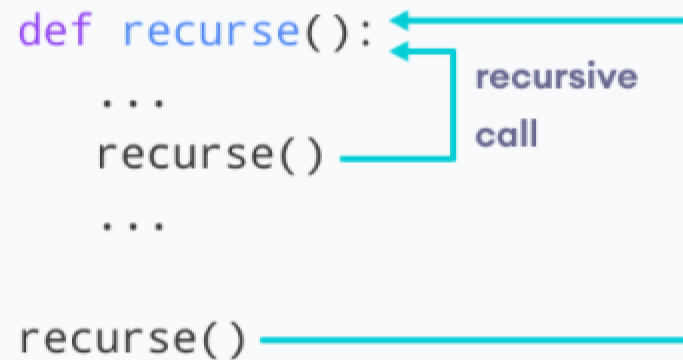
TP1: Example of a recursive function

Python Recursive Function

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

The following image shows the working of a recursive function called `recurse`.



Recursive Function in Python

TP1: Example of a recursive function

```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

Output:???

TP1: Example of a recursive function

```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

Output: The factorial of 3 is 6

TP1: Example of a recursive function

Example

Use recursion to add all of the numbers up to 10.

```java
public class MyClass {
  public static void main(String[] args) {
    int result = sum(10);
    System.out.println(result);
  }
  public static int sum(int k) {
    if (k > 0) {
      return k + sum(k - 1);
    } else {
      return 0;
    }
  }
}
```

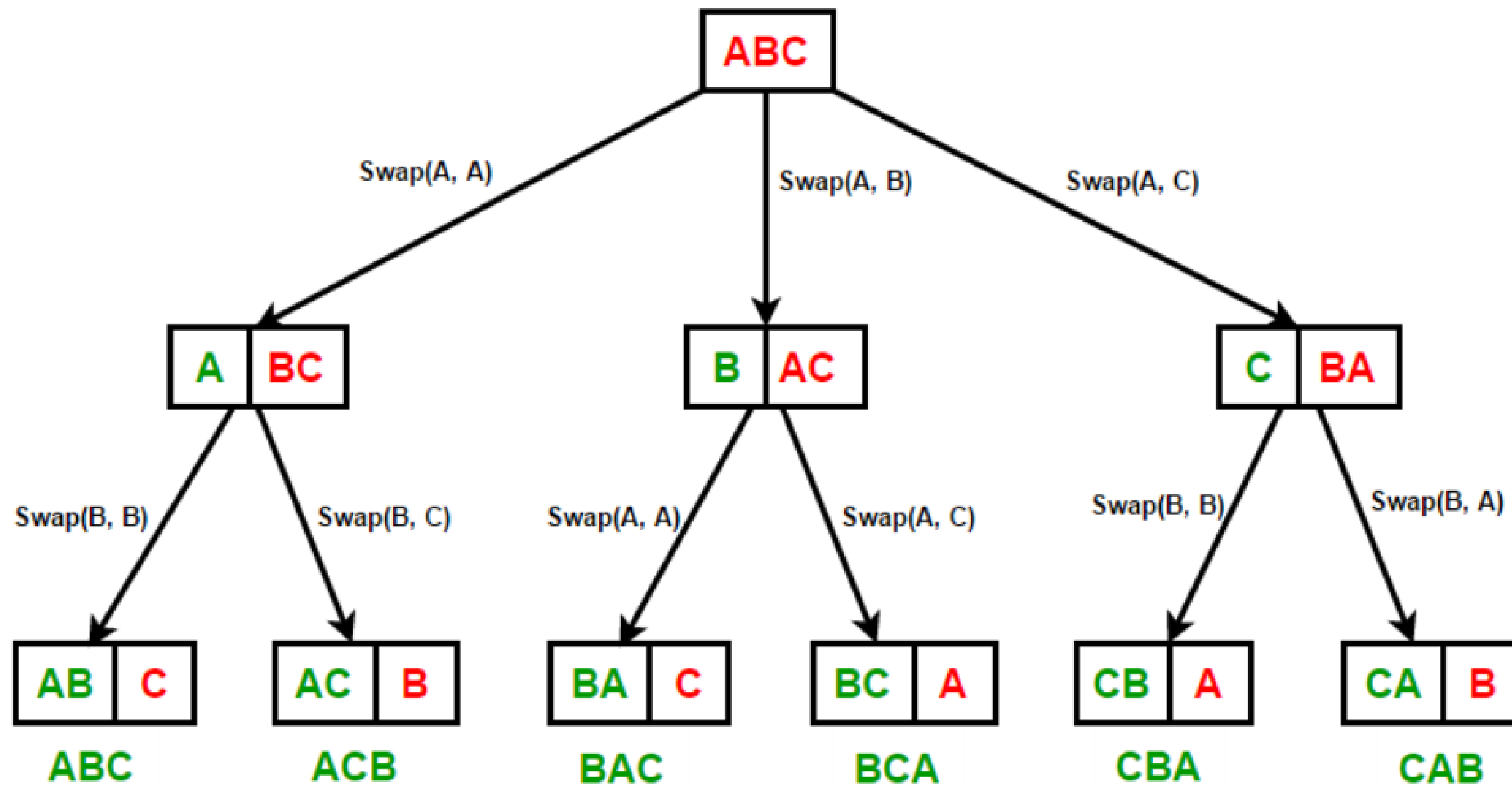
Output:???

Output:

10 + sum(9)
10 + (9 + sum(8))
10 + (9 + (8 + sum(7)))
...
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0

Permutation of a string of unique character

Below is the recursion tree for printing all permutations of string "ABC".



Recursion Tree for string "ABC"

```
class Main
{
        // Utility function to swap two characters in a character array
        private static void swap(char[] ch, int i, int j)



        // Recursive function to generate all permutations of a String
        private static void permutations(char[] ch, int currentIndex)



        // generate all permutations of a String in Java
        public static void main(String[] args)


}
```

Memoization is a technique for implementing dynamic programming to make recursive algorithms efficient. It often has the same benefits as regular dynamic programming without requiring major changes to the original more natural recursive algorithm.

Basic Idea

- The first thing is to design the natural recursive algorithm.
- If recursive calls with the same arguments are repeatedly made, then the inefficient recursive algorithm can be memoized by saving these subproblem solutions in a table so they do not have to be recomputed.

Implementation

To implement **memoization** to recursive algorithms, a table is maintained with subproblem solutions, but the control structure for filling in the table occurs during normal execution of the recursive algorithm. This can be summarized in steps:

1. A memoized recursive algorithm maintains an entry in a table for the solution to each of subproblem,
2. Each table entry initially contains a special value to indicate that entry has yet to be filled in.
3. When the subproblem is first encountered, its solution is computed and stored in the table.
4. Subsequently, the value is looked up rather than computed

To illustrate the steps above, let's take an example for computing nth Fibonacci number with a recursive algorithm as:

Implementation

To implement memoization to recursive algorithms, a table is maintained with subproblem solutions, but the control structure for filling in the table occurs during normal execution of the recursive algorithm. This can be summarized in steps:

1. A memoized recursive algorithm maintains an entry in a table for the solution to each of subproblem,
2. Each table entry initially contains a special value to indicate that entry has yet to be filled in.
3. When the subproblem is first encountered, its solution is computed and stored in the table.
4. Subsequently, the value is looked up rather than computed

What is the issue?

To illustrate the steps above, let's take an example for computing nth Fibonacci number with a recursive algorithm as:

```
// without memoization
static int fib(int n) {
   if (n == 0 || n == 1) return n;
   return fib(n - 1) + fib(n - 2);
}
```

Maximum Product of Two Sequences Problem