Algorithmique

TD 1 : Compter & Prouver

Exercice 1 (Puissance) Donnez deux algorithmes (un itératif & un récursif) rendant, à partir de deux entiers positifs x & y, le nombre x^y. Prouvez-les & donnez leurs complexités.

Exercice 2 (Maximum d'un tableau) Donnez deux algorithmes (un itératif & un récursif) qui calculent la plus grande valeur d'un tableau de nombres. Comparez leurs fonctionnements.

Exercice 3 (Algorithme mystère) Que fait l'algorithme 1 ? Justifiez votre réponse. Donnez sa complexité dans le cas le meilleur. En déduire la complexité dans le cas le pire & en moyenne. Donnez une version récursive de cet algorithme.

Algorithme 1

Données : A & B deux entiers positifs
Début
  $x \leftarrow A$; $y \leftarrow B$; $R \leftarrow 1$
  tant que $y \neq 0$ **:**
    si y est impair **:**
      $R \leftarrow R \times x$
      $y \leftarrow$ y - 1
    sinon :
      $x \leftarrow x \times x$
      $y \leftarrow y/2$
  rendre R
Fin

Exercice 4 (Palindrome) Implémentez un algorithme récursif permettant de savoir si un tableau de caractères est un palindrome (un palindrome se lit "à l'endroit" & "à l'envers" de la même façon, comme par exemple "À l'étape, épate-la !" (on considère ni la ponctuation, ni les espaces, ni les accents)). Quelle est sa complexité ?

Exercice 5 (Tours de Hanoï) Les *tours de Hanoï* sont un célèbre casse tête qui consiste à déplacer n disques de diamètres différents d'une tour de "départ" à une tour d' "arrivée" en passant par une tour "intermédiaire", tout en respectant les règles suivantes :

- on ne peut déplacer qu'un disque à la fois,
- on ne peut placer un disque sur un disque plus petit que lui.

On suppose que cette dernière règle est également respectée dans la configuration de départ.

Donnez un algorithme récursif permettant de résoudre le problème. Quelle est sa complexité ? Peut-on faire mieux ?

Exercice 6 (Ackermann) La fonction d'Ackermann se définit de la manière suivante, pour tous entiers m & n positifs :

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } n = 0 \\ A(m-1, A(m,n-1)) & \text{sinon} \end{cases}$$

Cette fonction est d'une importance théorique certaine (c'est un exemple de fonction récursive, & non récursive primitive) & est célèbre du fait qu'elle grossit très vite alors qu'elle s'écrit simplement.

Vérifiez que cette définition est valide (*i.e.* s'arrête quelque soient m & n).

Exercice 7 (Suppression de doublons) La structure de donnée utilisée ici est la *liste*. On considérera que la création d'une liste vide, l'ajout d'un élément en fin de liste & la lecture d'un élément dans une liste se font en $\mathcal{O}(1)$ opérations[1].

1. Donnez un algorithme permettant de résoudre le problème suivant :
   - **Données :** Une liste `L` & une valeur `val`.
   - **Rendre :** Une liste `L_2`, restriction de `L` aux valeurs différentes de `val`.

   Quelle est sa complexité ?
2. Utilisez la question précédente pour écrire un algorithme résolvant le problème suivant :
   - **Données :** Une liste `L`.
   - **Rendre :** Une liste `L_2` ne contenant qu'une seule occurrence de chaque valeur de `L` & en conservant le même ordre.

   Quelle est sa complexité ?
3. Même question que précédemment, mais on considère que la liste `L` en entrée est triée. Donnez un algorithme en $\mathcal{O}(n)$ pour résoudre ce problème, où n est le nombre d'éléments de `L`.
4. Si l'ordre des éléments de `L_2` n'est pas important, proposez une meilleure solution à la question du point 2.

Exercice 8 (Listes) On appelle *liste* une structure abstraite ordonnée telle que on puisse accéder de manière directe à l'élément d'indice i & à laquelle on puisse ajouter (& supprimer) autant d'éléments que l'on souhaite. Une caractéristique importante de cette structure est son nombre d'éléments.

Un implémentation des listes peut être effectuée comme suit :

- on commence par créer un tableau de taille $t = 1$, le nombre initial d'éléments étant $n = 0$.
- à chaque ajout d'élément :
  - test si $n < t$:
    - si oui, alors $n \leftarrow n + 1$.
    - sinon :
      - on alloue un tableau à $2 * t$ éléments & $t \leftarrow 2 \times t$
      - on copie les n premiers éléments du tableau initial dans le nouveau tableau & on supprime le tableau initial.
      - $n \leftarrow n + 1$.
  - décalage de tous les éléments d'indice supérieur au rang de l'ajout & insertion de l'élément.

Montrez que la complexité de l'ajout de n éléments à la fin d'une liste originellement vide est $O(n)$. Déduisez-en que les hypothèses de l'exercice précédent sont correctes.

[1] On montrera que ces hypothèses sont correctes à l'exercice suivant.