

TD n°4

Diviser pour regner, et introduction à la programmation dynamique

Exercice 1 : Point fixe

Donnez un algorithme qui prend en entrée un tableau trié T de n entiers (positifs ou négatifs) tous distincts, et qui décide s'il existe un indice i tel que $T[i] = i$. Donnez la complexité de votre algorithme.

Exercice 2 : Élément majoritaire

Un élément est majoritaire s'il apparaît plus de $n/2$ fois dans un tableau de taille n . On a un type de données qui ne supporte que l'opérateur d'égalité $=$. Donner un algorithme de type diviser pour régner pour trouver l'élément majoritaire, s'il existe. Donner la complexité de votre algorithme.

Exercice 3 : Le triangle de Pascal

On rappelle la récurrence du triangle de Pascal : $C_n^p = \begin{cases} 1 & \text{si } p = 0 \text{ ou } p = n \\ C_{n-1}^{p-1} + C_{n-1}^p & \text{sinon} \end{cases}$

1. Quel est le temps d'exécution du programme à droite ? Donnez une borne inférieure au nombre d'appels récursifs faits, à n fixé, pour la plus mauvaise valeur de p .

```
int C(int n, int p) {  
    if(p==0 || p==n) return 1;  
    else return C(n-1,p-1) + C(n-1, p);  
}
```

On considère ces deux programmes. Dans le pseudo-C utilisé, quand un tableau est déclaré il est rempli de 0 ; un tableau et une fonction peuvent avoir le même nom ; enfin majuscules et minuscules sont différenciés.

```
int C[N][P];  
  
for(n=0;n<=N;n++)  
    for(p=0;p<=n;p++)  
        if(p==0 || n==p)  
            C[n][p] = 1;  
        else  
            C[n][p] = C[n-1][p-1] + C[n-1][p];  
return C[N][P];
```

```
int C[N][P];  
  
int C(int n, int p) {  
    if(C[n][p] == 0)  
        if(p==0 || p==n)  
            C[n][p] = 1;  
        else  
            C[n][p] = C(n-1,p-1) + C(n-1, p);  
  
    return C[n][p];  
}
```

2. Quel est le temps de calcul du programme de gauche ?
3. Faire le calcul de $C(4,3)$ avec le programme de droite.
4. Montrer qu'il calcule bien C_n^p
5. En combien de temps ?

Exercice 4 : Fibonacci

Question 1 : Faire une fonction purement récursive qui calcule le n ième nombre de Fibonacci, obtenu

$$\text{par } F_n = \begin{cases} 1 & \text{si } n = 1 \text{ ou } n = 2 \\ F_{n-1} + F_{n-2} & \text{sinon} \end{cases}$$

Quelle est sa complexité?

Question 2 : En vous inspirant de la question 3, faire une fonction purement récursive `int F(int n)` qui calcule le n ième nombre de Fibonacci