

UNITÉ IN3ST02

TD1 - Corrigé

EXERCICE 1

	Temps de calcul d'un algorithme	Complexité de cet algorithme
1-	$T(n=1) = a$ $T(n>1) \leq b + T(n-1)$	$O(n)$
2-	$T(n=1) = a$ $T(n>1) \geq b + n + T(n-1)$	$\Omega(n^2)$
3-	$T(n=1) = a$ $T(n>1) = b + 2 T(n-1)$	$\theta(2^n)$
4-	$T(n=2^{p=0}) = a$ $T(n=2^{p>0}) = b + T(n/2)$	$\theta(\log_2(n))$
5-	$T(n=2^{p=0}) = a$ $T(n=2^{p>0}) = b + 2 T(n/2)$	$\theta(n)$

EXERCICE 2

- Méthode 1 : détermination du temps unitaire et du nombre d'exécutions de chaque opération puis sommation.

		Temps unitaire	Nb d'exécutions (au pire)	Temps total (au pire)
<code>e ← t[0] ;</code> <code>i ← 1 ;</code>	<i>initialisations</i>	<i>a</i>	<i>1</i>	<i>a</i>
Tant que <code>i < n ∧ e > t[i]</code> faire	<i>test</i>	<i>b</i>	<i>n</i>	$\sum_{m=1}^n b$
<code> t[i-1] ← t[i] ;</code> <code> i++ ;</code> <code>finTantQue;</code>	<i>corps de boucle</i>	<i>c</i>	<i>n-1</i>	$\sum_{m=1}^{n-1} c$
<code>t[i-1] ← e ;</code>	<i>terminaison</i>	<i>d</i>	<i>1</i>	<i>d</i>

$$\Rightarrow T(n) \leq \alpha + \beta n = O(n)$$

- Méthode 2 : expression du temps d'exécution par une relation de récurrence.

$$T(n=1) = c_1$$

$$T(n>1) \leq c_1 + T(n-1)$$

$$\Rightarrow T(n) = O(n)$$

EXERCICE 3

```
// Pré-condition d'application : 0 < b ≤ 10
Fonction eval(EntierNat x, EntierNat b) : EntierNat
    si x < b alors retourner x ; finSi ;
    // sinon
    retourner x%10 + b*eval(x/10, b) ;
FinFonction
```

```
eval(127, 1) == 10
eval(127, 8) == 1×8² + 2×8 + 7
eval(127, 10) == 1×10² + 2×10 + 7
eval(127, b) == 1×b² + 2×b + 7
```

Réversivité non terminale (voir cours/poly).

L'espace mémoire nécessaire pour l'exécution de cet algorithme est proportionnel au nombre d'appels récursifs, c'est-à-dire au nombre de chiffres de x, c'est-à-dire à $1 + \max(0, \lfloor \log_{10}(x) \rfloor)$

En notant n le nombre de chiffres de x :

$$T(n=1) = a$$

$$T(n>1) = b + T(n-1)$$

$$\Rightarrow T(n) = \theta(n)$$

EXERCICE 4

Cas général $x > 0$

en notant n le nombre de chiffres de x

	Temps unitaire	Nb d'exécutions	Temps total
----- k ← 1 ; -----	a	1	a
Faire p ← 1 ; -----	b	n	b n
Pour i variant de 1 à k p ← p * 10 ; finPour ; -----	c	$\sum_{k=1}^n k$	$c \sum_{k=1}^n k$
k ← k + 1 ; tant que x / p > 0 ; -----	d	n	d n
retourner x / (p/10) ; -----	e	1	e

$$\Rightarrow T(n) = \alpha + \beta n + \delta n^2 = \theta(n^2) \quad \text{car} \quad \sum_{k=1}^n k = n(n+1)/2$$

Ne pas calculer les 10^k successifs en repartant à chaque fois de 1 mais à partir de la valeur 10^{k-1} calculée à l'itération précédente. L'algorithme résultant (ci-dessous) est en $\theta(n)$.

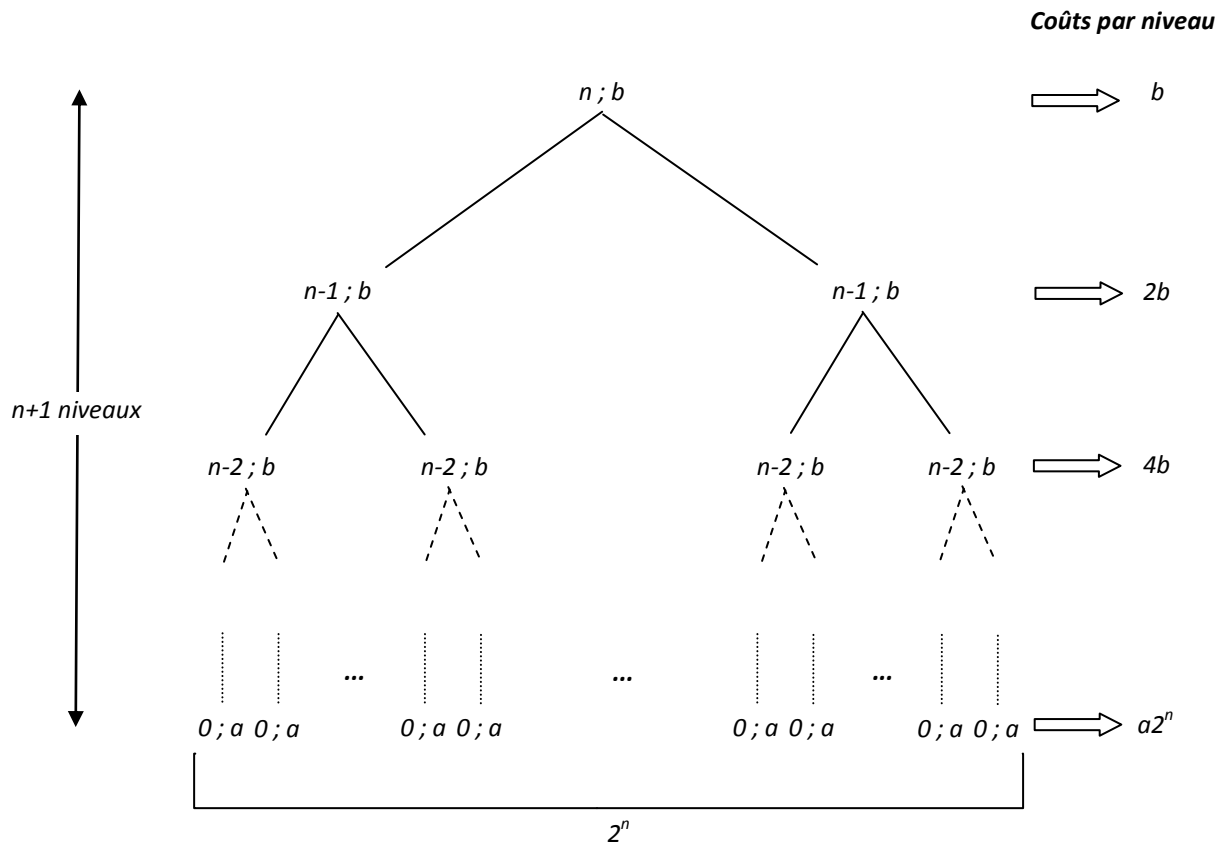
```

Fonction linear_msd(EntierNat x) : EntierNat
  Si x = 0 alors retourner 0 ; finSi ;
  // sinon
  p ← 10 ;
  Tant que x / p > 0 faire
    p ← p * 10 ;
  finTantQue ;
  retourner x / (p/10) ;
FinFonction
    
```

EXERCICE 5

- Méthode 1 : développement de la récursivité en un arbre dont les nœuds représentent les coûts à chaque étape puis sommation de ces coûts.

Le nombre d'appels récursifs ne dépend que de la valeur initiale n de la variable depth. Dans l'arbre des coûts ci-après, chaque nœud est étiqueté par un couple de valeurs : la valeur courante de la variable depth ; le temps de calcul de l'étape hors traitement de l'appel récursif.



$$\Rightarrow T(\text{depth}=n) = b \sum_{k=0}^{n-1} 2^k + a 2^n = \theta(2^n) \quad \text{car } \sum_{k=0}^{n-1} 2^k = 2^n - 1$$

- Méthode 2 : expression du temps de calcul par une relation de récurrence.

$$T(\text{depth}=0) = a$$

$$T(\text{depth}>0) = b + 2 T(\text{depth}-1)$$

$$\Rightarrow T(\text{depth}) = \theta(2^{\text{depth}})$$

EXERCICE 6

Tester si un tableau $t[0..n-1]$ est trié par valeurs croissantes se réalise en $O(n)$.

La boucle de l'algorithme de tri proposé s'exécute au pire $n!$ fois.

Le temps de calcul total de l'algorithme est donc :

$$T(n) = O(n + 2n \times n!) \approx O\left(n + 2n \sqrt{2\pi n} \left(\frac{n}{e}\right)^n\right)$$

En considérant qu'une opération élémentaire est opérée en 1 ns, il faudrait de l'ordre de 3000 ans pour trier un tableau de 20 éléments !!

EXERCICE 7

n doit être une puissance entière de 2 (n est divisé par 2 à chaque appel récursif et doit rester pair pour que la boucle « Pour tout k » traite bien tous les éléments de t).

$$T(n=2^{p=0}) = a$$

$$T(n=2^{p>0}) = b + c n + 2 T(n/2)$$

$$\Rightarrow T(n) = \theta(n \log(n))$$