Tutorial N°1

Exercice 01

1) Define the process of recurcivity in algorithmcs.

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

- A recursive algorithm takes one step toward solution and then recursively call itself to further move. The algorithm stops once we reach the solution.
- Since called function may further call itself, this process might continue forever. So, it is essential to provide a base case to terminate this recursion process.

2) Why is recursion needed in algorithmics?

Solution 01

1. Define the process of recurcivity in algorithmcs.

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

- A recursive algorithm takes one step toward solution and then recursively call itself to further move. The algorithm stops once we reach the solution.
- Since called function may further call itself, this process might continue forever. So it is essential to provide a base case to terminate this recursion process.

2. **Need of Recursivity**

- Recursivity helps in logic building. Recursive thinking helps in solving complex problems by breaking them into smaller subproblems.

- Recursive solutions work as a a basis for Dynamic Programming and Divide and Conquer algorithms.
- Certain problems can be solved quite easily using recursion like:
    a) Towers of Hanoi (TOH),
    b) Inorder/Preorder/Postorder Tree Traversals,
    c) DFS of Graph, etc.

Exercice 03

a) We ask you writing an algorithm that calculate a Fibonacci sequence.
b) Translate the written algorithm in C language.

Explanation:

Each number in the Fibonacci sequence is the sum of the two preceding numbers: `F(n) = F(n-1) + F(n-2)` with base cases `F(0) = 0` and `F(1) = 1`.

Solution

(n): if n <= 0: return 0 else if n == 1: return 1 a = 0 b = 1 for i from 2 to n: c = a + b a = b b = c return b

```
Algorithm Fibonacci;

Var n : integer;

Begin
    Read (n ) ;
:= 6; { Exemple : calcul du 6ème nombre de Fibonacci }
writeln('Fibonacci(', n, ') = ', Fibonacci(n)); readln; end.
<= 0: return 0 else if n == 1: return 1 a = 0 b = 1 for i
from 2 to n: c = a + b a = b b = c return b
```

Recursive Fibonacci in C

c

Copier Modifier

```c
#include <stdio.h>

// Recursive function to calculate Fibonacci
int fibonacci(int n) {
    if (n <= 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n = 6; // Example: Find the 6th Fibonacci number
    printf("Fibonacci(%d) = %d\n", n, fibonacci(n));
    return 0;
}
```

A recursive algorithm is an algorithm that invokes itself during execution with a

```c
#include <stdio.h>

// Recursive function to find the sum of
// numbers from 0 to n
int findSum(int n)
{
    // Base case
    if (n == 0)
        return 0;

    // Recursive case
    return n + findSum(n - 1);
}
```