**Chapter 5: Recursion**

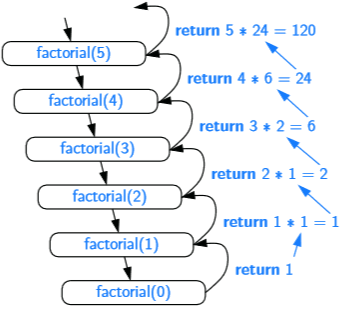
**Recursion**

* Recursion is a technique by which a method makes one or more calls to itself during execution, or by which a data structure relies upon smaller instances of the very same type of structure in its representation.
* When one invocation of the method makes a recursive call, that invocation is suspended until the recursive call completes.
* We will do the implementation of:
  + **Factorial Function**
  + **Sum of Squares**
  + **Reverse A String**
  + **Towers of Hanoi**
  + **English Ruler**
  + **Binary Search**
  + **File System**

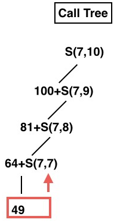
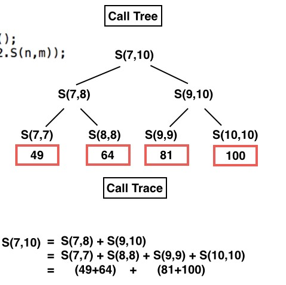
**Divide and Conquer**

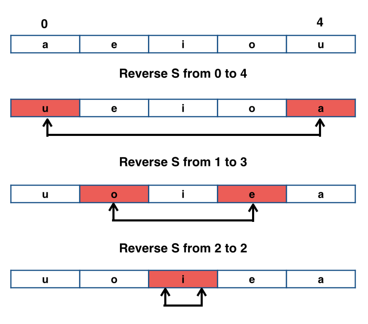
* Algorithmic design pattern
* This pattern consists of the following three steps:
  + **Divide**: If the input size is smaller than a certain threshold (say, one or two elements), solve the problem directly using a straightforward method and return the solution so obtained. Otherwise, divide the input data into two or more disjoint subsets.
  + **Conquer**: Recursively solve the subproblems associated with the subsets.
  + **Combine**: Take the solutions to the subproblems and merge them into a solution to the original problem.

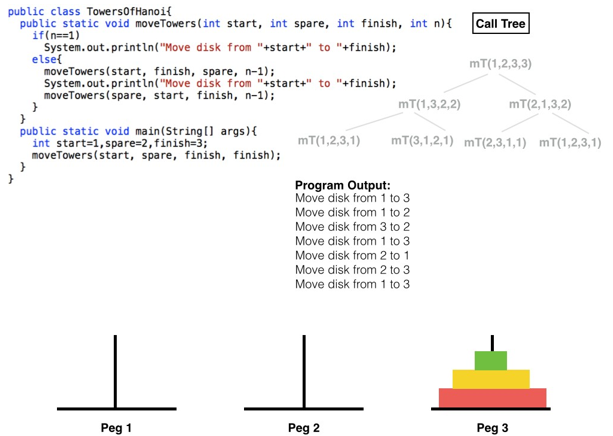
**The Factorial Function**

* The factorial function is important because it is known to equal the number of ways in which **n** distinct items can be arranged into a sequence, that is, the number of **permutations** of n items.
* First, we have one or more base cases, which refer to ﬁxed values of the function. The above deﬁnition has one **base case** stating that **n! = 1** for **n = 0**. Second, we have one or more **recursive** **cases**, which deﬁne the function in terms of itself. In the above deﬁnition, there is one recursive case, which indicates that **n! = n·(n−1)!** for **n ≥ 1**.
* Here is the recursion trace for the call factorial(5)
* In Java, each time a method (recursive or otherwise) is called, a structure known as an **activation record** or **activation frame** is created to store information about the progress of that invocation of the method. This frame stores the parameters and local variables speciﬁc to a given call of the method, and information about which command in the body of the method is currently executing.

**Sum of Squares**

* We find the sum of squares within a given range.
* Establish the base case that if **n** and **m** are the same then we return the square value of one of them.
* Here is the equation for the first implementation. As an example, here is the call tree for the range 7 to 10.
* In the second implementation we make the problem even simpler by dividing the work in two, here is the call tree for range 7 to 10.

**Reverse A String**

**Towers of Hanoi**

**English Ruler**