Université de Lille

FST- Département Informatique

ACT

2018–2019
Master 1 d'Informatique
TD-Feuille 2

Diviser pour régner

Rappel du Master Theorème. *Si* $T(n) = aT(\lfloor n/b \rfloor) + O(n^d)$, *avec* $a > 0, b > 1, d \geq 0$ *alors*

- *si* $d > log_b a$, $T(n) = O(n^d)$
- *si* $d = log_b a$, $T(n) = O(n^d logn)$
- *si* $d < log_b a$, $T(n) = O(n^{log_b a})$

Exercice 1 : Recherche de la première occurrence d'une valeur

Soit T un tableau trié de n entiers et x un entier. On cherche si x est dans le tableau, et, si oui, on cherche le plus petit indice i du tableau tel que $T[i] = x$, donc i tel que $T[i] = x$ et T[j]<x, si $0 \leq j < i$.

Par exemple soit T=[3,7,7,8,9,9,9,10,11]; pour $x = 7$, l'indice retourné est 1, pour $x = 9$, c'est 4, pour $x = 11$ c'est 8; pour $x = 6$, la sortie est "Absent du tableau".

Proposer un algorithme en $O(\log n)$ pour le problème et **justifier** qu'il est correct.

Exercice 2 : Recherche d'une valeur encadrée

Soit T un tableau trié de n entiers et a et b deux entiers ($a < b$). On cherche un algorithme en $O(\log n)$ qui retourne $Vrai$ Ssi il existe i tel que $a < T[i] < b$. Pour chacun des algorithmes suivants dire si ils correspondent bien à cette spécification. Justifier.

```
//A1;
for (int i=0;i<n;i++)
   if ((T[i]>a) && (T[i]<b)) return true;
return false;
```

```
//A2;
int m;
int g=0;
int d=n-1;
  while (g<d)  {
     m= (g+d) /2;
     if (T[m]>=b) d=m;
     else if (T[m]<= a) g=m;
     else return true;
  }
  return ((T[d]>a) && (T[d]>b));
```

```
//A3;
boolean rech(int g,int d, int a, int b) {
int m= (g+d) /2;
   if (T[m]>=b) return rech(g,m-1,a,b);
   else if (T[m]<=a) return rech(m+1,d,a,b));
   else return true;
   //end if;}
rech(0,n-1, a,b);
```

```
//A4;
int m;
int g=0;
int d=n-1;
while (g<=d){
  m= (g+d) /2;
  if (T[m]>=b) d=m-1;
  else if (T[m]<=a) g=m+1;
  else return true;
}
 return false;
```

```
//A5;
int m;
int g=0;
int d=n-1;
  while (g<d) {
    m= (g+d) /2;
    if (T[m]>=b) d=m-1;
    else if (T[m]<= a) g=m+1;
    else return True;
  }
  return false;
```

Exercice 3 :

Soit un tableau trié par ordre croissant de n entiers distincts(qui peuvent être négatifs).

Proposer un algorithme en $O(\log n)$ qui retourne $Vrai$ Ssi il existe i tel que $T[i] = i$.

Exercice 4 : From "Algorithms", by S. Dasgupta, C. Papadimitriou, U. Vazirani

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and them combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two problems of size $n-1$ and them combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and them combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

Exercice 5 : Produit de matrices: Algorithme de Strassen

Soient M et N deux matrices carrées $n \times n$.

Q 1. Quelle est la complexité de l'algorithme "classique" de multiplication de deux matrices?

On suppose que n est une puissance de 2. On découpe les matrices M et N en 4 blocs $n/2 \times n/2$ comme suit:

$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, $N = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$

Et donc le produit est:

$MN = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$

Q 2. Déduire de ce qui précède un premier algorithme et calculer sa complexité.

Q 3. Soient maintenant $P_1 = A(F - H), P_2 = (A + B)H, P_3 = (C + D)E, P_4 = D(G - E), P_5 = (A + D)(E + H), P_6 = (B - D)(G + H), P_7 = (A - C)(E + F)$.

Q 3.1. Exprimer $AE + BG, AF + BH, CE + DG$ et $CF + DH$ avec les P_i en utilisant uniquement l'addition et la soustraction.

Q 3.2. En déduire un algorithme pour calculer le produit de matrices et évaluer sa complexité.

Exercice 6 : Défi de la semaine inspiré de Codechef: l'élément caché

Soit un tableau trié en ordre croissant de N entiers positifs. Chaque entier apparaît exactement K fois, sauf l'un d'entre eux qui apparaît entre 1 et $K - 1$ fois. L'objectif est d'identifier cet élément. La donnée est donc N et le tableau, mais K n'et pas connu. Proposer un algorithme en $O(\log N)$ permettant d'identifier l'élément recherché.

Par exemple, pour $N = 11$ et $1, 1, 1, 4, 4, 7, 7, 7, 9, 9, 9$, l'élément recherché est 4.

Pour $N = 17$ et $1, 1, 1, 1, 1, 4, 4, 4, 4, 4, 7, 7, 9, 9, 9, 9, 9$, l'élément recherché est 7. Pour $N = 51$ et $1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9$, l'élément recherché est 8.

Pour $N = 5$ et $1, 4, 4, 6, 6$, l'élément recherché est 1.