# RECURRENCE

# Recurrence relation

- In mathematics, a recurrence relation is an equation that recursively defines a sequence.

- We are going to use Recurrence Relations to solve for the run-time of a recursive algorithm.

- Recurrence relations will be the mathematical tool that allows us to analyze recursive algorithms.

# Recurrence

- A  recurrence relation is an equation which is defined in terms of itself.
- The recurrence equations arise very naturally to express the resources used by recursive procedures.
- A **recurrence relation** expresses the running time of a recursive algorithm.
- This includes
  - how many recursive calls are generated at each level of the recursion,
  - how much of the problem is solved by each recursive call, and
  - how much work is done at each level

# Recursion

- **What is Recursion?**

- A problem-solving strategy that solves large problems by reducing them to smaller problems of the same form.

# Recursive Algorithm

- Recall that a recursive or inductive definition has the following parts:

1. **Base Case**: the initial condition or basis which defines the first (or first few) elements of the sequence

2. **Inductive (Recursive) Case**: an inductive step in which later terms in the sequence are defined in terms of earlier terms.

# Recursion Review

- An example is the recursive algorithm for finding the factorial of an input number n.

  Where n=4!

  = 4*3*2*1 = 24

- Note that each factorial is related to the factorial of the next smaller integer:

  n! = n * (n-1)!

  So, 4! = 4 * (3-1)! = 4 * 3!

  We stop at 1! = 1

- In mathematics, we would define:

  n! = n * (n-1)!    if n > 1

  n! = 1    if n = 1

# Recursion Review

- The recursive algorithm for finding factorial of input number n
- Where n =4
- 4! =4 *3! =4*3*2*1

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

| | | |
|---|---|---|
| factorial(1) : | return 1; | **1** |
| factorial(2) : | return 2 * factorial(1); | **2 * 1 = 2** |
| factorial(3) : | return 3 * factorial(2); | **3 * 2 = 6** |
| factorial(4) : | return 4 * factorial(3); | **4 * 6 = 24** |

# Recurrence Relation

- Let’s determine the run-time of factorial,
  - Using Recurrence Relations

- We can see that the total number of operations factorial for input size n

1. The sum of the 2 operations (the ‘*’ and the ‘-’)
2. Plus the number of operations needed to execute the function for n-1.
3. OR if it’s the base case just one operation to return.

# Recurrences and Running Time

- An equation or inequality that describes a function in terms of its value on smaller inputs.

$$T(n) = T(n-1) + n$$

- Recurrences arise when an algorithm contains recursive calls to itself
- What is the actual running time of the algorithm?
- Need to solve the recurrence
  - Find an explicit formula of the expression
  - Bound the recurrence by an expression that involves n

# Example Recurrences

- $T(n) = T(n-1) + n$ $\Theta(n^2)$
  - Recursive algorithm that loops through the input to eliminate one item
- $T(n) = T(n/2) + c$ $\Theta(\lg n)$
  - Recursive algorithm that halves the input in one step
- $T(n) = T(n/2) + n$ $\Theta(n)$
  - Recursive algorithm that halves the input but must examine every item in the input
- $T(n) = 2T(n/2) + 1$ $\Theta(n)$
  - Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

# Recurrence Relation

- **Solution Methods**
  - Master Method.
  - Recursion-tree Method.
  - Substitution Method.