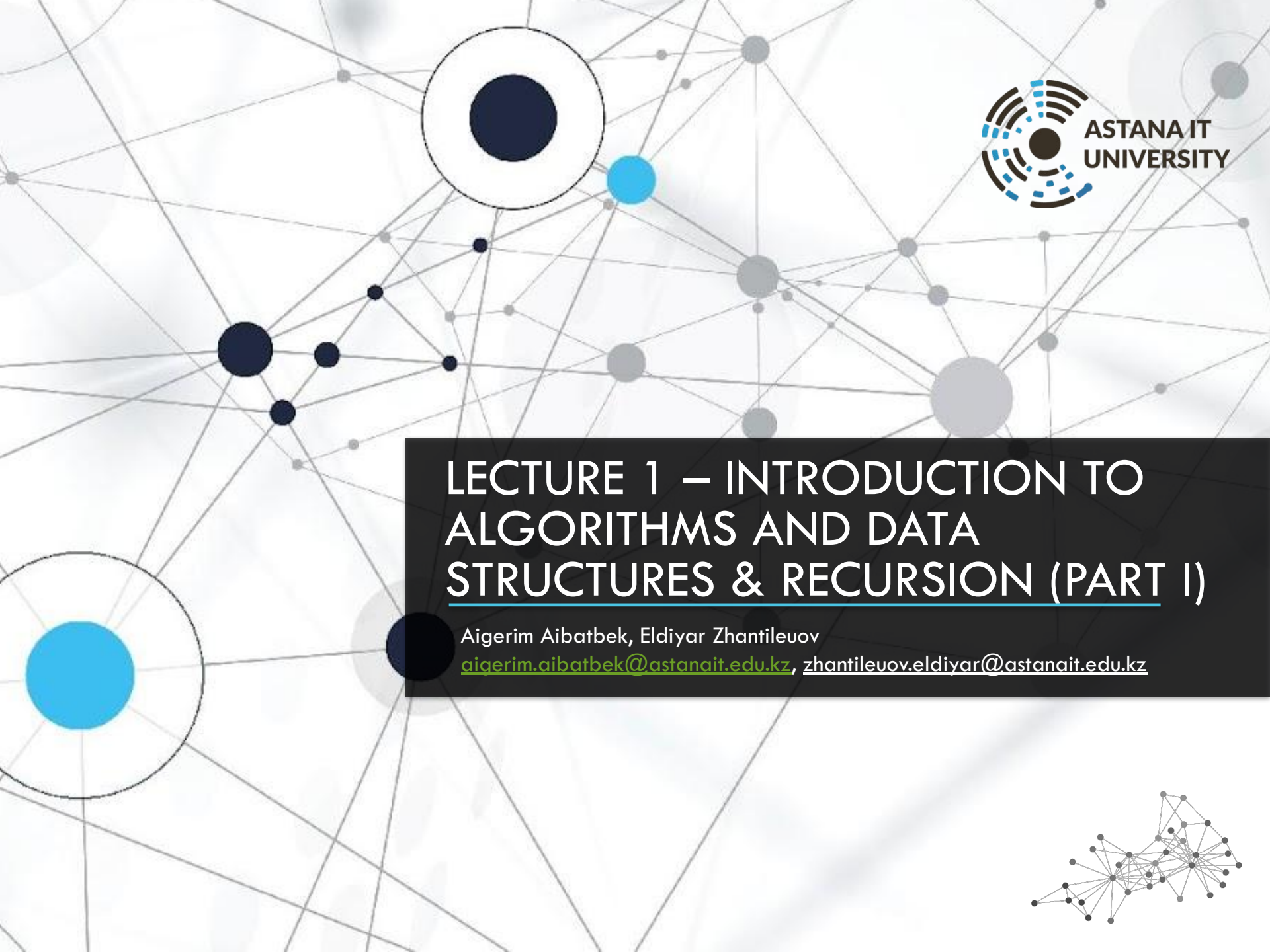


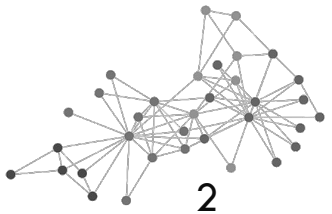
LECTURE 1 – INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES & RECURSION (PART I)

Aigerim Aibatbek, Eldiyar Zhantileuov
aigerim.aibatbek@astanait.edu.kz, zhantileuov.eldiyar@astanait.edu.kz

ASTANA IT UNIVERSITY

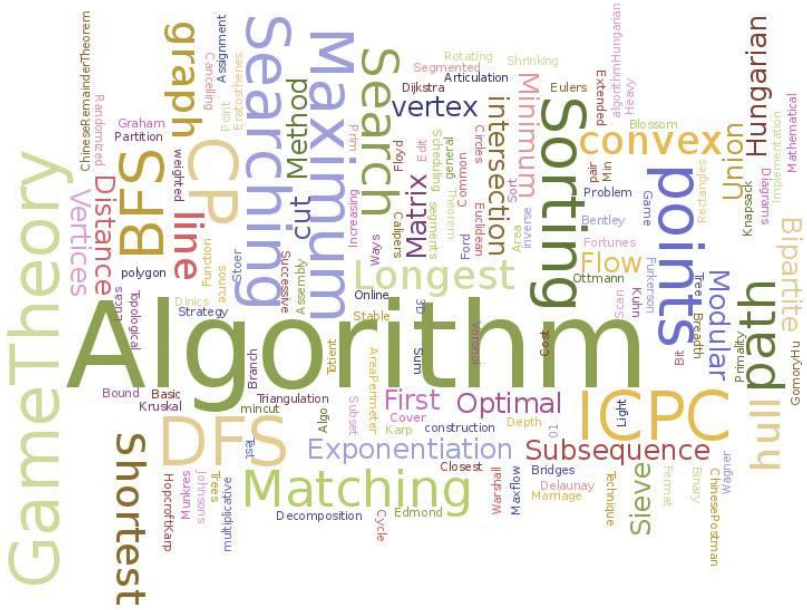
CONTENT

1. Introduction to the Algorithms and Data Structure
2. Function Review
3. Function Call and Stack
4. Recursion Overview
5. Simple Example
6. Recursion Sum Up



INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES

- **The main focus** of the course is designed on solving computational problems that involve collections of data. You will study a core set of data abstractions, data structures, and algorithms that provide a foundation for creating and maintaining efficient programs.

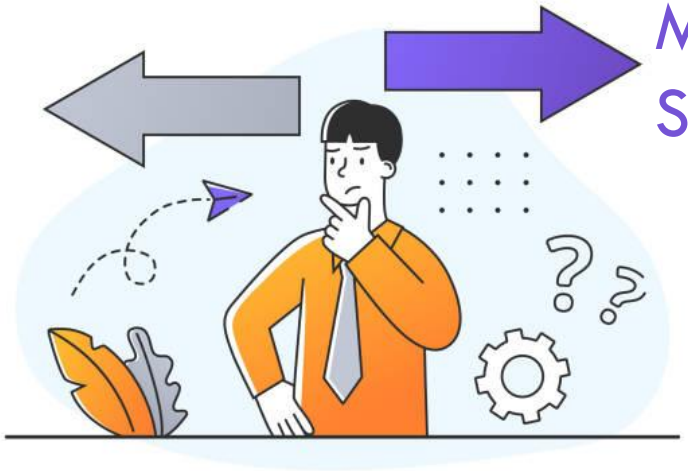


INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES

By the end of this course the you will be able to:

- Choose appropriate algorithms and data structures for storing data, searching and sorting, as well as implementing those algorithms.

Quick Sort

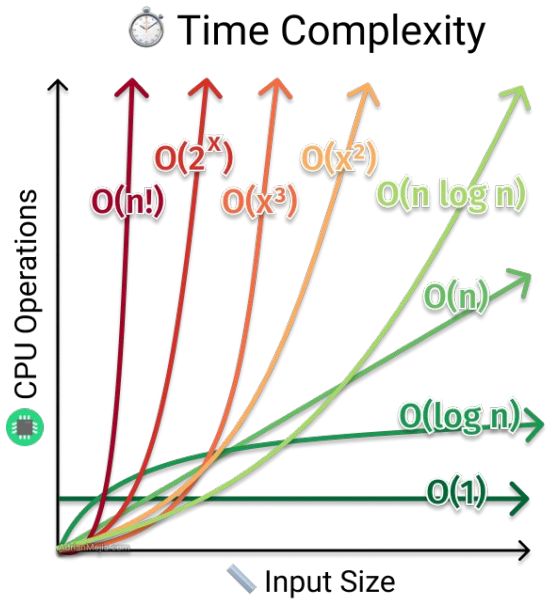


Merge Sort

INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES

By the end of this course the you will be able to:

- Analyze the runtime performance of various algorithms and programs in terms of the size of their inputs, averages, best, and worst cases.



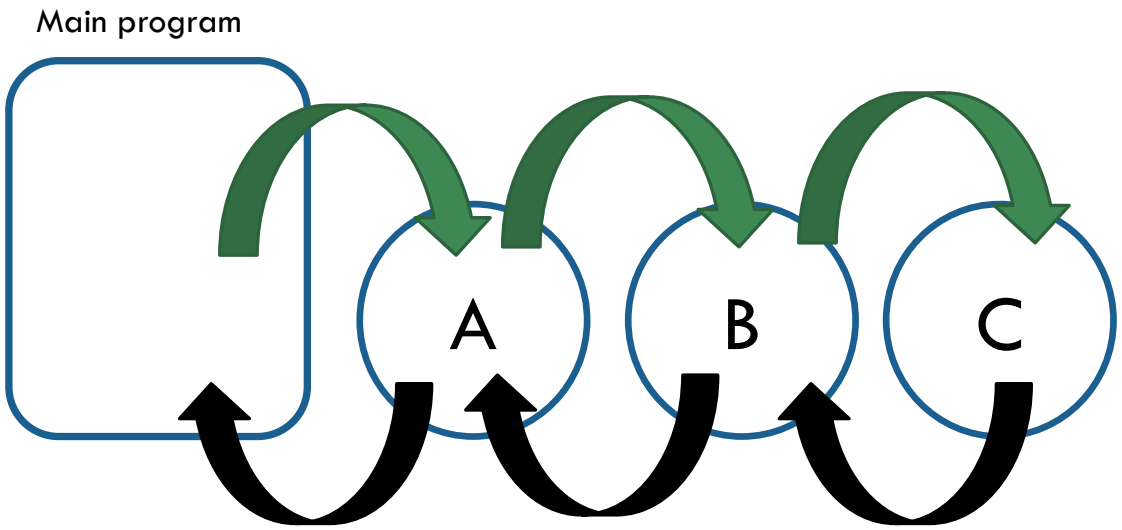
FUNCTION REVIEW

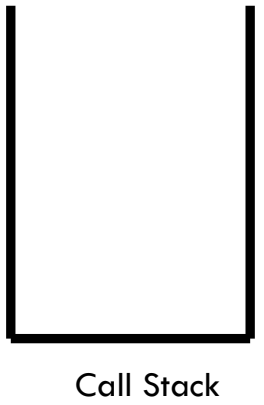
Main program

void doSmth(){}

When you call a function from another function, the calling function is **paused in partially completed state.**

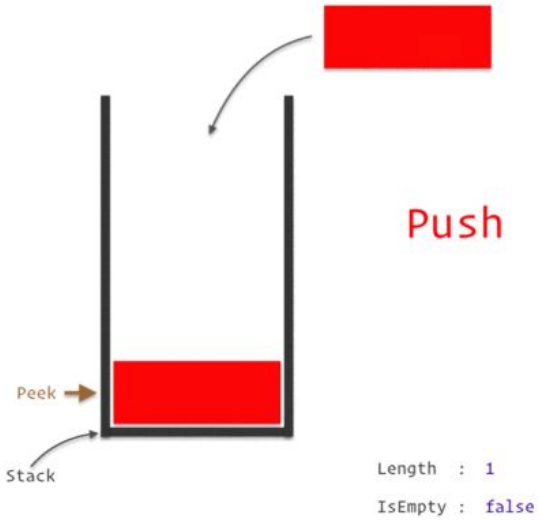
FUNCTION CALL AND STACK





FUNCTION CALL AND STACK CONTINUES

- When you run a program, the computer creates a **stack** for you
- Each time you invoke a function, the function is placed to the stack
- A stack is a **last-in/first-out** memory structure. The first item referenced or removed from a stack is always the last item entered into the stack.
- If some function call has produced an excessively long chain of recursive calls, it can lead to **stack overflow**

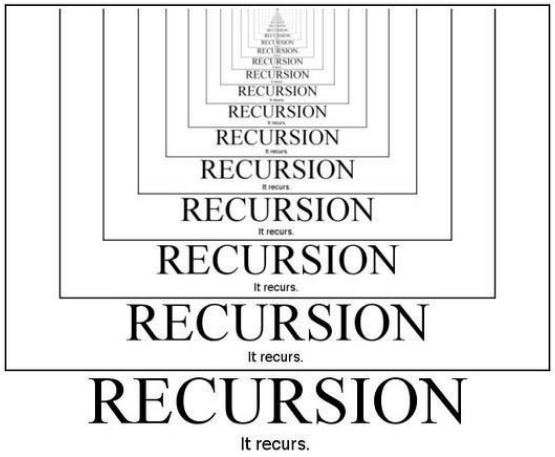


ANOTHER EXAMPLE ON FUNCTION CALL

Russian folk fairy-tale “Repka” (eng. Turnip)

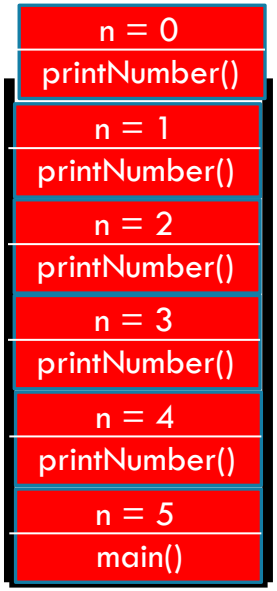
RECURSION OVERVIEW

- Recursion is a programming technique where a function calls itself with some part of the task. And it is the process of repeating items in a self-similar way.
- Way of describing a problem. So it's a way of characterizing a problem independent of how we might implement it.
- Way of designing solutions by by **Divide-and-Conquer**
  - The idea of taking a hard problem, and breaking it up into some smaller, simpler problems, where those smaller problems are easier to solve than the original one and the solutions to the small problems can easily be combined to solve the big problem.



RECURSION EXAMPLE – PRINT THE NUMBERS FROM N TO 1

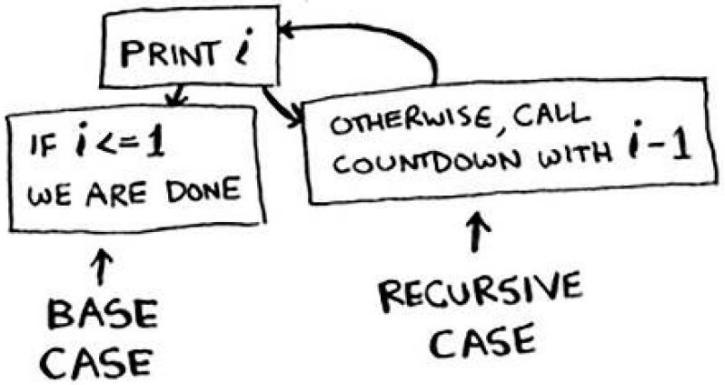
```
public static void printNumber(int n) {

    if(n == 0) return;

    System.out.println(n);
    printNumber(n - 1);
}


public static void main(String[] args) {
    printNumber(5);

}
```



RECURSION OVERVIEW CONTINUES

Recursive solutions involve two major parts:

1. Base case(s), is simple enough to be solved directly.
2. Recursive case(s) . A recursive case has three components:
   a) Divide the problem into one or more simpler or smaller parts of the problems
   b) Invoke the function (recursively) on each part, and
   c) Combine the solutions of the parts into a solution for the problem.



RECURSION – SUM UP

- Recursion is no different than a function call
- Every function call creates a new frame (block) inside the stack
- Recursive function has 2 parts: Base case & Recursive case
- The system keeps track of the sequence of function calls that have been started but not finished yet (active calls)
  - order matters

- Recursion pitfalls
  - miss base-case (infinite recursion, stack overflow)
  - no convergence (solve recursively a problem that is not simpler than the original one)

ASTANA IT UNIVERSITY

TO BE
CONTINUED...