Recurrence Relations – Part 1

Introduction:

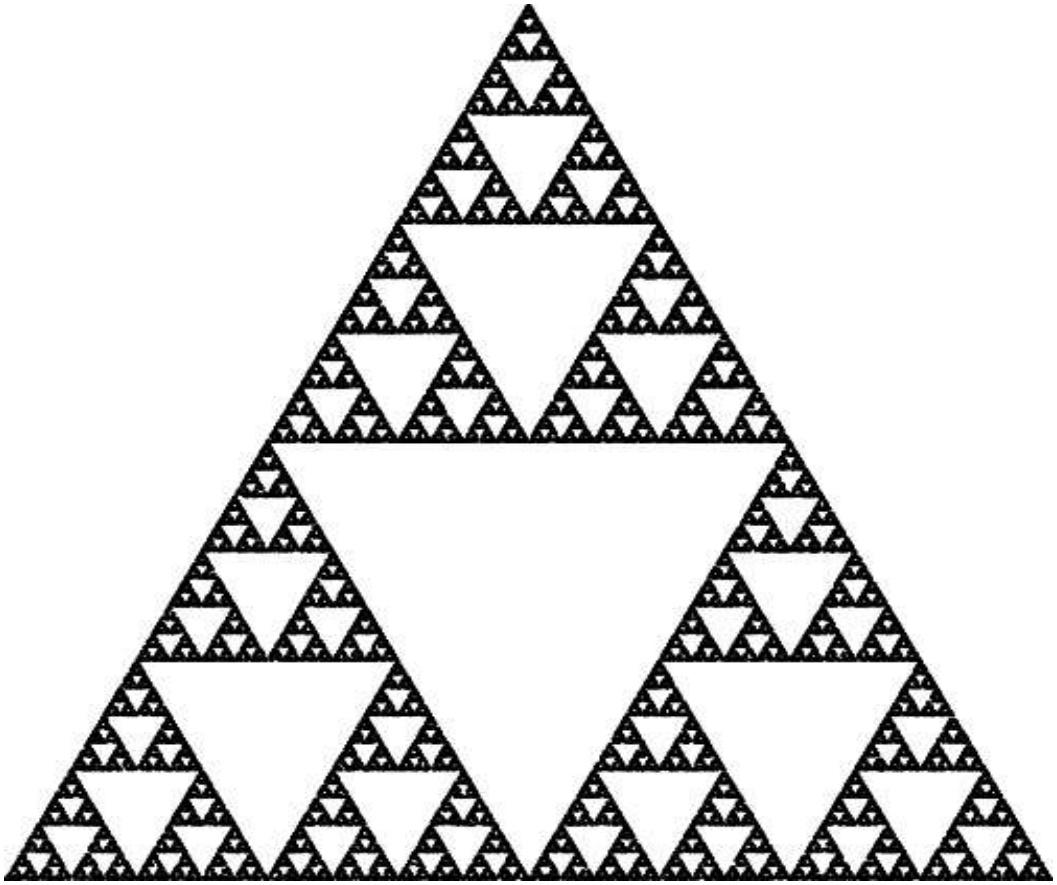
The analysis of recursive algorithm requires recurrence equations.

The recurrence equation defines a sequence using the elements of that sequence.

Here base condition or initial condition or basis step represents same terminology.

Similarly recursive step or preceding equation also represents same terminilogy.

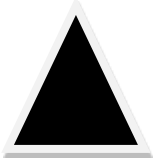
Sierpinski Triangle



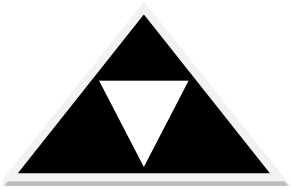
The above figure shows a Sierpinski triangle , which named after Waclaw Sierpinski, a Polish mathematician.

***The following steps of the alogirthm to create the figure*:**

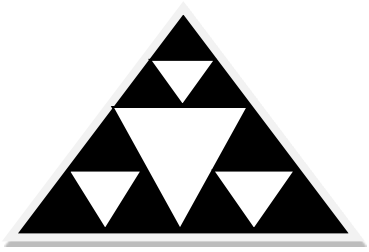
1. ***Choose any bounded triangle whose base parallel to the horizontal axis.***



2. ***Connect the midpoints of each side to form four separate triangles and cut out the triangle in the center.***



3. ***For each of the three remaining triangles, repeat step*** **2.**



4. ***Iterate infinitely.***

This repition of a method in a self − similar way is known as recursion and the process is called a recursive process.

Let us consider the sequence of positive integers $\{1, 3, 9, 27, \ldots.\}$. ***Let*** a_r. ***denote the rth term of this sequence. Then we define the rth term as:***

$$a_r = 3 \times a_{r-1}$$

$for\ r \geq 2\ with\ the\ initial\ condition\ a_1 = 1. Here\ a_{r-1}, is\ its\ previous\ term.$

$The\ above\ equation\ can\ be\ represented\ through\ a\ program.$

```cpp
#include <iostream>
using namespace std;

int calculate_a(int r)
{
    if (r == 0)
    {
        return 1; // Base case: a₀ = 1
    }
    else
    {
        return 3 * calculate_a(r - 1);
    }
}

int main()
{
    int r_value = 5;
    int result = calculate_a(r_value);
   cout << "The value of a₅ is: " << result << endl;
    return 0;
}
```

Recurrence Relation:

A recurrence relation is an equation that recursively defines a sequence; that is , each term of the sequence is defined as a function of the preceeding terms.

Difference Equation is also known as Recurrence Relation. i.e. Difference Equation is another name of Recurrence Relation.

Recursive Definition

Recursive Definition consists of the following steps:

Basis step:

This step defines primitive value or a set of primitive values.

Recursive step:

This step defines the rule(s) to find a new element from the

existing elements.

***Example* 1:**

$a)\mathbf{1, 2, 3, 4, 5, \ldots.}$

***Solution*:**

Let the nth term of the sequence be denoted by a_n .

The first term of the sequence is $\mathbf{1}$, ***and each successive term can be obtained by adding*** $\mathbf{1}$ ***to the preceeding term.***

Thus, the sequence a_n ***can be defined as follows:***

$$a_1 = 1$$

$$a_n = a_{n-1} + 1 \, , n \geq 2$$

We can write a recursive program using the above equation.

```cpp
#include <iostream>
using namespace std;
int calculate_a(int n)
{
    if (n == 1)
    {
        return 1; // Base case: a₁ = 1
    }
    else
    {
        return calculate_a(n - 1) + 1;
    }
}

int main()
{
    int n_value = 5;
    int result = calculate_a(n_value);
    cout << "The value of a₅ is: " << result << endl;
    return 0;
}
```

Recursively Defined Functions

A function whose domain is the set of non − negative integers can be defined recursively using the recursive definition. The basis step defines the function for some primitive values and the recursive step provides a way to calculate the value of the function for the other integers.

***Example* 1:**

Write the recursive definition of the function $f(x) = 2^x$ ***defined from the set of natural numbers (including*** 0***) to the set of natural numbers.***

***Solution*:**

Since $f(0) = 1$ ***and*** $f(x+1) = 2^{x+1} = 2 \times 2^x = 2 \times f(x)$,
***the function can be defined recursively as follows*:**

$i.e.\ f(0) = 1$ ***(Base Case)***

$f(1) = 2^1$ ***or*** $2 \times f(0) = 2 \times 1 = 2$

$f(2) = 2^2$ ***or*** $2 \times f(1) = 2 \times 2 = 4$

$f(3) = 2^3$ ***or*** $2 \times f(2) = 2 \times 4 = 8$

… ….

Hence:

$$f(x) = \begin{cases} 1 & for\ x = 0 \\ 2 \times f(x-1) & for\ x \geq 1 \end{cases}$$

Therefore the above function can be represented by a program:

```cpp
#include <iostream>
using namespace std;
int calculate_f(int x)
{
    if (x == 0)
    {
        return 1; // Base case: f(0) = 1
    }
    else
    {
        return 2 * calculate_f(x - 1);
    }
}

int main()
{
    int x_value = 5;
    int result = calculate_f(x_value);
     cout << "The value of f(5) is: " << result << endl;
    return 0;
}
```

Example 2:

Write the recursive definition of the function $f(x) = x!$, from the set of natural numbers (including 0) to the set of natural numbers.

$\boldsymbol{If\ we\ see\ factorial\ using\ iteration}$:

$\boldsymbol{Factorial\ of\ 0 = 1}$

$\boldsymbol{Factorial\ of\ 1 = 1}$

$\boldsymbol{Factorial\ of\ 2 = 1 * 2 = 2}$

$\boldsymbol{Factorial\ of\ 3 = 1 * 2 * 3 = 6}$

$\boldsymbol{Factorial\ of\ 4 = 1 * 2 * 3 * 4 = 24}$

$\boldsymbol{Factorial\ of\ 5 = 1 * 2 * 3 * 4 * 5 = 120}$

…….

$\boldsymbol{The\ same\ thing\ can\ be\ achieved\ by\ recursion}$:

$\boldsymbol{f(0) = 1}$

$\boldsymbol{f(1) = 1 \times f(0)\ (i.e.\ 1\ \times 0!) = 1}$

$\boldsymbol{f(2) = 2 \times f(1)\ (i.e.\ 2\ \times 1!) = 2}$

$\boldsymbol{f(3) = 3 \times f(2)\ (i.e.\ 3\ \times 2!) = 6}$

$\boldsymbol{f(4) = 4 \times f(3)\ (i.e.\ 4\ \times 3!) = 24}$

$\boldsymbol{f(5) = 5 \times f(4)\ (i.e.\ 5\ \times 4!) = 120}$

……..

$\boldsymbol{Hence\ the\ factorial\ function\ can\ be\ defined\ recursively}$:

$$f(x) = \begin{cases} 1 & for\ x = 0 \\ x \times f(x-1) & for\ x \geq 1 \end{cases}$$

We can write exhibit it through a program:

```cpp
#include <iostream>
using namespace std;

int factorial(int n)
{
    if(n==0)
    {
        return 1;
    }
    return n * factorial(n - 1);
}

int main()
{
    int n;
    cout<<"Enter the number: ";
    cin>>n;
    cout<<"Factorial of "<<n<<" is "<<factorial(n);
    return 0;
}
```

Recursively Defined Sets

A set is said to be recursively defined if the elements of the set can be defined using the recursive definition.
The basis step defines the primitive elements of the set and the recursive definition generates the other elements of the set.

Example $\mathbf{1}$: $A = \{1, 4, 7, 10, \ldots\ldots\}$

Solution:

The first term is $\mathbf{1}$, ***and each successive term can be obtained from the previous term by adding*** $\mathbf{3}$. ***Thus the elements of the set A can be defined recursively as follows***:

$(i)\ 1 \in A$

(ii) ***if*** $x \in A$, ***then*** $x + 3 \in A$

We can create a program to check whether the element is in set or not.

```cpp
#include <iostream>
using namespace std;
bool isInSetA(int x)
{
    if (x == 1)
    {
        return true;
    }
    else if (x > 1)
    {
        return isInSetA(x - 3);
    }
    return false;
}

int main()
{
    int valuesToCheck[] = {1, 4, 7, 10};
    int numValues = sizeof(valuesToCheck) /
sizeof(valuesToCheck[0]);

    for (int i = 0; i < numValues; i++)
    {
        int value = valuesToCheck[i];
        cout << value << " is" << (isInSetA(value) ? " " : " not ")
<< "in set A." << endl;
    }

    return 0;
}
```

***Example* 2**: $A = \{2, 5, 11, 23, \ldots..\}$

Solution:

The first term is* 2, *and each successive term can be obtained from the previous term by multiplying it by* 2 *and adding* 1. *Thus the elements of the set A can be defined recurisvely as follows:

$(i)\ 2 \in A$

(ii) ***if*** $x \in A$, ***then*** $x + 3 \in A$

We can generate a similar program as above and check the sequence.

****