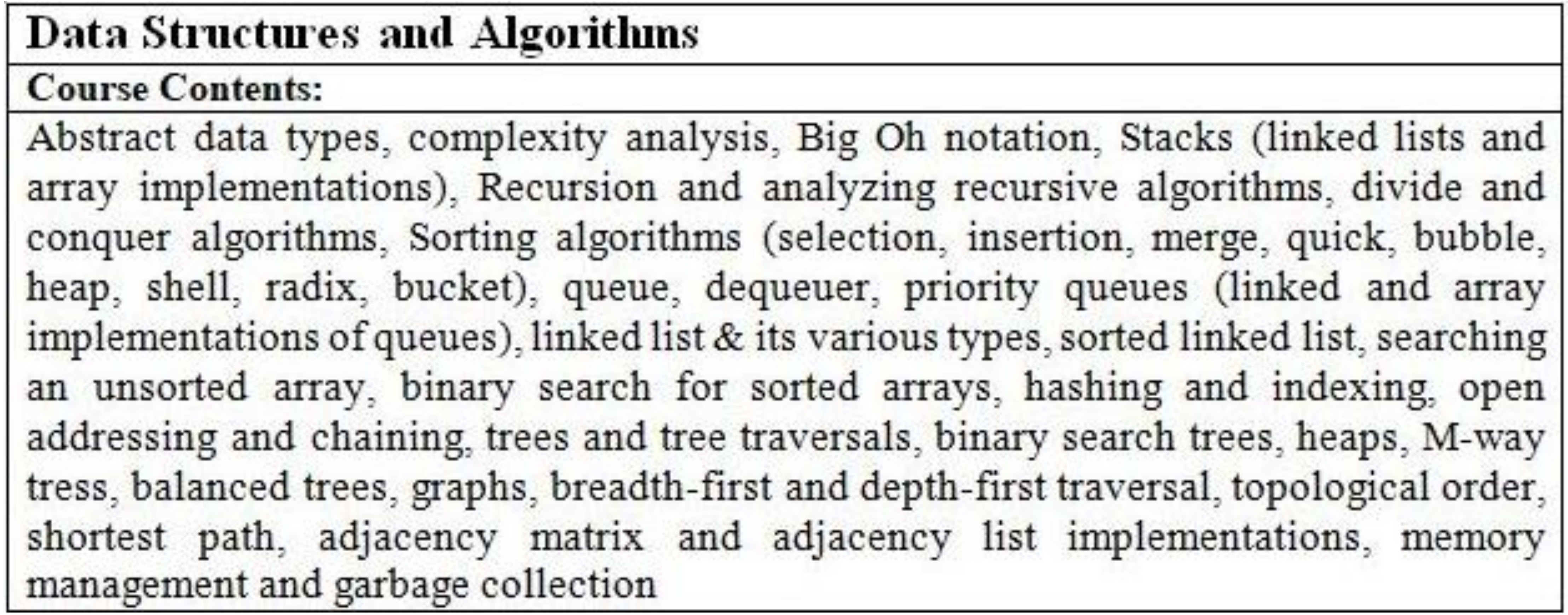
Recursion & Analyzing Recursive Algorithms

Data Structures and Algorithms – CSI 401

Arfan Shahzad

{ arfanskp@gmail.com }



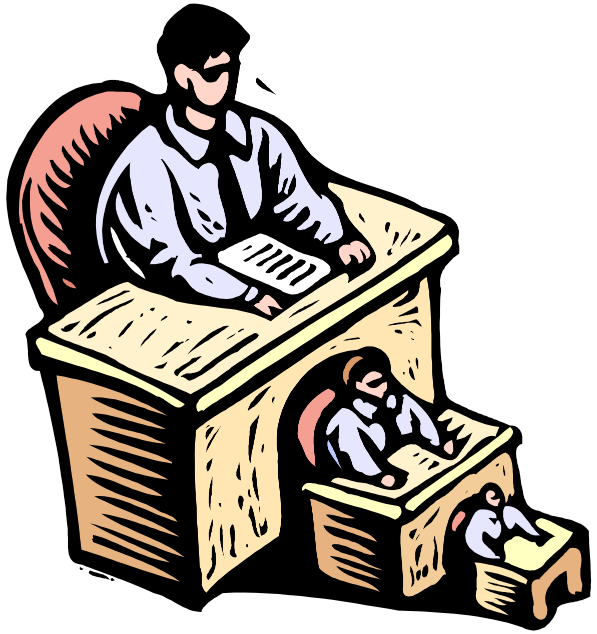
Data Structures and Algorithms
Course Contents:
Abstract data types, complexity analysis, Big Oh notation, Stacks (linked lists and array implementations), Recursion and analyzing recursive algorithms, divide and conquer algorithms, Sorting algorithms (selection, insertion, merge, quick, bubble, heap, shell, radix, bucket), queue, dequeuer, priority queues (linked and array implementations of queues), linked list & its various types, sorted linked list, searching an unsorted array, binary search for sorted arrays, hashing and indexing, open addressing and chaining, trees and tree traversals, binary search trees, heaps, M-way tress, balanced trees, graphs, breadth-first and depth-first traversal, topological order, shortest path, adjacency matrix and adjacency list implementations, memory management and garbage collection

Recursion

- **Recursion** is a problem-solving approach, that splits a problem into one or more **simpler versions of itself**.
- It is a programming technique in which a **function calls itself**.
- **Recursion** occurs when a thing is defined in terms of itself or of its type.

Recursion cont…

- For instance, when the surfaces of two mirrors are exactly parallel with each other, the nested images that occur are a form of infinite recursion.



Recursion cont…
Recursive Thinking: The General Approach

- **if problem is "*small enough*"**
- solve it *directly*
- **else**

1. break into one or more *smaller subproblems*
2. solve each subproblem *recursively*
3. *combine* results into solution to whole problem

Recursion cont…
Requirements for Recursive Solution

- At least one ***"small"*** case that you can solve directly
- A way of ***breaking*** a larger problem down into:

1. One or more *smaller* subproblems
2. Each of the *same kind* as the original
3. A way of ***combining* subproblem** results into an overall solution to the larger problem

Recursion cont…
General Recursive Design Strategy

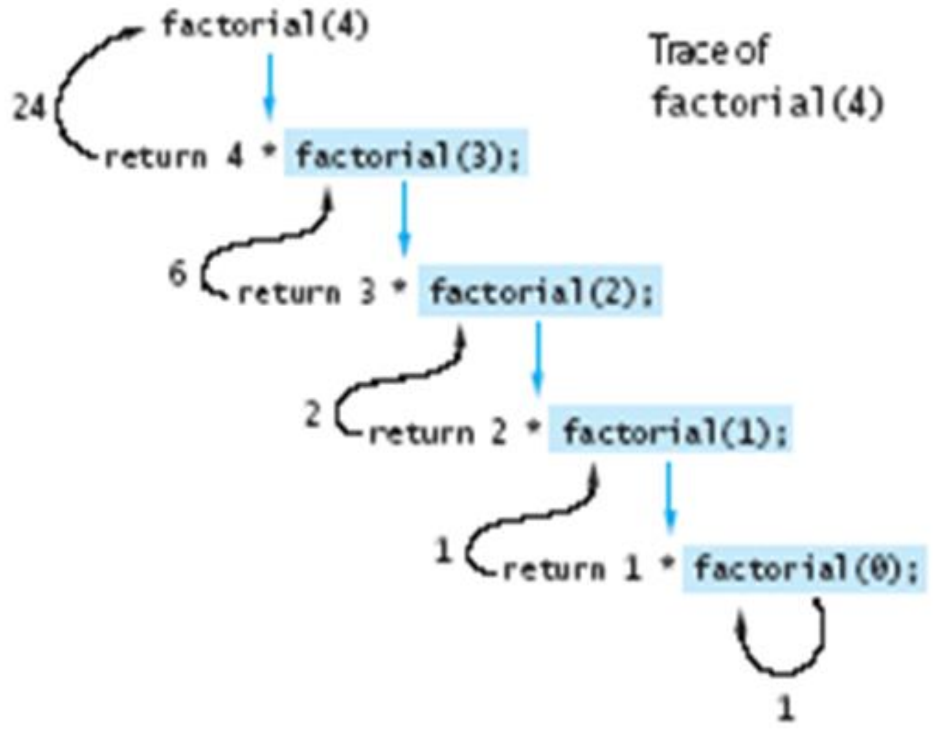
- Identify the ***base case(s)*** (for direct solution)
- Devise a problem ***splitting strategy***
- Subproblems must be smaller
- Subproblems must work towards a base case
- Devise a solution ***combining strategy***

Recursion cont…
Recursion Examples in math

- Mathematicians often faced ***recursive problems***
- These lead naturally to ***recursive algorithms***
- Examples include: Factorial, Powers, Greatest common divisor, etc.

Recursion cont…
Recursive Definitions: Factorial

- If a recursive function never reaches its ***base case***, a stack overflow error occurs
- 0! = 1
- n! = n x (n-1)!



Recursion cont…
Recursive Definitions: Factorial Algorithm

```
• int factorial (int n)  {
•     if (n == 0)  // or: throw exception if < 0
•         return 1;     Base Case
•     else
•         return n * factorial(n-1);
• }
```

Recursion cont…
Recursive Definitions: Power

- $x^0 = 1$ $x^n = x \times x^{n-1}$

```
int power(int x, int n) {
    if (n <= 0)   // or: throw exc. if < 0
        return 1;      //Base case
    else
        return x * power(x, n-1);
}
```

Recursion cont… Recursion Versus Iteration

- Recursion and iteration are somehow ***similar***
- **Iteration:** Loop repetition test determines whether to exit
- **Recursion:** Condition tests for a base case
- We can write iterative solution to a problem solved recursively, ***but:***
  - Recursive code often simpler than iterative
  - Thus easier to write, read, and debug

Recursion cont…
Characteristics of Recursive Methods

- The recursive method calls itself to solve a smaller problem.
- The base case is the smallest problem that the routine solves and the value is returned to the calling method. **(Terminal condition).**
- Calling a method involves certain overhead in transferring the control to the beginning of the method and in storing the information of the return point.

Recursion cont…
Characteristics of Recursive Methods

- Memory is used to store all the intermediate arguments and return values on the internal stack.
- The most important advantage is that it simplifies the problem conceptually.

Recursion cont…
How do I write a recursive function?

- Determine the **size factor**
- Determine the **base case(s)**: the one for which you know the answer
- Determine the **general case(s):** the one where the problem is expressed as a smaller version of itself
- Verify the algorithm: use the "Three-Question-Method"

Recursion cont…
Three-Question Verification Method

1. **The Base-Case Question:** Is there a nonrecursive way out of the function, and does the routine work correctly for this "base" case?
2. **The Smaller-Caller Question:** Does each recursive call to the function involve a smaller case of the original problem, leading inescapably to the base case?
3. **The General-Case Question:** Assuming that the recursive call(s) work correctly, does the whole function work correctly?

Recursion cont…
Deciding whether to use a recursive solution

- When the **depth** of recursive calls is relatively “shallow” then:
- The recursive version does about the **same amount of work** as the nonrecursive version
- The recursive version is **shorter and simpler** than the nonrecursive solution
- But if **depth** of recursive calls is **“deep”** then avoid **recursive solutions** it becomes very slow