TD3 : Fonctions récursives (1)

Compétences

- Lire et prévoir le résultat d'une fonction récursive
- Évaluer la terminaison d'une fonction récursive

Exercices

* **Ex. 1 —** On considère les fonctions `foo1` et `foo2` suivantes :

```python
def foo1(n):
    if n == 0:
        print(0)
    else:
        print(n)
        foo1(n - 1)
```

```python
def foo2(n):
    if n == 0:
        print(0)
    else:
        foo2(n - 1)
        print(n)
```

Décrire précisément la pile d'exécution des fonctions `foo1` et `foo2` pour $n = 3$

* **Ex. 2 —** On considère les fonctions `foo3` et `foo4` suivantes :

```python
def foo3(n):
    if n == 0:
        print(0)
    else:
        print(n)
        foo4(n - 1)
```

```python
def foo4(n):
    if n == 0:
        print(0)
    else:
        foo3(n - 1)
        print(n)
```

Décrire précisément la pile d'exécution des fonctions `foo3` et `foo4` pour $n = 3$

* **Ex. 3 —** On considère la fonction `puissanceRecur` suivante :

```python
def puissanceRecur (a, n):
    if n == 0:
        return 1
    return a * puissanceRecur(a, n-1)
```

Décrire précisément la pile d'exécution pour l'appel `puissanceRecur(2, 4)`

** **Ex. 4 —** Monsieur ça marche ! Cette réponse est un classique chez les étudiants. Les principales difficultés sont souvent liées à de mauvaises évaluations des coûts tant temporels que spatiaux. Partons d'un exemple connu, l'algorithme de recherche dichotomique. Étant donné un tableau **`t`** de taille **`n`** contenant une **`liste triée`** par ordre croissant d'éléments et un élément **`x`**, on cherche à déterminer si **`x`** se trouve dans **`t`**. Cette algorithme se prête facilement à une programmation récursive.

1. Proposer une version récursive de la fonction `binarySearch(x, t, g, d)`
2. Faire une évaluation de la complexité dans le pire cas de cet algorithme. Conclure.