# TP1 - Sous-séquences maximales

Sébastien Delecraz     Éloi Perdereau

10 octobre 2013

**Résumé**

Ce rapport propose une étude de différents algorithmes pour résoudre le problème du calcul de la sous-séquence de somme maximale dans un tableau d'entier. Quatre algorithmes de complexité différentes sont présentés ici. On trouvera donc pour chacun leur pseudo-code et une étude de leur complexité. Enfin nous confronterons les différents résultats obtenus et l'analyse théorique réalisée.

# 1 Algorithme naïf

## 1.1 Pseudo-code

**Algorithme 1 :** Naïf

**Entrées** : T, n
**Sorties** : Sous-séquence maximale
**début**
  $S_{max} \leftarrow -\infty$
  **pour tous les** $1 \leq k \leq n$ **faire**
    **pour tous les** $k \leq l \leq n$ **faire**
      $S \leftarrow 0$
      **pour tous les** $k \leq j \leq l$ **faire**
        $S \leftarrow S + T[j]$
      **fin**
      **si** $S > S_{max}$ **alors**
        $S_{max} \leftarrow S$
      **fin**
    **fin**
  **fin**
  **retourner** $S_{max}$
**fin**

## 1.2 Complexité

L'algorithme se compose de trois boucles de parcours du tableau imbriquées. Sa complexité est donc :

$$C(n) = n * n * n = \theta(n^3)$$

## 2 Algorithme moins naïf

### 2.1 Pseudo-code

**Algorithme 2 :** Moins naïf

**Entrées** : T, n
**Sorties** : Sous-séquence maximale
**début**
    $S_{max} \leftarrow -\infty$
    **pour tous les** $1 \leq k \leq n$ **faire**
        $S \leftarrow 0$
        **pour tous les** $k \leq l \leq n$ **faire**
            $S \leftarrow S + T[l]$
            **si** $S > S_{max}$ **alors**
                $S_{max} \leftarrow S$
            **fin**
        **fin**
    **fin**
    **retourner** $S_{max}$
**fin**

### 2.2 Complexité

L'algorithme se compose de deux boucles de parcours du tableau imbriquées. Sa complexité est donc :

$$C(n) = n * n = \theta(n^2)$$

## 3 Algorithme diviser pour régner

### 3.1 Pseudo-code

**Algorithme 3 :** Diviser pour régner

**Entrées** : T, k, l
**Sorties** : Sous-séquence maximale
**début**
  **si** $l - k = 1$ **alors retourner** $T[k]$;
  **si** $l - k = 2$ **alors retourner** $max\{T[k], T[k+1], T[k]+T[k+1]\}$;
  $j \leftarrow \frac{l-k}{2}$
  $S_1 \leftarrow Diviser\_pour\_regner(T, k, j)$
  $S_2 \leftarrow Diviser\_pour\_regner(T, j+1, l)$
  $S_3 \leftarrow S_{tmp} \leftarrow T[j]$
  **pour** *i variant de j-1 à k descendant* **faire**
    $S_{tmp} \leftarrow S_{tmp} + T[i]$
    **si** $S_{tmp} > S_3$ **alors**
      $S_3 \leftarrow S_{tmp}$
    **fin**
  **fin**
  $S_4 \leftarrow S_{tmp} \leftarrow T[j]$
  **pour** *i variant de j+1 à l-1 montant* **faire**
    $S_{tmp} \leftarrow S_{tmp} + T[i]$
    **si** $S_{tmp} > S_4$ **alors**
      $S_4 \leftarrow S_{tmp}$
    **fin**
  **fin**
  $S_0 \leftarrow S_3 + S_4 - T[j]$
  **retourner** $max\{S_0, S_1, S_2\}$
**fin**

### 3.2 Complexité

Cet algorithme applique la méthode diviser pour régner. On peut écrire son équation de récurrence :

$$\begin{aligned} C(n) &= 2 * C(n/2) + 2 * (n/2) \\ &= 2 * C(n/2) + \theta(n) \end{aligned}$$

On applique ensuite le Master Theorem :
$c = \log_2 2 = 1$
$f(n) = \theta(n)$
On est dans le cas 2 car $f(n) = \theta(n^c)$. Donc la complexité totale de l'algorithme est :

$$C(n) = \theta(n \log n)$$

# 4 Algorithme incrémental

## 4.1 Pseudo-code

**Algorithme 4 :** Incrémental

```
Entrées : T, n
Sorties : Sous-séquence maximale
début
    S_max ← T[1]
    S_1 ← 0
    S_2 ← 0
    maj ← false
    pour tous les 2 ≤ i ≤ n faire
        S_1 ← S_1 + T[i]
        S_2 ← S_2 + T[i]
        si T[i] < 0 alors
            si T[i] > S_max alors
                S_max ← T[i]
                S_1 ← 0
                S_2 ← 0
            fin
            si S_2 < 0 alors
                S_2 ← 0
            fin
        fin
        sinon
            S_tmp ← S_max
            si S_1 + S_tmp > S_tmp alors
                S_max ← S_max + S_1
                maj ← true
            fin
            si S_2 > S_tmp&&S_2 > S_1 + S_tmp alors
                S_max ← S_2
                maj ← true
            fin
            si maj == true alors
                S_1 ← 0
                S_2 ← 0
            fin
        fin
    fin
fin
```

## 4.2 Complexité

L'algorithme se compose de deux boucles de parcours du tableau imbriquées. Sa complexité est donc :

$$C(n) = n = \theta(n)$$

# 5 Résulats expérimentaux

Le graphique ce dessous nous montre pour chaque algorithme leur temps d'execution en fonction de la taille des instances données en entrée.

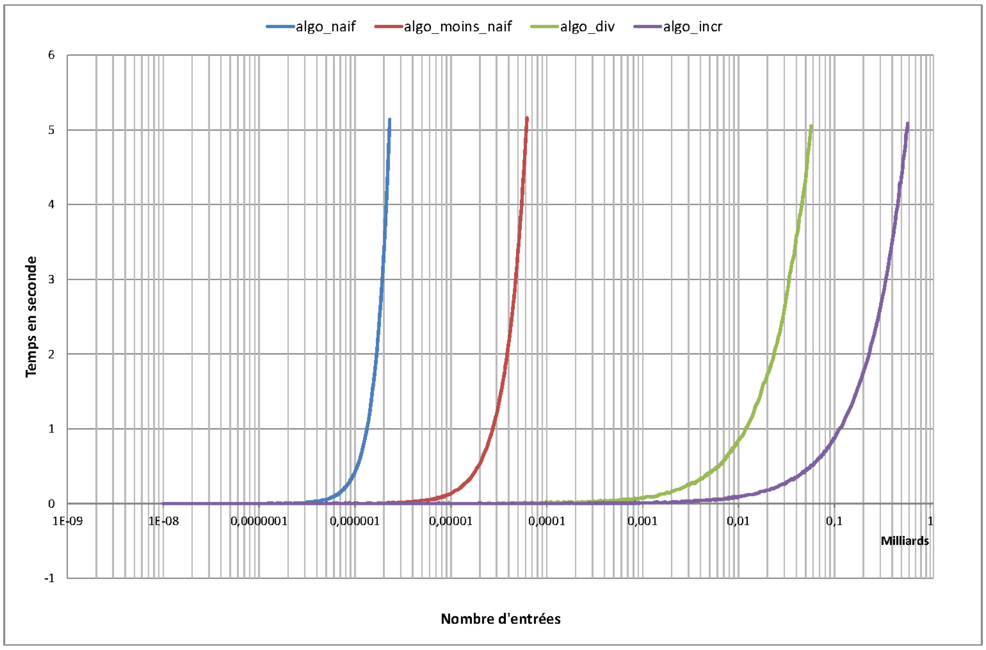


FIGURE 1 – Évolution des perfomances des algorithmes