TD Complexité Algorithmique

Licence 2 Informatique

Année académique 2024-2025

Exercice 2.

1/ On considère un tableau trié en ordre croissant de n éléments dans lequel on cherche un élément X. On utilise pour cela la recherche dichotomique suivante :

- $a = 1$;
- $b = n$;
- $trouve = faux$;

tant que *(*$trouve = faux$*) et* $(b - a) \geq 0$ **faire**
  $m = \lfloor \frac{a+b}{2} \rfloor$;
  si $T[m] == X$ **alors**
    $trouve = vrai$;
  fin
  sinon si $T[m] > X$ **alors**
    $b = m - 1$;
  fin
  sinon
    $a = m + 1$;
  fin
fin

Algorithme 1: Recherche dichotomique

Combien d'itérations, au maximum, l'algorithme effectuera si $n = 2^k$ où k est un entier positif.

2/ Quelle complexité, au pire des cas, a-t-on si la recherche se fait de manière séquentielle ?

Corrections :

1/ Soient a_i et b_i les valeurs a et b calculées à chaque itération de la boucle "pour". On peut montrer que :

$$b_i - a_i \leq \frac{b_{i-1} - a_{i-1}}{2} - 1 \leq \frac{b_{i-1} - a_{i-1}}{2}$$

D'où le fait que $b_i - a_i \leq \frac{b_0 - a_0}{2^i} - 1$ où $a_0 = 1$ et $b_0 = 2^k$.

Il s'en suit que $b_i - a_i \leq \frac{b_0 - a_0}{2^i} - 1 \leq \frac{2^k}{2^i}$.

Or la boucle s'arrête quand $b_i - a_i < 1$. Il faut pour cela que $i \geq k + 1$. D'où le résultat.

On peut donc écrire que l'algorithme est en $O(\log_2(n))$ au pire des cas.

2/ A titre comparatif une recherche séquentielle est en $O(n)$ au pire.

□

Exercice 3. On désigne par $C(n)$ la complexité du tri fusion où n est la taille du tableau à trier. n est supposé être une puissance de 2. Compte tenu de la récursivité dans l'algorithme, on sait que :

$$C(n) = 2C(n/2) + \Theta(n).$$

Montrer que $C(n) \in \Theta(n\log(n))$.

Si $n = 2^k$, *on a :*

- $C(n) = 2C(n/2) + \Theta(n)$
- $C(n/2) = 2C(n/2^2) + n/2$
- $C(n/2^2) = 2C(n/2^3) + n/2^2$
- ...
- $C(n/2^i) = 2C(n/2^{i+1}) + n/2^i$
- ...
- $C(n/2^{k-1}) = 2C(n/2^k) + n/2^{k-1}$

$C(n) = k2^k = n\log_2(n)$ □

Exercice 5. Soit T un tableau de n entiers et x un autre entier quelconque.

1/ Ecrire un algorithme de complexité $\Theta(n^2)$ permettant de déterminer s'il existe (ou non) deux indices distincts i et j tels que $T[i] + T[j] = x$.

Corrections : L'algorithme est le suivant :

```
trouve = faux ;
i = 1 ;
tant que (trouve == faux et i ≤ n) faire
|   j = i + 1;
|   tant que (trouve == faux et j ≤ n) faire
|   |   si T[i] + T[j] == x alors
|   |   |   trouve = vrai;
|   |   fin
|   |   sinon
|   |   |   j = j + 1;
|   |   fin
|   fin
|   i = i + 1;
fin
```

Algorithme 2: Recherche en $\Theta(n^2)$

2/ Peut-on aussi affirmer que cet algorithme est en $O(n^2)$?

Corrections : Oui car $\Theta(n^2) \subset O(n^2)$ (voir cours).

3/ On suppose que l'on dispose d'une méthode de tri en $O(n\log(n))$ permettant de trier T en ordre croissant. En déduire un algorithme de complexité globale aussi $O(n\log(n))$ répondant à la question 1.

L'algorithme est le suivant :

```
trouve = faux ;
i = 1 ;
j = n ;
tant que (trouve == faux et i < j) faire
    si T[i] + T[j] == x alors
        trouve = vrai;
    fin
    sinon si T[i] + T[j] < x alors
        i = i + 1;
    fin
    sinon
        j = j − 1;
    fin
fin
```

Algorithme 3: Recherche linéaire en $\Theta(n)$ avec T supposée triée.

Cette recherche étant en $\Theta(n)$ et le tri étant supposée en $O(n\log(n))$. L'algorithme est globalement en $O(n\log(n))$.

$\square$