Section IV.1: Recursive Algorithms and Recursion Trees

Definition IV.1.1: A recursive algorithm is an algorithm that solves a problem by (1) reducing it to an instance of the same problem with smaller input and (2) having a part (for the smallest instances) where the solution is computed directly without the algorithm making any calls to itself. Thus, a recursive algorithm is made of two parts – a base part and a recursive part. The base part is where the solution is computed directly for smallest inputs. The recursive part is where the solution is computed by making one or more calls to the algorithm itself using smaller inputs.

That is, a recursive algorithm calls itself using a smaller version of the original input. Recursion is similar to mathematical induction. In mathematical induction, a statement can be proved to be true for a value by using the assumption that it is true for smaller values. In mathematical induction, we also must have a basis that can be proven directly.

Definition IV.1.2: A recursion tree is a tree that is generated by tracing the execution of a recursive algorithm. A recursion tree shows the relationship between calls to the algorithm. Each item in the tree represents a call to the algorithm.

Developing a Recursive Algorithm

A recursive algorithm can be developed in two main steps. Step 1 is developing the base part and Step 2 is developing the recursive part.

Step 1: Developing the base part.

When developing the base part, define the solution for the smallest inputs. Write a statement giving the solution for the smallest input. This statement should be in the following form: if the input is the smallest, then here is the solution. The base part may consist of one or more statements. In many cases, defining the solution for the smallest input is adequate, but at times it may be appropriate to define the solution for the smallest two inputs, or smallest three, etc.

Step 2: Developing the recursive part.

To develop the recursive part, determine a way to define the solution for values greater than the smallest by using the solution for smaller inputs. The statement for this part should include calling the algorithm with smaller inputs.

Example IV.1.1: This example finds the sum of $N$ integers stored in an array. Solution (a) does not use recursion. Solution (b) uses a recursive algorithm. Solution (c) shows the recursion tree for tracing Solution (b).
Solution: (a) - Iterative Solution

Algorithm Array Sum
Output: sum of the $N$ integers in array $A$

Algorithm Body:

```
\begin{align*}
j &:= 1 \\
sum &:= 0 \\
\text{while } j < N &
\begin{align*}
sum &:= sum + A[j] \\
j &:= j + 1
\end{align*}
\end{align*}
```

end Algorithm Array Sum

Solution: (b) Recursive Solution using the steps in developing a recursive algorithm

Step 1: Developing the base part
The smallest input is where $N=0$. Thus, the output is zero, since the sum of an empty array is zero. In other words, if $N$ is zero, then the sum is zero.

Step 2: Developing the recursive part
For $N$ greater than zero, the sum of $N$ integers is the $N^{th}$ integer plus the sum of the previous $N-1$ integers. Notice the sum with input $N$ is based on finding the sum with input $N-1$.

Algorithm Array Sum
Output: sum of the $N$ integers in array $A$

Algorithm Body:

```
\begin{align*}
\text{if } N=0 &\text{ then sum := 0 [using Step 1]} \\
\text{else sum := } A[N] + \text{ Array Sum of integers } A[1], A[2], \ldots, A[N-1] &\text{ [using Step 2]}
\end{align*}
```

end Algorithm Array Sum

```
7 + 15 = 22, return 22 [solution]
```

```
3 + 12 = 15, return 15
```

```
6 + 6 = 12, return 12
```

```
4 + 2 = 6, return 6
```

```
2 + 0 = 2, return 2
```

```
Array Sum with inputs 0 and [] is 0
```

The arrows going down show the order that calls are made to **Array Sum**. To add an array of five numbers, six calls are made to **Array Sum**. The last call is the one that satisfies the base part of the recursive algorithm. Once the base condition is satisfied (that is, \( N \) is zero), the algorithm starts returning values to be added. The arrows going up show the returns from calls to the algorithm. Zero is the first value that is returned, and it is added to 2. Then the sum 2 is returned and added to 4, the sum 6 is returned and added to 6, the sum 12 is returned and added to 3, the sum 15 is returned and added to 7, and, finally, the sum 22 is returned.

**Example IV.1.2:** Compute the \( N^{th} \) number in the Fibonacci series of numbers. (See Epp, page 431 for an explanation of Fibonacci numbers.) Solution (a) shows the recursive algorithm for this problem. Solution (b) show the recursion tree tracing the execution of Solution (a).

**Solution: (a)** Recursive solution using the steps in developing a recursive algorithm

**Step 1: Developing the base part.** The solutions for the smallest values are as follows. If \( N \) is zero, then the \( N^{th} \) fibonacci number is one. If \( N \) is one, then the \( N^{th} \) fibonacci number is one.
Step 2: Developing the recursive part.
For $N$ greater than one, the $N^{\text{th}}$ Fibonacci number is the $(N-1)^{\text{st}}$ Fibonacci number plus the $(N-2)^{\text{nd}}$ Fibonacci number.

Algorithm Fibonacci
 Input: non-negative integer $N$ specifying the Fibonacci number that is to be computed.
 Output: $f$, the $N^{\text{th}}$ Fibonacci number.
 Algorithm Body:
    if $N=0$ or $N=1$ then $f := 1$
    else $f := \text{Fibonacci}$ with input $N-1 + \text{Fibonacci}$ with input $N-2$
 end Algorithm Fibonacci

Solution: (b) Recursion tree using algorithm Fibonacci with $N = 4$.

Each unshaded box shows a call to the algorithm Fibonacci with the input value of $N$ in parentheses. Each shaded box shows the value that is returned from the call. Calls to algorithm Fibonacci are made until a call is made with input value one or zero. When a call is made with input value one or zero, a one is returned. When a call is made with $N > 1$, two calls are made to algorithm Fibonacci, and the value that is returned is the sum of the values from the two calls. The final number that is returned is 5. Thus the $4^{\text{th}}$ number in the Fibonacci series is 5.
Exercises:

(1) Develop a recursive algorithm to search an array of unordered integers for a specified value. The algorithm should return an array index value specifying the location of the value, if found, or return zero, if not found. The input values are $N$ (the number of items in the array) and the array $A[1], A[2], \ldots, A[N]$. Show the recursion trees using $N=5$ and $A= [12, 4, 6, 13, 14]$ when searching for 6, and when searching for 5.

(2) Show the recursion tree for algorithm Fibonacci when $N = 6$. What is the $6^{th}$ number in the Fibonacci series?