Algorithmie

TD 2 (Tris) Autonomie

1 Tris Naïfs

1.1 Tri par sélection

Ce tri fonctionne ainsi :

> A l'étape k de l'algorithme, on cherche le k^{me} plus petit élément que l'on place à la la k^{me} position du tableau à trier.

Combien d'étape possède cet algorithme ? Le tableau est-il bien trié à la fin de celui-ci ?

Proposez une version en pseudo-code de cet algorithme.

1.2 Tri par insertion

Ce tri fonctionne ainsi :

> On considère les éléments du tableau les uns après les autres que l'on place parmi les éléments déjà triés. (ceux qui le précèdent)

Combien d'étape possède cet algorithme ? Le tableau est-il bien trié à la fin de celui-ci ?

Proposez une version en pseudo-code de cet algorithme.

1.3 Complexité maximale

Quelle est la complexité de ces deux algorithmes ?

1.4 Complexité en moyenne

Calculez la complexité en moyenne de ces deux algorithmes. Quelles sont les hypothèses à effectuer pour ce calcul ?

1.5 Cas d'utilisation

Dans quels cas ces deux algorithmes vont-il être efficaces ?

Pour quels cas particuliers ?

2 Tri par Fusion

Cette méthode de tri utilise le principe classique en algorithmie 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.

2.1 Algorithme

Supposons que nous possédions un méthode de fusion permettant à partir de deux tableaux triés de construire un tableau trié contenant les valeurs des deux précédents tableaux.

Donner un pseudo-code de l'algorithme du tri par fusion.

2.2 Fusion de deux tableaux triés

La difficulté de conception de ce type d'algorithme est dans la combinaison des solutions locales en une solution globale. Cette combinaison doit être effectuée de la façon la plus rapide possible, c'est à dire en utilisant au maximum les 2 solutions locales. Essayons de le faire pour le tri.

Supposons que nous ayons deux tableaux $t1$ & $t2$ triés. Proposez un algorithme en $\mathcal{O}(|t1| + |t2|)$ opérations permettant de créer un tableau t trié contenant les valeurs de $t1$ & de $t2$.

2.3 Complexité maximale

Quel est la complexité de cet algorithme ?

3 Tri par Base

Ce tri s'applique *uniquement aux entiers positifs*[1], que l'on considère écrits en base 2. 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.

[1] Il est apdatable aux autres types de nombres, mais ce n'est pas ici le sujet.

3.1 Algorithme

- 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.

3.2 Complexité

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

4 Tri Rapide

Le tri rapide (*quicksort* pour les anglicistes) est une méthode de tri très populaire, & ceci pour plusieurs raisons, en particulier :

- facile à implémenter,
- efficace pour de nombreux types de données,
- peu gourmand en ressources.

Cependant, il n'est pas parfait puisqu'il effectue un nombre quadratique d'opérations dans le pire des cas (comme vous allez le calculer).

4.1 Algorithme

L'algorithme 1 est une implémentation du tri rapide. Pourquoi est-ce une méthode de tri ?

Algorithm 1: TRIRAPIDE – Initialisé par gauche ← 0 & droite ← $n-1$

```
Input: tab ; gauche ; droite
begin
    if droite > gauche then
        i ← gauche ;
        j ← droite−1 ;
        pivot ← tab[droite] ;
        while i < j do
            while tab[i] ≤ pivot do
                i ← i+1;
            while tab[j] > pivot do
                j ← j−1;
            if i < j then
                tab[i], tab[j] ← tab[j], tab[i] (échange) ;
        tab[i], tab[droite] ← tab[droite], tab[i] (échange) ;
        TRIRAPIDE(tab, gauche, i−1) ;
        TRIRAPIDE(tab, i+1, droite) ;
    return tab
end
```

4.2 Complexité maximale

Quelle est la complexité de cet algorithme ?

4.3 Complexité en moyenne

Calculez la complexité en moyenne du tri rapide. Quelles sont les hypothèses à effectuer pour ce calcul ?