

## Introduction

Recursion is one of the **most important and most difficult topics** in lower division computer science courses. We describe our development process to create a concept inventory (CI) that can be used as a pre- and post-test to evaluate the effectiveness of recursion teaching. We have developed a tutorial on recursion as part of the OpenDSA framework. The tutorial provides additional value over and above typical instruction as it currently occurs. As a side benefit, the act of doing practice exercises helps to insure that students spend adequate time learning recursion. The recursion tutorial will be used in CS2 courses, replacing or augmenting current instruction on recursion.

## Concept Inventory

- Addresses Recursion misconceptions.
- Can be used as pre- and post-tests.
- Measures the learning gain as pre-post- test difference.

**Write: Recursive Call**

```

int Question1(int Q1, int Q2) {
    if (Q1 == 0)
        return Q2;
    // Missing code
}
    
```

What should replace "missing code" so that the value of the invocation of Question1(1) returns the value 21?

**Trace: Backward Flow**

```

int Question2(int Q1, int Q2) {
    if (Q1 == 0)
        return Q2;
    return Question2(Q1-1, Q2);
}
    
```

What is the printed value of the invocation Question2(7)? Either give a number, or if it will eventually lead to an infinite recursion just write "infinite recursion", (without the quotes).

**Write: Base Case**

```

int Question3(int Q1, int Q2) {
    if (Q1 == 0)
        return Q2;
    // Missing code
}
    
```

For the following recursive method, give the missing code so that when passed 2 numbers, it will find the sum of all the integers between them inclusive. Example: Given 1 and 4, add 1 + 2 + 3 + 4 and return 10. If the two numbers are equal, then return only one of them and not their summation. Example: Given 2 and 2, return 2.

**Trace: Infinite Recursion**

```

int Question4(int Q1, int Q2) {
    if (Q1 == 0)
        return Q2;
    return Question4(Q1, Q2);
}
    
```

What is the returned value of the invocation Question4(2,3)? Either give a number, or if it will eventually lead to an infinite recursion just write "infinite recursion", (without the quotes).

**Write: A Complete Function**

Write a recursive method `int BinaryToInt(String binary)` that takes a string with a binary number (it only has 1 and 0s) and returns the integer (numeric) representation of the number. Your routine must be recursive.

```

int BinaryToInt(String binary) {
    // Implementation, remember recursive and base cases
}
    
```

For example, the following code `System.out.println(BinaryToInt("1010"))` prints 10.

How to convert from binary to decimal. Each position in a binary number is valued at  $2^{\text{position}}$ . That is, the right most value is  $2^0$ , the next right most is  $2^1$ , etc.

Binary digit	1	0	1	0
Power of 2	$2^3$	$2^2$	$2^1$	$2^0$
Decimal value	8	4	2	1

## Results

We used draft concept inventory as a pre-test Fall 2014 to measure how much students struggle with recursion.

Percentage of students that struggle with each Misconception

Misconception	Percentage
Backward Flow	80%
Limiting Case	63.4%
Recursive Case	47.6%
Forward Flow	44.1%
Confusion with loop structure	35%
Infinite Recursion	16.6%

## Surveys

We gave surveys to CS instructors and students regarding their views on how well students are learning recursion using present instructional methods. The surveys identify differences between required versus actual number of hours spent by students on recursion.

Question	Count/10	Average
Course Level Taught	CS2: 9 CS3: 1	N/A
Required Recursion Background	Nothing Required: 8 Basic Knowledge: 2	2 hours
Time Spent on Recursion in Class	5 to 10 hours: 8 Can not be estimated: 2	2.5 hours
Time Required by the student to spend on Recursion	4 to 7 hours: 2 8 to 27 hours: 8	10 hours
Does students need more time on recursion?	Yes: 10 No: 0	N/A

Instructors Survey Responses

Question	Percentage	Average
Time spent on Recursion	≤ 7 hours: 71.2% > 7 hours: 28.8%	4 hours
Time spent on Coding Bat Exercises	≤ 3 hours: 71% > 3 hours: 29%	2 hours
Confidence Level (on a scale of 5)	≤ 3: 65.3% > 3: 34.7%	2.5

Students Survey Responses

## Recursion Tutorial

The tutorial uses greater interaction with automated feedback. It is an online tutorial designed to address the common misconceptions that the students have in recursion. It differentiates between code writing and code reading/tracing skills.

This tutorial is part of OpenDSA, a collection of open-source tutorials that make extensive use of algorithm visualizations and automatically assessed interactive exercises.

## Recursion Tutorial

The tutorials presents a large number of interactive exercises, allowing students to practice with automatically assessed small-scale programming and non-programming exercises.

## Programming Exercises

- Automatic programming assignment assessment framework
- Randomized tracing exercise instances
- Static code analysis to detect common misconceptions
- Feedback
  - Compilation errors
  - Run time errors
  - Incorrect vs Correct Answer

**Code Completion Exercises**

Recursion Code Completion: Base Case

Given the following recursive function, write down the missing code that should be there at the base case so that the function calculates the value of an array element and return it as string. The value number required will be passed into the other (recursion) call.

```

int Question1(int arr[], int size) {
    if (size == 0)
        return "0";
    // Missing code
}
    
```

**Harder Completion Exercises**

Recursion Harder Code Completion

Given the following recursive function write down the missing code at the line such that the function calculates the value of an array element and return it as string. The value number required will be passed into the other (recursion) call.

```

int Question2(int arr[], int size) {
    if (size == 0)
        return "0";
    if (size == 1)
        return "1";
    // Missing code
}
    
```

**Code Writing Exercises**

Recursion Writing Exercises Set 3

Recursion is a useful recursive definition that relies on the coefficients in the equation of the recursive call. This exercise is designed to help you understand the use of the recursive call in the function. You will be given a recursive function and you will be asked to write the missing code. You will be given a recursive function and you will be asked to write the missing code. You will be given a recursive function and you will be asked to write the missing code.

```

int Question3(int n) {
    if (n == 0)
        return 1;
    if (n == 1)
        return 1;
    // Missing code
}
    
```

**Code Tracing Exercises**

Recursion Tracing: Function return

Consider the following function:

```

int Question4(int n) {
    if (n == 0)
        return 1;
    if (n == 1)
        return 1;
    return Question4(n-1) + Question4(n-2);
}
    
```

What does this function return? Either write a number or write "infinite recursion".

**Summary Exercises**

Recursion Summary Questions

Answer TRUE or FALSE.

A recursive method is always efficient over a non-recursive method.

## Visualizations

- Uses successful mental models from the literature.
- Illustrates the steps of code tracing through the copies model.
- Illustrates the steps of code writing through the delegation model.
- Addresses common recursion misconceptions.

**A tracing of a factorial through copies model**

The theory will multiply the return value of the method by 24. This last copy will return the result of the required factorial.

```

int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial(n-1);
}
    
```

**Multiply x and y through delegation model**

Your friend will do a smaller version of the problem by multiplying x-1 and y. When he returns the result back, you will add a y to that result to complete your task.

```

int multiply(int x, int y) {
    if (x == 0)
        return 0;
    return y + multiply(x-1, y);
}
    
```

**Different Versions of a Recursive Function**

You may even think there's no reason to bother with recursion and prefer to write it as shown in the third version.

```

// First Version
int sum(int arr[], int size) {
    if (size == 0)
        return 0;
    else
        return arr[size-1] + sum(arr, size-1);
}

// Second Version
int sum(int arr[], int size) {
    if (size == 0)
        return 0;
    else
        return arr[size-1] + sum(arr, size-1);
}

// Third Version
int sum(int arr[], int size) {
    if (size == 0)
        return 0;
    else
        return arr[size-1] + sum(arr, size-1);
}
    
```

**Recursive Domino Effect to print 1 to N**

Since the first domino has to be tipped over manually, the solution for the base case, PrintOne(1), is solved nonrecursively by printing 1.

```

void PrintOne(int n) {
    if (n == 1)
        print n / \n;
    else
        PrintOne(n-1); // To print integers from 1 to n-1
    print n / \n;
}
    
```

**Tracing of a Sum Function**

The result returned added to arr[n-1]. The value of arr[n-1] is arr[3-1] = arr[2] = 6. So, add 0 + 6.

```

int sum(int arr[], int n) {
    if (n == 0)
        return 0;
    else
        return arr[n-1] + sum(arr, n-1);
}
    
```

**Full Tracing of Towers of Hanoi**

This process will repeat till we have all the disks moved to support D as we will see in the rest of this tracing visualization.

```

void TowersOfHanoi(int disk, char source, char dest, char spare) {
    if (disk == 0)
        // Move disk from Source to Destination
    else
        TowersOfHanoi(disk-1, source, spare, dest);
    TowersOfHanoi(disk-1, spare, dest, source);
    TowersOfHanoi(disk-1, source, dest, source);
}
    
```