IIT Bombay

Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: Recursive Functions – Part B

Quick Recap of Relevant Topics

IIT Bombay

- Use of simple functions in programs
- Contract-centric view of programming with functions
- Flow of control in function call and return
- Activation records and call stack
- Parameter passing by value and reference
- Recursive functions

Overview of This Lecture

IIT Bombay

- Designing recursive functions
  - Termination and recursive changing of parameters
- Recursion vs iteration

Acknowledgment

IIT Bombay

- Some examples in this lecture are from

  An Introduction to Programming Through C++

  by Abhiram G. Ranade

  McGraw Hill Education 2014

- All such examples indicated in slides with the citation **AGRBook**

Recall: Encoding Example

IIT Bombay

- We want to store quiz 1 and quiz 2 marks of CS101 students in an encoded form
- Encoding strategy: encode(m, n) = $2^m \times 3^n$
- Assume all marks are integers in {1, 2, … 10}

Observe: encode(m, n) = encode(m, n-1) x 3, if m, n > 1
= encode(m-1, 1) x 2, if m > 1, n=1
= 2 x 3 = 6, if m=1, n=1

A Recursive Solution

IIT Bombay

```
#include <iostream>
using namespace std;
int new
int mai
 for ( …
  cipher
 …}
 …
 return 0;
}
```

```
// PRECONDITION: …
int newEnc(int q1Marks,
           int q2Marks)
           (q2Marks) {
           .:
           1Marks == 1) {return 6;}
           {return
            2*newEnc(q1Marks – 1, 1);
      }
      break;
   default: … }
}
// POSTCONDITION: …
```

Are we really sure that every call to newEnc that satisfies precondition eventually terminates?

Caveats Using Recursive Functions

IIT Bombay

- Must specify how to terminate the recursion

  Otherwise, recursion (calling a function from itself) can go on forever

- Must ensure … anges parameters i… entually terminates

Changing parameters in an orderly way to ensure termination

encode(m, n) = encode(m, n-1) x 3, if m, n > 1

= encode(m-1, 1) x 2, if m > 1, n=1

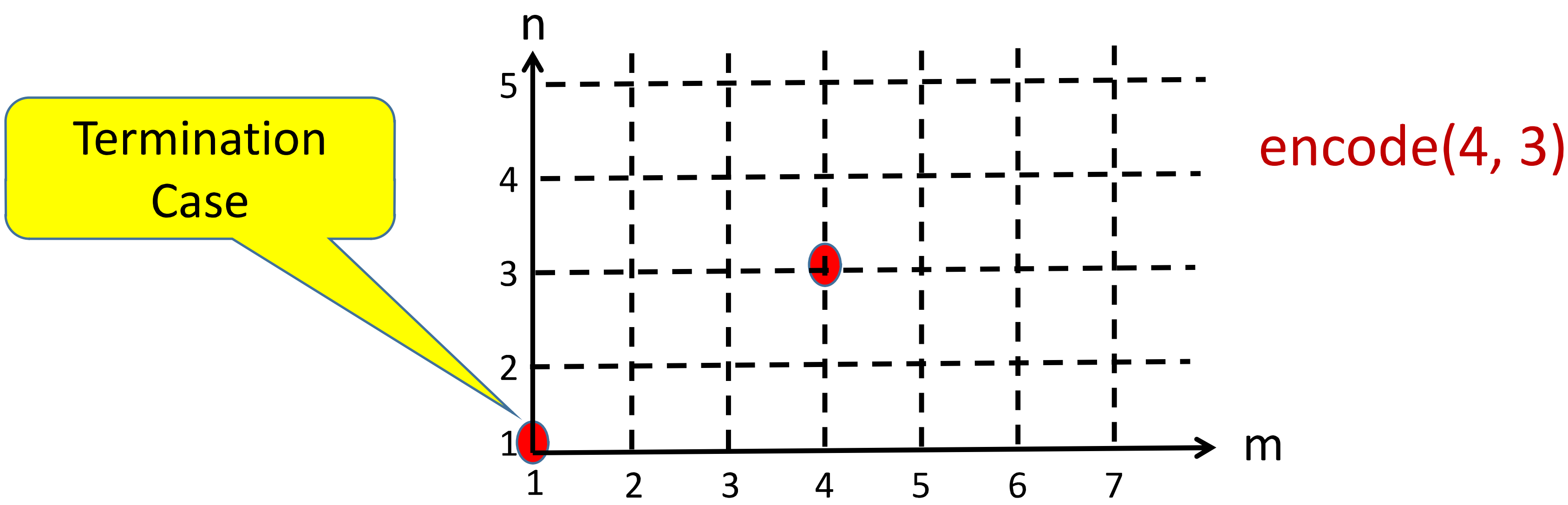
= 2 x 3 = 6, if m=1, n=1

Termination case

Caveats Using Recursive Functions

IIT Bombay

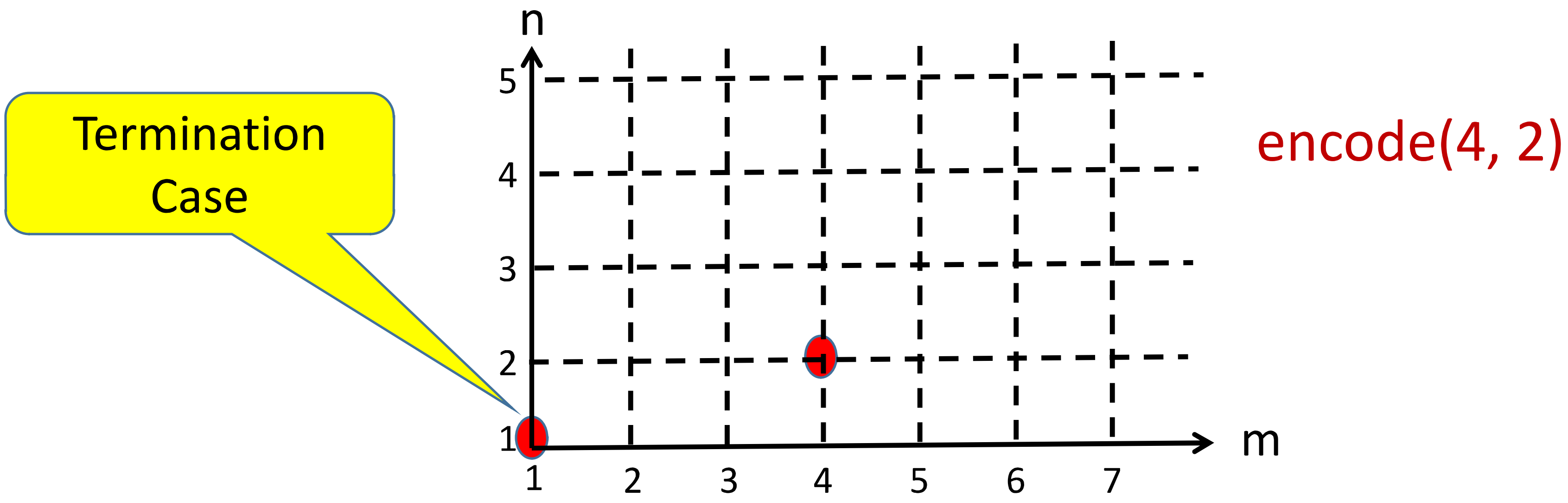
- Think of all possible valuations of parameters as ordered with a fixed end (termination case)
- Recursion must change values of parameters so that we move along this order monotonically towards fixed end



Caveats Using Recursive Functions

IIT Bombay

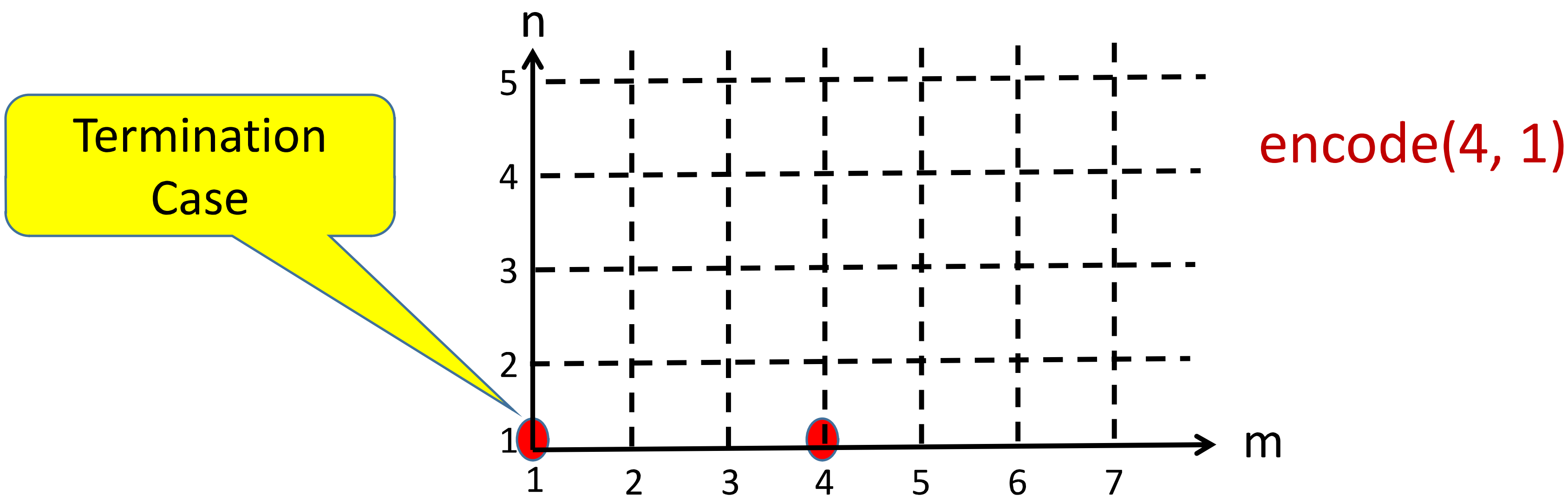
- Think of all possible valuations of parameters as ordered with a fixed end (termination case)
- Recursion must change values of parameters so that we move along this order monotonically towards fixed end



Caveats Using Recursive Functions

IIT Bombay

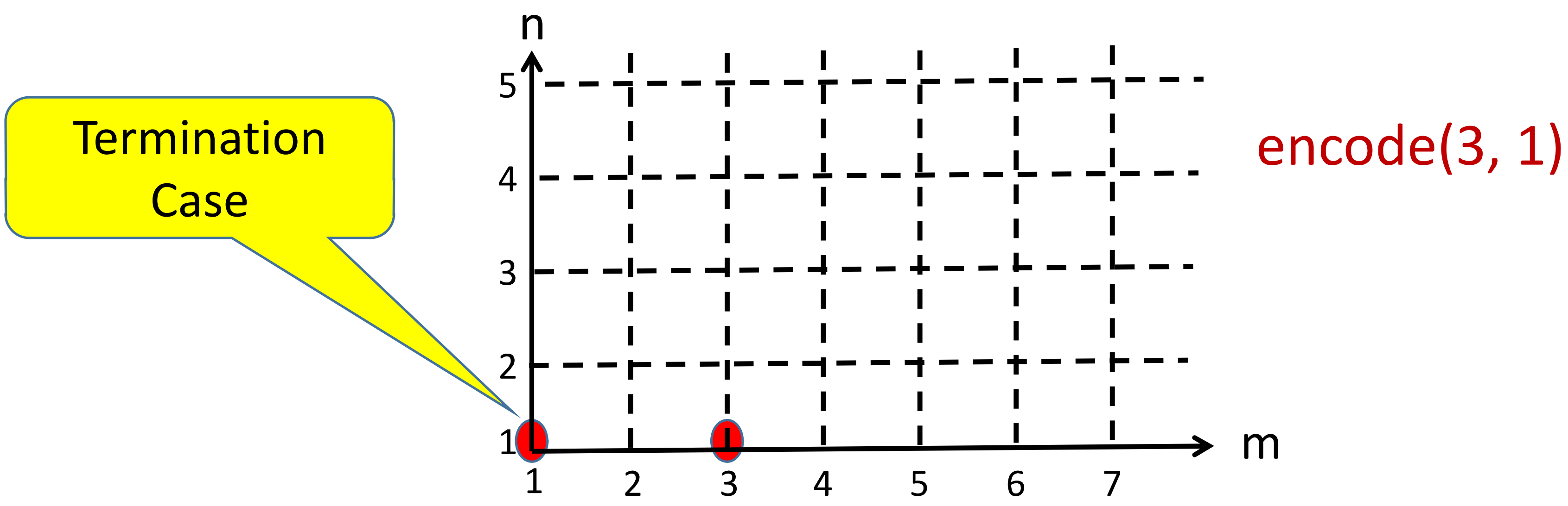
- Think of all possible valuations of parameters as ordered with a fixed end (termination case)
- Recursion must change values of parameters so that we move along this order monotonically towards fixed end



Caveats Using Recursive Functions

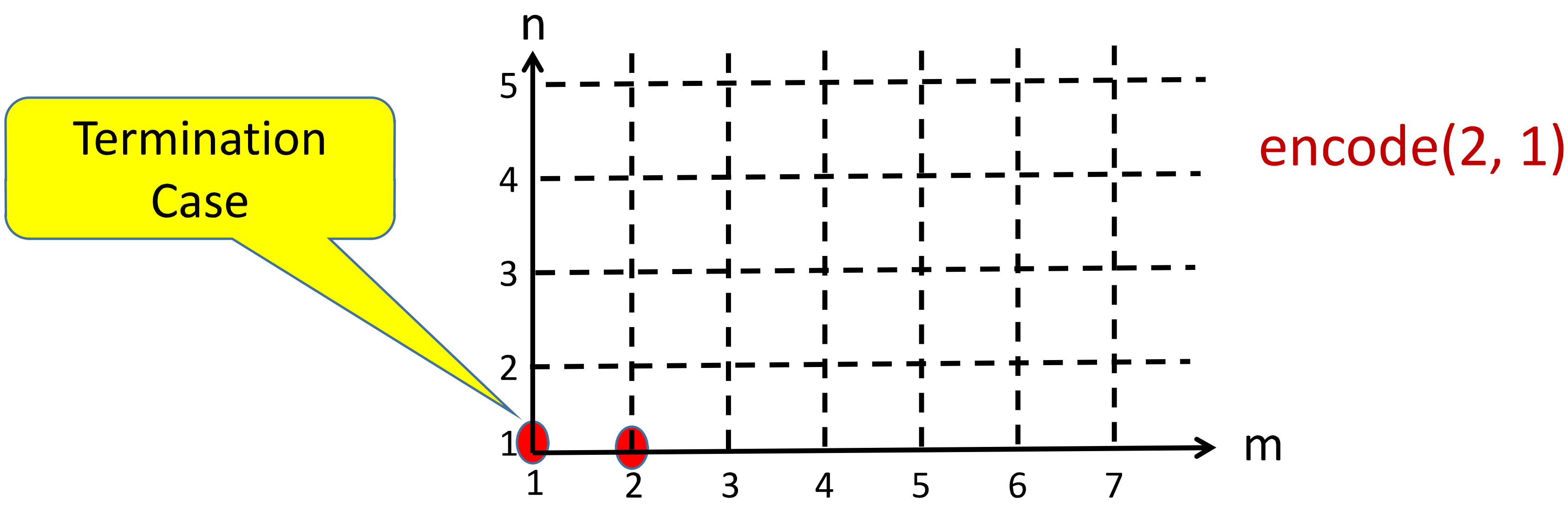
IIT Bombay

- Think of all possible valuations of parameters as ordered with a fixed end (termination case)
- Recursion must change values of parameters so that we move along this order monotonically towards fixed end



Caveats Using Recursive Functions

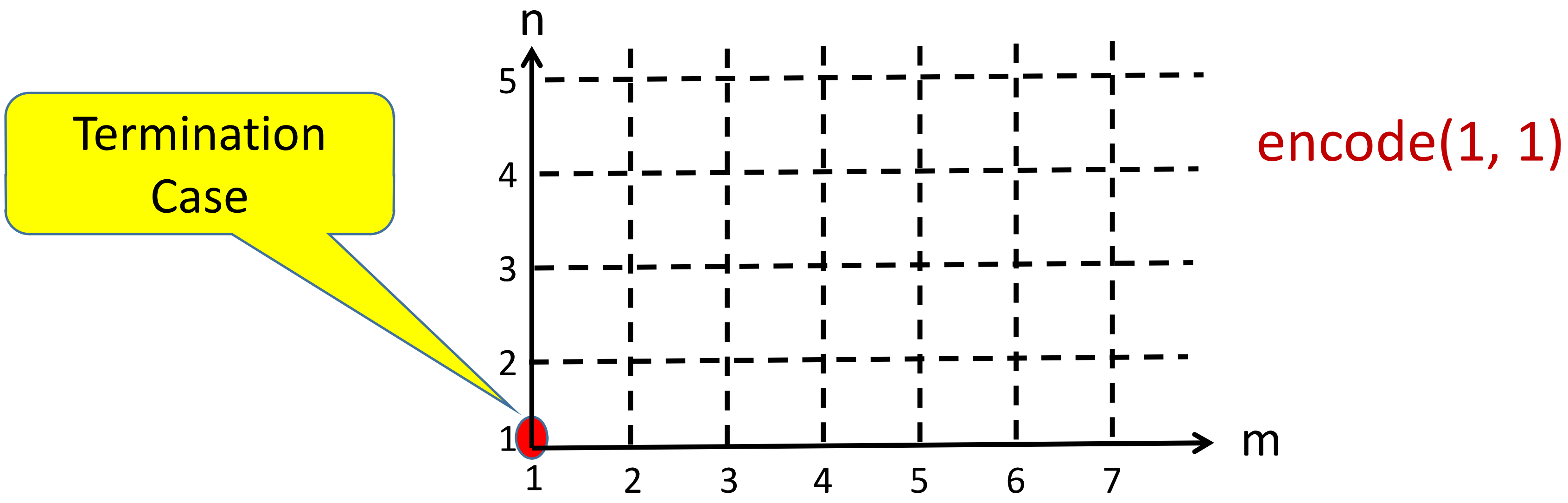
- Think of all possible valuations of parameters as ordered with a fixed end (termination case)
- Recursion must change values of parameters so that we move along this order monotonically towards fixed end



Caveats Using Recursive Functions

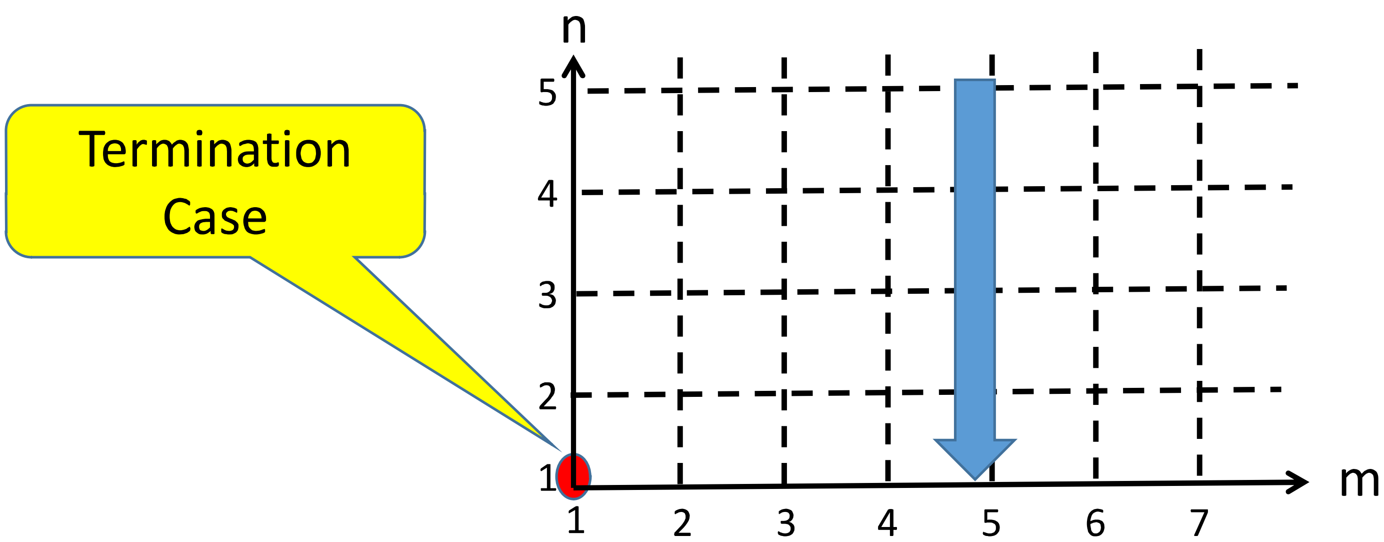
IIT Bombay

- Think of all possible valuations of parameters as ordered with a fixed end (termination case)
- Recursion must change values of parameters so that we move along this order monotonically towards fixed end



Caveats Using Recursive Functions

- Think of all possible valuations of parameters as ordered with a fixed end (termination case)
- Recursion must change values of parameters so that we move along this order monotonically towards fixed end

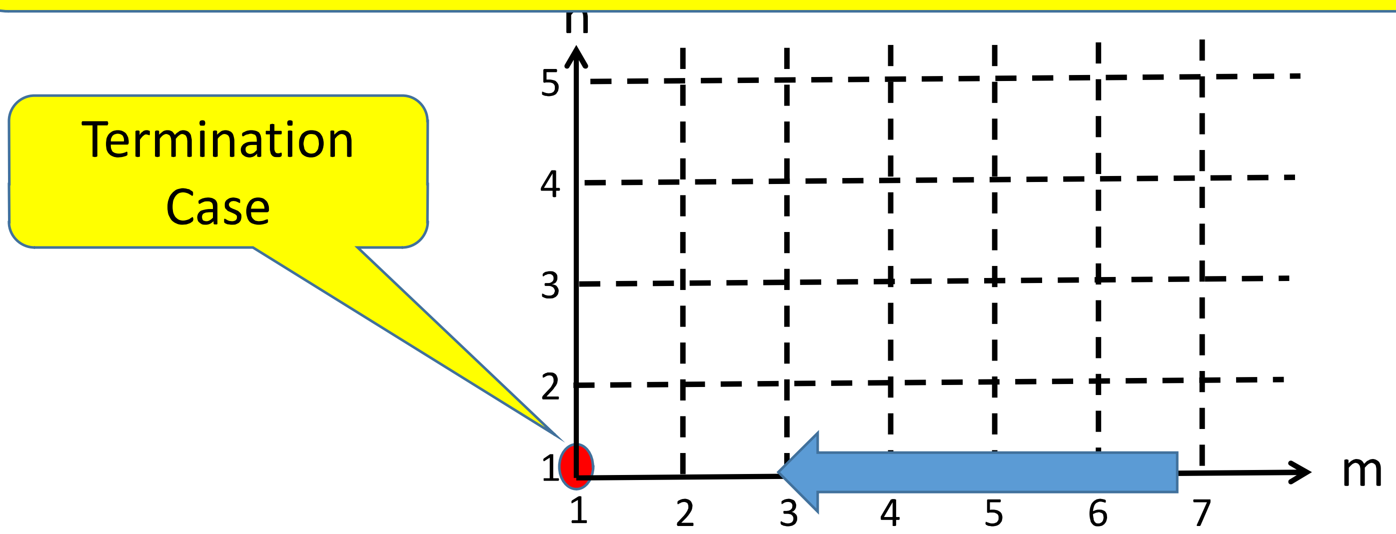


Caveats Using Recursive Functions

IIT Bombay

Well-founded ordering of parameters with a "least" element

Move monotonically along order towards "least" element



Caveats Using Recursive Functions

encode(m, n) = $2^m \times 3^n$ can also be thought as

Changing parameters in this way doesn't ensure termination

encode(m, n) = encode(m, n+1)/3, if m, n > 1

= encode(m+1, 1)/2, if m > 1, n = 1

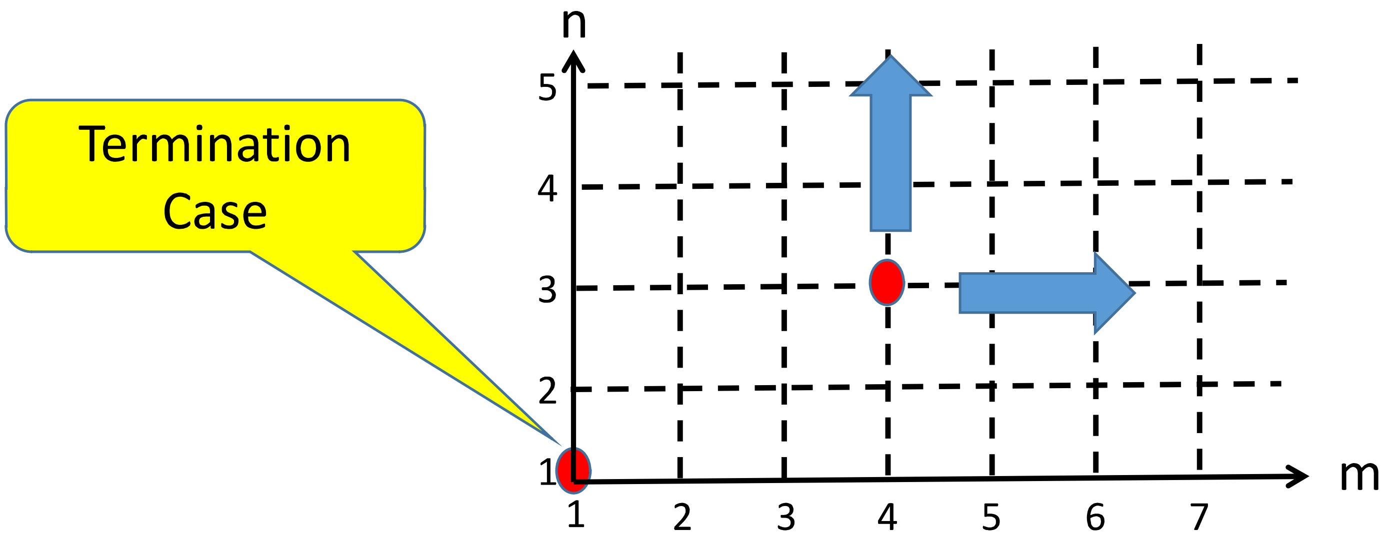
= 2 x 3 = 6, if m = 1, n = 1

Termination case

Caveats Using Recursive Functions

IIT Bombay

- Think of all possible valuations of parameters as ordered with a fixed end (termination case)
- Recursion must change values of parameters so that we move along this order monotonically towards fixed end



A Second Example of Recursion

IIT Bombay

Given n, compute factorial(n) = 1 x 2 x … x n

```
// PRECONDITION:  integer n >= 0
int factorial(int n)
{
  if ( n == 0 ) {return 1;}  // factorial(0) = 1 – Termination case
  else {
    return (n * factorial(n-1));  // Reduce parameter monotonically
                                  //  to 0, and use recursion
  }
 }
// POSTCONDITION: return value = factorial(n)
```

A Third Example [Sec 10.3 of AGRBook]

Virahanka numbers: $V_0 = V_1 = 1$, and $V_n = V_{n-1} + V_{n-2}$ for $n >= 2$

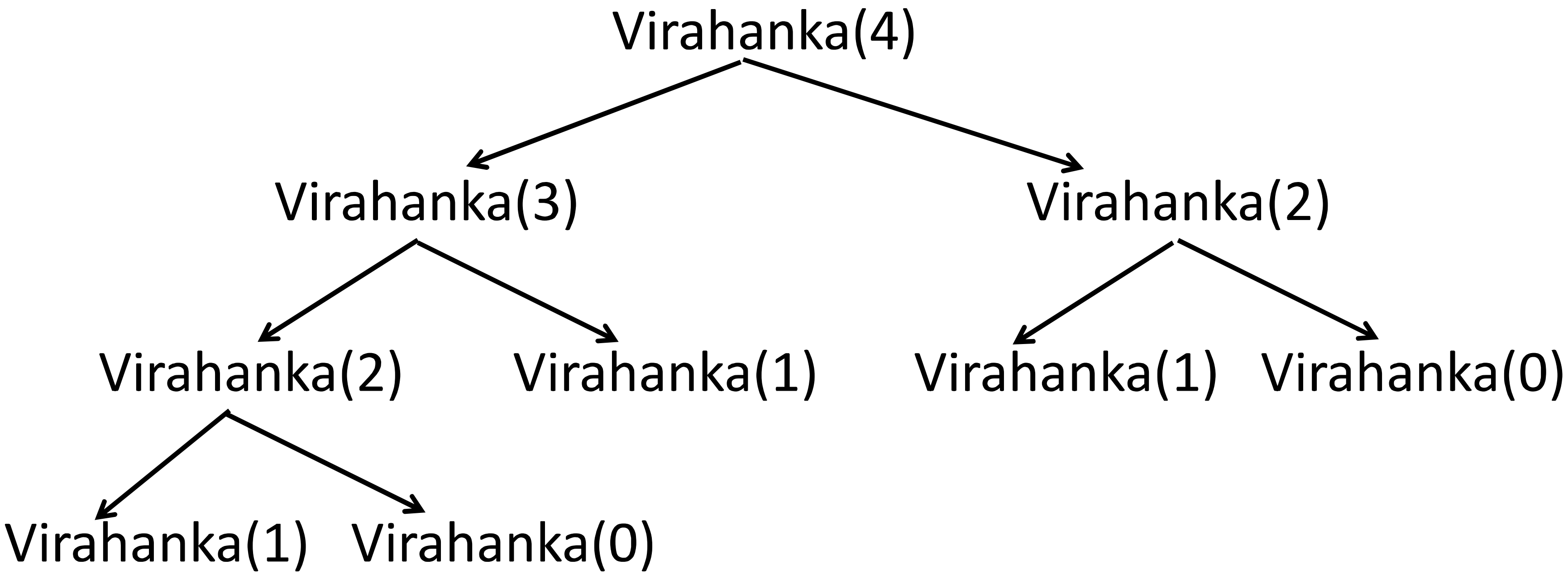
Also known as Fibonacci numbers

(although Virahanka studied these in the context of counting specific types of poetic meters before Fibonacci!)

```
// PRECONDITION: integer n >= 0
int Virahanka(int n)
{ if ((n == 0) || (n == 1)) { return 1; }
  else { return ( Virahanka(n-1) + Virahanka(n-2) ); }
}
// POSTCONDITION: return value = V_n
```

Watch Number of Recursive Calls

IIT Bombay



Number of calls required to compute Virahanka(n) grows exponentially with n

Is There A Better Way?

IIT Bombay

An iterative solution is much better here

```
int Virahanka(int n)
{ int count, result;
  int prevVN = 1, prevPrevVN = 1;
  if ((n == 0) || (n == 1)) { return 1; }
  else {
     for(count = 2; count <= n; count++) {
        result = prevVN + prevPrevVN;
        prevPrevVN = prevVN;  prevVN = result;
      }
   return result; }
}
```

Recursion vs Iteration

- Recursive formulation usually clean, intuitive and succinct

  Need to worry about recursion termination (well-founded ordering of parameter values)

  Need to worry about number of recursive calls
- Iterative formulation may be less clean or intuitive (not always!)

  Need to worry about loop invariants, loop variants and termination

  Can be very efficient if formulated correctly
- Best practice: Judicious mix of iteration and recursion

Summary

IIT Bombay

- Recursive functions
  - Termination and ordering of parameter values
  - Recursion: Monotonically move towards termination case
- Recursion vs iteration