

## Algo

- Numero: 3
- Prof: Alexandre Duret-Lutz
- Date: 23 Octobre 2017

### Simplification de l'écriture

$f(n) = \theta(n) \iff f(n)$  est (comparable a) une fonction lineaire.

On a donc:

$$\theta(n) + \theta(n) = \theta(n) \quad (1)$$

$$\theta(n) \times \theta(n) = \theta(n^2) \quad (2)$$

$f(n)$  est dominee par une fonction lineaire.

$f(n)$  domine une fonction  $\theta$  lineaire.

### Application sur SelectionSort

```
1 def SelectionSort(A, n):  
2     for i in range(n - 1):  
3         min = i  
4         for j in range(i + 1, n):  
5             if A[j] < A[min]:  
6                 min = j  
7         swap(A, min, i)  
8     return A
```

Voici la complexite en chaque ligne:

2.  $\theta(n)$
3.  $\theta(n)$
4.  $\theta(n^2)$
5.  $\theta(n^2)$
6.  $\theta(n^2)$
7.  $\theta(n)$
8.  $\theta(1)$  ou  $\theta(n)$  (Depend si on renvoi une copie ou un pointeur).

Finalement ca nous fait une complexite:  $\theta(n^2)$

## Application sur InsertSort

```

1 def InsertionSort(A,n):
2     for i in range(1, n):
3         key = A[i]
4         j = i - 1
5         while (j >= 0 && A[j] > key):
6             A[j + 1] = A[j]
7             j-=1
8         A[j + 1] = key
9     return A

```

Voici la complexite en chaque ligne:

2.  $\theta(n)$
3.  $\theta(n)$
4.  $\theta(n)$
5.  $\Omega(n) \leq ? \leq \theta(n^2)$
6.  $0 \leq ? \leq \theta(n^2)$
7.  $0 \leq ? \leq \theta(n^2)$
8.  $\theta(n)$
9.  $\theta(1)$

Finalement ca nous fait une complexite:

$$\theta(n) \leq T(n) \leq \theta(n^2) \quad (3)$$

De maniere plus formel:

$$T(n) \in \Omega(n) \cap \theta(n^2) \quad (4)$$

## Definitions Ensemblistes

$$\theta(g(n)) = \{f(n) | \exists n_0 \in \mathbb{N}, \exists c > 0, \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$$

Exemples:

$$3n + 3 \in \theta(n)$$

$$4 \in \theta(n)$$

$$2n^2 \notin \theta(n)$$

$$\Omega(g(n)) = \{f(n) | \exists n_0, \exists c > 0, \forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$$

Exemples:

$$n \in \Omega(n)$$

$$3n^2 + 2n + 7 \in \Omega(10000n)$$

$$42 \notin \Omega(n)$$

$$\sqrt{n} \notin \Omega(n)$$

$$\theta(g(n)) = \theta(g(n)) \cap \Omega(g(n)) = \{f(n) | \exists n_0 \in \mathbb{N}, \exists c_1 > 0, \exists c_2 > 0, \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0 \implies f(n) = \theta(g(n))$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \implies f(n) = o(g(n))$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty \implies g(n) = o(f(n)) \text{ and } f(n) = w(g(n))$$

$$f(n) = \theta(g(n)) \iff g(n) = \Omega(f(n)) \quad (5)$$

## Tri fusion (MergeSort)

Principe:

On a un tableau que l'on coupe en deux et on tri recursivement les deux parties. Il faut ensuite fusionner les deux tableaux triés.

```
1 def MergeSort(A, b, e):
2     if e - b <= 1:
3         return
4     m = b + (e - b) // 2
5     MergeSort(A, b, m)
6     MergeSort(A, m, e)
7     Merge(A, b, m, e)
```

```
1 def Merge(A, b, m, e):
2     i = b, j = m, k = b
3     for k in range(b, e):
4         if (j >= e or (i < m and A[i] <= A[j])):
5             B[k] = A[i]
6             i += 1
7         else:
8             B[k] = A[j]
9             j += 1
10    A = B[:] // Copy B into A
```

Calculons la complexite de `Merge`:

On pose  $n = e - b$ .  $T_n(n)$  = complexite temporelle de `Merge`.

2.  $\theta(1)$
3.  $\theta(n)$
4.  $\theta(n)$
5.  $\theta(n)$
6.  $\theta(n)$
7.  $\theta(n)$
8.  $\theta(n)$
9.  $\theta(n)$
10.  $\theta(n)$

Donc `Merge` est de complexite  $\theta(n)$

---

Pour la fonction `MergeSort` celle ci est recursive il faut donc determine la complexite dans le cas d'arret et le ca courant:

- Le cas d'arret:  $T_{MS}(n) = \theta(1)$
- Le cas courant:  $T_{MS}(n) = T_{MS}(\lceil \frac{n}{2} \rceil) + T_{MS}(\lfloor \frac{n}{2} \rfloor) + \theta(n)$

Pour resoudre ca on va simplifier:

$$T(n) = \begin{cases} \theta(1) & \text{if } n \text{ is } \leq 1 \\ 2T(\frac{n}{2}) & \text{otherwise} \end{cases}$$

Calcul par substitutions:

$$T(n) = 2T(\frac{n}{2}) + cn$$

$$T(n) = 2(2T(\frac{n}{4}) + c\frac{n}{2}) + cn$$

$$T(n) = 4T(\frac{n}{4}) + 2cn/2 + cn$$

$$T(n) = 8T(\frac{n}{8}) + 4cn/4 + 2cn/2 + cn$$

Apres  $i - 1$  substitutions:

$$T(n) = 2T(\frac{n}{2}) + icn$$

On arrete les substitutions lorsque  $\frac{n}{2} = 1$

C'est a dire  $i = \log_2(n)$  alors  $T(\frac{n}{2^{\log_2(n)}}) = T(n) = \theta(n)$

### Differentes classes de complexites

$n$	$n$	$n \log n$	$n^2$	$2^n$
$10^1$	3.3ns	11ns	33ns	0.3ms
$10^2$				$1.3 \times 10^{13}$ ans
$10^3$				
$10^4$				
$10^5$		0.5ns	3.3s	
$10^6$		6.6ns	5.5min	
$10^7$		77ns	9.3h	
$10^8$		0.9s	28j	
$10^9$		10s	10ans	
$10^{10}$	3.3s	1.8min	57ans	

On a le droit a 10 soumissions max sinon on perd des poitns de maniere exp:

Sujet des tps ici