INFORMATIQUE

TD 1 : Complexité

Exercice 0 (Addition) Donnez des algorithmes pour additionner deux entiers.

Exercice 1 (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 2 (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. 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 3 (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 :
  - si $n < t$: $n \leftarrow n + 1$
  - 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.

Exercice 4 (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 5 (Tri par Base) Ce tri s'applique *uniquement aux entiers positifs*, que l'on considère écrits en base 2[1]. Le principe de ce tri est très simple :

1. On considère d'abord le bit de poids le plus faible (i.e. le plus à droite) & on répartit l'ensemble à trier en deux sous ensembles :
   - les entiers dont le bit de poids le plus faible est 0
   - les entiers dont le bit de poids le plus faible est 1
2. On concaténe les deux sous-ensembles, en commençant par celui des bits à 0.
3. On recommence sur le bit à gauche de celui qu'on vient de traiter.
   ...

Les parcours d'ensembles se font, toujours, de la gauche vers la droite.

Donnez le pseudo-code de cet algorithme (on supposera que l'on dispose d'une fonction qui, étant donnés deux entiers n & i, donne le i^{me} bit de n).

Prouvez cet algorithme.

Donnez la complexité de cet algorithme. Rappelez la complexité minimale du tri (dans le cas le pire). Commentaires.

Exercice 6 (Quicksort) Cet algorithme est lui aussi une application du *diviser-pour-régner*. Son principe est le suivant : On sépare les éléments du tableau à trier T en deux : ceux qui sont plus petits que le premier élément de T & ceux qui sont plus grands. On obtient deux "petits tableaux" que l'on trie puis que l'on concatène.

Donnez une version de cet algorithme n'utilisant pas de tableaux auxiliaires.

[1] Il est adaptable aux autres types de nombres (& en particulier, on peut très bien compter en base 10), mais ce n'est pas ici le sujet.