



Northeastern
University

Lecture 23: Recursion - 3

Prof. Chen-Hsiang (Jones) Yu, Ph.D.
College of Engineering

Materials are edited by Prof. Jones Yu from

Data Structures and Abstractions with Java, 5th edition. By Frank M. Carrano and Timothy M. Henry.
ISBN-13 978-0-13-483169-5 © 2019 Pearson Education, Inc.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to
Algorithms, 4th Edition, The MIT Press, 2022. (ISBN: 978-0262046305)

Tail Recursion

Tail Recursion

- In a tail-recursive method, **the last action** is a **recursive call**.
- This call performs a repetition that can be done by using **iteration**.
- Converting a **tail-recursive method** to an **iterative one** is usually a straightforward process. (Well..., sometimes, not always.)

Tail Recursion

```
public static void countDown(int n)
{
    System.out.println(n);
    if (n > 1)
        countDown(n - 1);
} // end countDown
```

When **the last action** performed by a recursive method is **a recursive call**.

Tail Recursion

- Converting a recursive method to an iterative one

```
public static void countDown(int integer)
{
    if (integer >= 1)
    {
        System.out.println(integer);
        countDown(integer - 1);
    } // end if
} // end countDown
```

- An interactive version

```
public static void countDown(int integer)
{
    while (integer >= 1)
    {
        System.out.println(integer);
        integer = integer - 1;
    } // end while
} // end countDown
```



Using a Stack Instead of Recursion

Using a Stack Instead of Recursion

```
public static void displayArray(int array[], int first, int last)
{
    if (first == last)
        System.out.print(array[first] + " ");
    else
    {
        int mid = first + (last - first) / 2;
        displayArray(array, first, mid);
        displayArray(array, mid + 1, last);
    } // end if
} // end displayArray
```

Recursive approach

Using a Stack Instead of Recursion

```
public class Record
{
    private int first, last;

    private Record(int firstIndex, int lastIndex)
    {
        first = firstIndex;
        last = lastIndex;
    } // end constructor
} // end Record
```

- An **iterative** displayArray to maintain its own stack

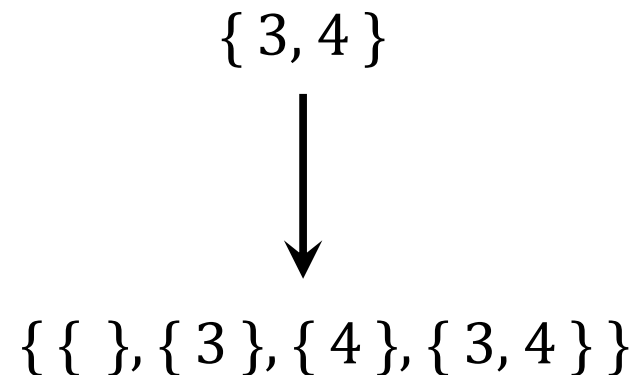
```
public void displayArray(int first, int last)
{
    boolean done = false;
    StackInterface<Record> programStack = new LinkedStack<>();
    programStack.push(new Record(first, last));

    while (!done && !programStack.isEmpty())
    {
        Record topRecord = programStack.pop();
        first = topRecord.first;
        last = topRecord.last;

        if (first == last){
            System.out.println(array[first] + " ");
            done = true;
        }else{
            int mid = first + (last - first) / 2;
            // Note the order of the records pushed onto the stack
            programStack.push(new Record(mid + 1, last));
            programStack.push(new Record(first, mid));
        } // end if
    } // end while
} // end displayArray
```

Exercise

- Write a method to return all subsets of a set



i.e. **the power set** of $\{ 3, 4 \}$ is $\{ \{ \}, \{ 3 \}, \{ 4 \}, \{ 3, 4 \} \}$

Answer

```
public class Subsets {
    private static ArrayList<ArrayList<Integer>> getSubsets(ArrayList<Integer> set, int index){

        ArrayList<ArrayList<Integer>> allsubsets;

        if(set.size() == index){
            allsubsets = new ArrayList<ArrayList<Integer>>();
            allsubsets.add(new ArrayList<Integer>());
        }else{
            allsubsets = getSubsets(set, index+1);
            int item = set.get(index);

            ArrayList<ArrayList<Integer>> moresubsets = new ArrayList<ArrayList<Integer>>();
            for(ArrayList<Integer> subset: allsubsets){
                ArrayList<Integer> newsubset = new ArrayList<Integer>();
                newsubset.addAll(subset);
                newsubset.add(item);
                moresubsets.add(newsubset);
            }
            allsubsets.addAll(moresubsets);
        }
        return allsubsets;
    }

    public static void main(String[] args){
        ArrayList<Integer> data = new ArrayList<Integer>();
        for(int i = 0; i < 5; i++){
            data.add(i+1);
        }
        ArrayList<ArrayList<Integer>> result = getSubsets(data, 0);
        System.out.println(result);
    }
}
```

Dynamic Programming

Definition

- **Dynamic programming** was developed by Richard Bellman in the 1950s. It is a **math optimization method** and **an algorithmic paradigm**.
- **Dynamic programming**, like the **divide-and-conquer method**, solves problems by combining solutions to subproblems.
- **Programming** in this context refers to **a tabular method**, not to writing computer code.

Definition (cont.)

- A **dynamic-programming algorithm** solves each subproblem just once and **saves the answer in a table**.
- It avoids **the work of recomputing the answer** every time when it solves each subproblem.
- Dynamic programming is an extension to the **divide-and-conquer method**. When you recursively divide the original problem into a smaller problem, you can apply it to save the answer.

Example: Fibonacci Numbers

- The Fibonacci series begins with 0 and 1.
- Each subsequent number is the sum of the previous two.

The series: 0 1 1 2 3 5 8 13 21 34 55 89 ...

Indexes: 0 1 2 3 4 5 6 7 8 9 10 11

- The series can be recursively defined as:

» $\text{fib}(0) = 0;$

$\text{fib}(1) = 1;$

$\text{fib}(\text{index}) = \text{fib}(\text{index}-2) + \text{fib}(\text{index}-1) ; \text{index} \geq 2$

Fibonacci Numbers

$\text{fib}(0) = 0;$

$\text{fib}(1) = 1;$

$$\text{fib}(4) = \text{fib}(3) + \text{fib}(2)$$

$$= (\text{fib}(2) + \text{fib}(1)) + \text{fib}(2)$$

$$= ((\text{fib}(1) + \text{fib}(0)) + \text{fib}(1)) + \text{fib}(1) + \text{fib}(0)$$

$$= ((1 + \text{fib}(0)) + \text{fib}(1)) + \text{fib}(1) + \text{fib}(0)$$

$$= \dots$$

$$= 1 + 0 + 1 + 1 + 0$$

$$= 3$$

Fibonacci Numbers (cont.) - recursion approach

```
public class ComputeFibonacci {
    /** Main method */
    public static void main(String[] args) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);
        System.out.print("Enter an index for a Fibonacci number: ");
        int index = input.nextInt();

        // Find and display the Fibonacci number
        System.out.println("The Fibonacci number at index " + index + " is " + fib(index));
        input.close();
    }

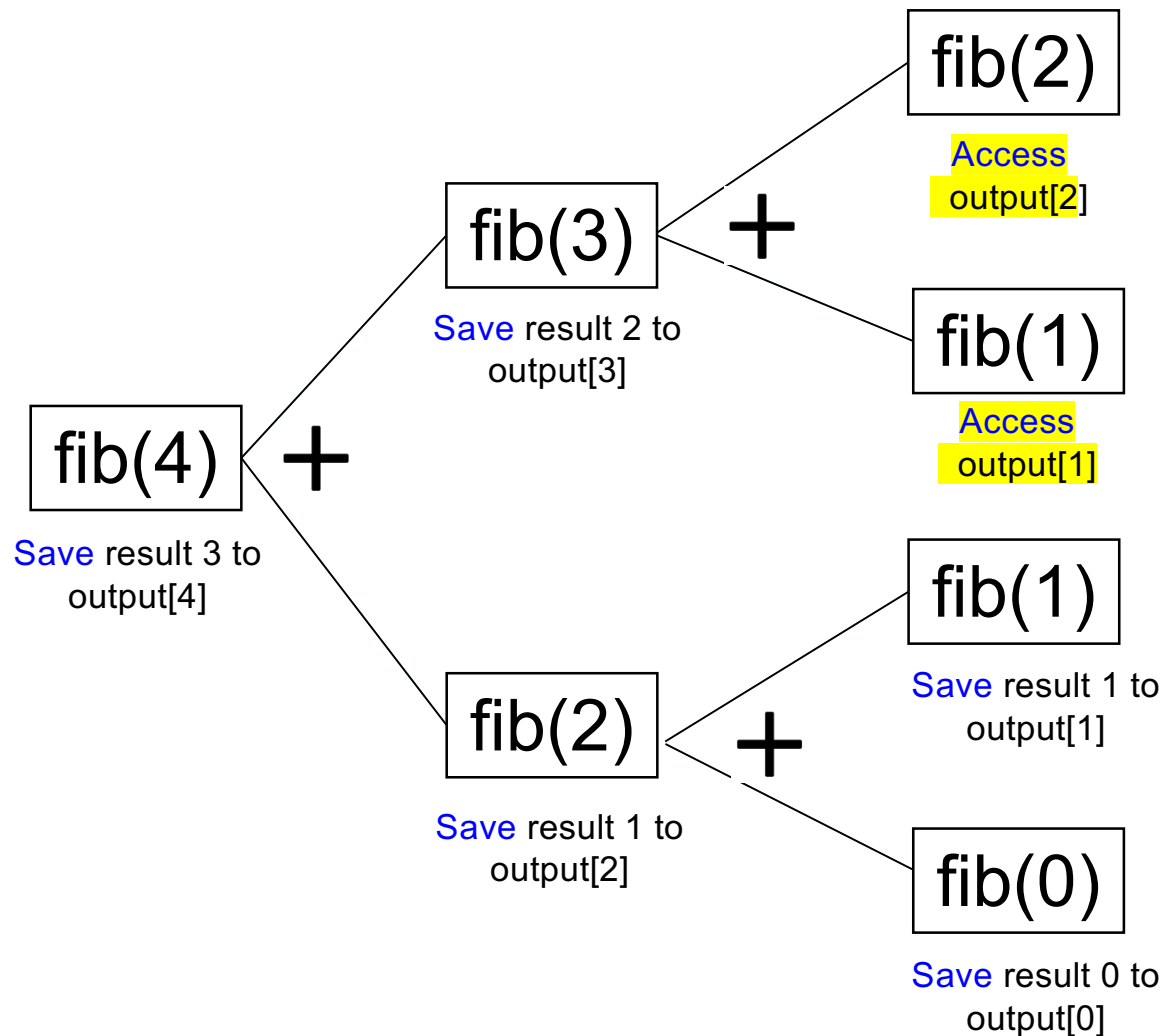
    /** The method for finding the Fibonacci number */
    public static int fib(int index) {
        if (index == 0) // Base case
            return 0;
        else if (index == 1) // Base case
            return 1;
        else // Reduction and recursive calls
            return fib(index - 1) + fib(index - 2);
    }
}
```

```
Enter an index for a Fibonacci number: 6
The Fibonacci number at index 6 is 8
```

Fibonacci Numbers (cont.)

fib(0) = 0;

fib(1) = 1;



Exercise

- Please update `fib(int index)` method such that you can apply dynamic programming skills to calculate Fibonacci number.

```
/** The method for finding the Fibonacci number */  
public static long fib(int index) {  
    if (index == 0) // Base case  
        return 0;  
    else if (index == 1) // Base case  
        return 1;  
    else // Reduction and recursive calls  
        return fib(index - 1) + fib(index - 2);  
}
```

```
Enter an index for a Fibonacci number: 6  
The Fibonacci number at index 6 is 8
```

Answer

```
public class ComputeFibonacci {
    /** Main method */
    public static void main(String[] args) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);
        System.out.print("Enter an index for a Fibonacci number: ");
        int index = input.nextInt();

        // Find and display the Fibonacci number
        System.out.println("The Fibonacci number at index " + index + " is " + fib(index));
        input.close();
    }

    public static int[] output = new int[1000];

    /** The method for finding the Fibonacci number */
    public static int fib(int index) {
        int result = output[index];
        if (result == 0) {
            if (index == 0) // Base case
                return 0;
            else if (index == 1) // Base case
                return 1;
            else // Reduction and recursive calls
                return fib(index - 1) + fib(index - 2);
        }
        output[index] = result;
        return result;
    }
}
```