Algorithmique

TD 2 : Trier & Chercher

Exercice 0 (Recherche Dichotomique) Donnez un algorithme qui cherche un élément dans un tableau trié. Prouvez-le. Quelle est sa complexité ?

Cette méthode est-elle applicable à la recherche d'un élément dans un tableau quelconque (i.e. non trié) ? Quelle est la complexité minimale de la recherche dans un tableau non trié ?

Exercice 1 (Un Tri Naïf : le Tri par Insertion) On considère le (principe de) tri suivant : *On prend les éléments du tableau les uns après les autres & on les place parmi les éléments déjà triés.*

Prouvez cet algorithme & proposez-en une version en pseudo-code.

Quelle est la complexité, dans le cas le pire, le meilleur & en moyenne de cet algorithme ? Quelles hypothèses avez-vous dû faire pour le calcul en moyenne ? Conseillerez-vous cet algorithme ?

Exercice 2 (Mergesort) Cette méthode de tri utilise le principe classique en algorithmique du *diviser-pour-régner*. Le principe général de cette technique est de découper le problème global en deux sous problèmes que l'on traite séparément. On combine ensuite les deux solutions des sous-problèmes pour fabriquer une solution au problème global.
Ici, on va d'abord séparer le tableau en deux parties à-peu-près de même taille, puis on va trier (séparément) les deux petits tableaux, & enfin on va fusionner les deux petits tableaux triés.

Donnez le pseudo-code de cet algorithme (il est conseillé d'écrire deux fonctions, une qui fait la fusion de deux tableaux & une qui trie).

Quelle est la complexité de cet algorithme ?

Exercice 3 (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 concaténe.

Donnez le pseudo-code de cet algorithme.

Quelle est sa complexité dans le cas le meilleur, le pire & en moyenne ? A quoi correspondent ces cas ?

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

Exercice 4 (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.

[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.