Travaux Dirigés d'Algorithmique n°2

Cours d'Informatique de Deuxième Année

—DEUG SCIENCES M3-MI—

Récursivité, Complexité d'algorithme

La récursivité est certainement la manière la plus "naturelle" de coder la plupart des fonctions mathématiques récursives (telle factorielle), mais a pour principal inconvénient d'obliger le compilateur à utiliser une pile pour mémoriser les calculs intermédiaires.
Par ailleurs, il est possible que le fait d'écrire les algorithmes instantanément tels qu'on les perçoit conduise à des temps d'exécution rédhibitoires. Il est alors important d'estimer avant l'exécution physique, non seulement le temps, mais aussi parfois l'espace nécessaires au programme : on parle de complexité d'algorithme.

▸ **Exercice 1. (somme des premiers entiers impairs)** *Soit S_n la somme définie par :*

$$S_n \quad = \quad 1 + 3 + 5 + \cdots + (2n - 1)$$

- *trouver un algorithme récursif pour calculer S_n ;*
- *en déduire une fonction* `Somme` *prenant en argument un entier* `n` *et qui implémente cet algorithme ;*
- *évaluer la complexité en temps de cet algorithme.*

▸ **Exercice 2.** *(**fonction puissance**) Nous nous proposons de calculer récursivement a^n (a et n entiers), de trois manières différentes. Donner une fonction pour chacune des formules récursives suivantes et estimer la complexité en temps des trois algorithmes de calculs : $a^0 = 1$ et si $n > 0$:*

1. $a^n := a \times a^{n-1}$;
2. *si n est pair alors* $a^n := (a^2)^{\frac{n}{2}}$ *sinon,* $a^n := a \times (a^2)^{\frac{n-1}{2}}$;
3. *si n est pair alors* $a^n := (a^{\frac{n}{2}})^2$ *sinon,* $a^n := a \times (a^{\frac{n-1}{2}})^2$.

▸ **Exercice 3. (Recherche dans un tableau trié)** *Pour qu'un élément appartienne à un tableau trié en ordre croissant, il suffit de tester s'il est plus grand que l'élément du milieu du tableau. Si c'est le cas, il ne reste plus qu'à relancer la recherche dans la partie droite du tableau, sinon dans la partie gauche.*

- *Ecrire une fonction récursive* `Dichotomie` *qui prend en arguments un tableau d'entiers, sa taille, ainsi que l'élément à trouver et qui renvoie vrai si l'élément est dans le tableau.*
- *Evaluer la complexité en temps de la fonction* `Dichotomie`*.*

▸ **Exercice 4. (Image miroir d'une chaîne de caractères)** *Par définition, l'image miroir d'une chaîne de caractères* `ch` *est la chaîne obtenue par la lecture de droite à gauche de* `ch`*. Par exemple, l'image miroir de* `"abcdefgh"` *est* `"hgfedcba"`*.*

- *Trouver un algorithme récursif qui affiche l'image miroir d'une chaîne de caractères.*
- *En déduire une procédure* `AfficheMiroir`*.*
- *Donner la complexité en temps de la procédure* `AfficheMiroir`*.*

▸ **Exercice 5. (Somme d'un sous-tableau)** *On considère un tableau d'entiers relatifs de taille* `TAILLE_MAX`*. Le but de cet exercice est d'extraire un sous-tableau contigu dont la somme des éléments est maximale. Nous allons utiliser des algorithmes qui utilisent des stratégies de résolution différentes. Ecrire avant tout une fonction* `max` *qui retourne le maximum de deux entiers.*

1. *la façon la plus naturelle de résoudre ce problème est de prendre toutes les paires d'indices* $(i, j), i < j$ *et de recalculer la somme du sous-tableau compris entre* i *et* j *;*
2. *une deuxième méthode consiste à reprendre l'algorithme ci-dessus en utilisant la somme calculée à l'étape précédente.*
3. *on peut aussi se servir du principe de "diviser pour régner" (qui est récursive) :*
   - *on coupe le tableau en deux parties égales* A *et* B*.*
   - *on calcule le maximum sur* A*, sur* B *et sur un sous-tableau chevauchant* A *et* B *;*
   - *le résultat cherché est alors le maximum des trois valeurs calculées (pourquoi ?).*
4. *voici un autre algorithme :*

```
MaxActuel := 0; MaxDroit  := 0;
pour i:= 1 .. TAILLE_MAX faire debut
        MaxDroit  := max (MaxDroit+Tab[i], 0);
        MaxActuel := max (MaxActuel, MaxDroit);
        fin;
Retourner MaxActuel.
```

   Expliquez son principe.

Donner la complexité en temps de chacun des algorithmes. Lequel choisiriez-vous ?