Exercice 1

On rappelle la fonction récursive `fibo_rec` qui calcule le nième terme de la suite de fibonacci définie par:

$$u_0 = 1 \ , \ u_1 = 1 \quad \text{et pour tout } n \geqslant 2 : u_n = u_{n-1} + u_{n-2}$$

```
def fibo_rec(n):
    '''
    une version récursive
    n est le rang du terme de la suite qu on veut calculer.

    '''

    if n <= 1:
        return n
    return fibo_rec(n-1) + fibo_rec(n-2)
```

1. Calculer u_5 en utilisant l'algorithme récursif proposé.
2. Établir un arbre récursif qui schématise le fonctionnement récursif de l'algorithme.
3. Donner le nombre d'additions effectuées pendant l'exécution de l'appel :

   ```
   fibo_rec(5)
   ```

   puis l'appel

   ```
   fibo_rec(7)
   ```

4. Quelle remarque peut-on faire quand aux opérations effectuées pour calculer les termes de la suite de fibonacci en utilisant l'algorithme précédent?
5. Estimez la complexité de cet algorithme récursif.
6. L'arbre précédent montre clairement que la même fonction `fibo_rec` est appelée plusieurs fois pour le calcul des mêmes termes (chevauchement de sous-problèmes). D'où l'idée de réécrire la fonction , toujours de manière récursive , mais en la modifiant de manière à ce qu'elle sauvegarde les termes calculés une première fois. Compléter la fonction `fibo_memo_rec` ci-dessous

   ```
   def fibo_memo(n):
       T=[0]*(n+1)
       T[0],T[1] = 1,1
           return fibo_memo_rec(T,n)
   def fibo_memo_rec(T,n):
       #Si le terme de rang n a déjà été calculé
       if T[n]>0:
           return .......
       #le cas de base
       if ..............:
           return 1
       else:
           T[n]= ...................
       return .....
   ```

   La fonction `fibo_memo_rec` commence par vérifier si elle a déjà résolu le problème. Si c'est le cas , elle renvoie la valeur sauvegardée dans le tableau T , évitant ainsi la répétition de calculs déjà effectués.