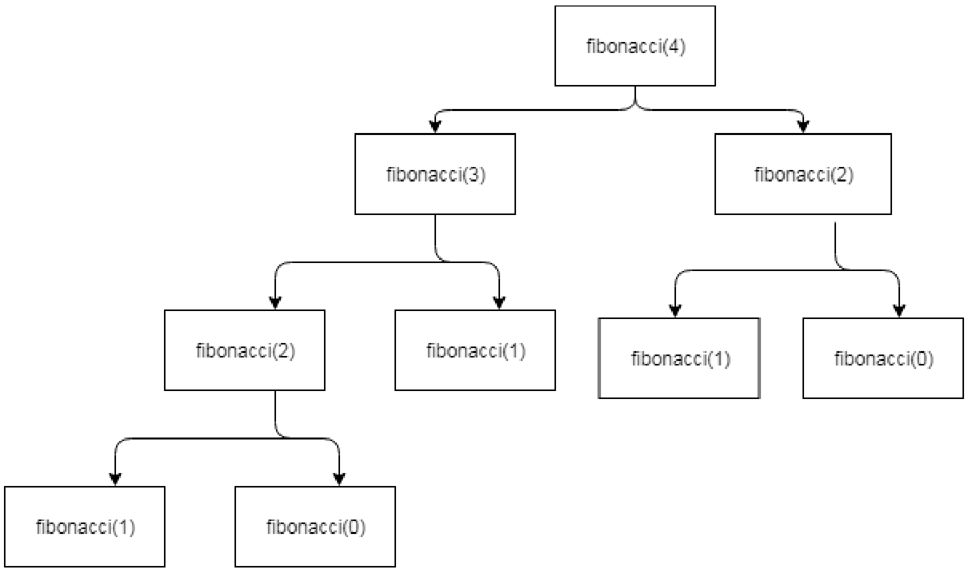
Problem 1 Consider the recursive functions.

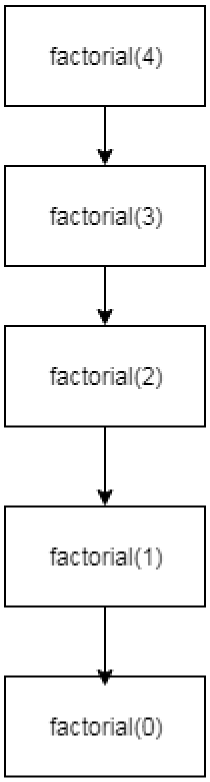
a) Draw the recursion tree generated by calling fibonacci(4).

Sol:



b) Draw the recursion tree generated by calling factorial(4).

Sol:



c) By referring to the recursion trees drawn above, explain why recursion (the recursion fairy) is more efficient at solving factorial than fibonacci.

Sol: The recursion is more efficient in solving factorial rather than fibonacci because in factorial as you can see from recursion tree ,we are only solving the function at the different numbers once , while in fibonacci recursion tree from above ,we are solving the function(2) twice ,which leads the recursion to do more work as need to solve the same problem again and again.so in fibonacci recursion solving same problem multiple times makes it less efficient than factorial.

Problem 2)

For each of the following question, write down an algorithm using pseudo-code.

a) Write down an algorithm which takes the root of a tree as argument and returns the size of the tree.

Sol :

```
function sizeOfTree(root):
   If the root has no children:
        return 1
   else:
        count=1
        for each child in root.children:
             number = sizeOfTree(child)
             count = count + number
        return count
```

b)Suppose you are given a tree in which values are numbers.

Write down an algorithm which takes the root of such a tree as argument and returns the sum of all numbers in the tree.

Sol:

```
function sumValuesTree(root):
    If the root has no children:
        return   root.value
    else:
        count=root.value
        for each child in root.children:
            number = sumValuesTree(child)
            count = count + number
        return count
```

Problem 3)

We previously had to find an algorithm to decide whether a list of numbers is already sorted in increasing order or not. In this question, we will write a Python recursive function is_increasing to solve this problem.

A list is considered to be increasing if each of its element is greater than or equal to its predecessor. Remember that a constant or empty list is considered to be increasing.

a) What is the base case of the algorithm is_increasing.

Sol : The most easier case is ,

If the list is empty then ,it would return True.

b) What is the recursion step of the algorithm is_increasing? In other words, how do you break down the problem to a smaller one to ask the recursion fairy?

Sol : I will compare the last two numbers in the list , if they are in increasing order then we ask the recursion fairy to do computation for list sliced having length one less from previous list(neglecting last number)

```
length=list.length
If list[length-1]<=list[length-1]:
      Check=recursion fairy check for list[  : length-1]
       return Check
return False
```

Problem 4)   A key feature available across most text-based programs, whether they are text editor, web browser or pdf reader, is to be able to search a given word in the text.

By example, it is likely that if you press CTRL-F right now, the program you are using will prompt you for a word to search and highlight in this document. String-searching algorithms are an important category of algorithms which are meant to efficiently find one or more occurence of a word or sentence into text.

In this problem we will write a recursive function word_in_string which finds whether a word in contained in a text.

a) What are the base cases (there are two) of the algorithm word_in_string

Sol : First base case: if length of word is greater than string then it would return False.

Second base case : if word matched the string ,then it would return True.

b) What is the recursion step of the algorithm word_in_string? In other words, how do you break down the problem to a smaller one?

Sol : I will compare the word with the part of a string from last, whose length is equal to word length ,if it matched then it would return True else ,I would ask the recursion fairy to do computation for string sliced by removing last character.