

TP1 - Sous-séquences maximales

Sébastien Delecraz

Éloi Perdereau

7 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 Analyse théorique

1.1 Algorithme naïf

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

Complexité $C(n) = n * n * n = O(n^3)$

1.2 Algorithme moins naïf

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

Complexité $C(n) = n * n = O(n^2)$

1.3 Algorithme diviser pour régner

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  $\max\{T[k], T[k + 1], T[k] + T[k + 1]\}$ ;  
     $j \leftarrow \frac{l-k}{2}$   
     $S_1 \leftarrow \text{Diviser\_pour\_regner}(T, k, j)$   
     $S_2 \leftarrow \text{Diviser\_pour\_regner}(T, j + 1, l)$   
     $S_3 \leftarrow S_{tmp} \leftarrow T[j]$   
    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

Complexité $C(n) = 2 * C(n/2) + 2 * \frac{n}{2} = \sum_{i=0}^{\log n} 2^i * \frac{n}{2^i} = O(n \log n)$

1.4 Algorithme incrémental

Algorithme 4 : Incrémental

Entrées : T, n

Sorties : Sous-séquence maximale

début

$S_{max} \leftarrow T[1]$

$S_1 \leftarrow 0$

$S_2 \leftarrow 0$

$maj \leftarrow false$

pour tous les $2 \leq i \leq n$ **faire**

$S_1 \leftarrow S_1 + T[i]$

$S_2 \leftarrow S_2 + T[i]$

si $T[i] < 0$ **alors**

si $T[i] > S_{max}$ **alors**

$S_{max} \leftarrow T[i]$

$S_1 \leftarrow 0$

$S_2 \leftarrow 0$

fin

si $S_2 < 0$ **alors**

$S_2 \leftarrow 0$

fin

fin

sinon

$S_{tmp} \leftarrow S_{max}$

si $S_1 + S_{tmp} > S_{tmp}$ **alors**

$S_{max} \leftarrow S_{max} + S_1$

$maj \leftarrow true$

fin

si $S_2 > S_{tmp}$ **alors**

$S_{max} \leftarrow S_2$

$maj \leftarrow true$

fin

si $maj == true$ **alors**

$S_1 \leftarrow 0$

$S_2 \leftarrow 0$

fin

fin

fin

 retourner S_{max}

fin

Complexité $C(n) = n = O(n)$

2 Résultats expérimentaux

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

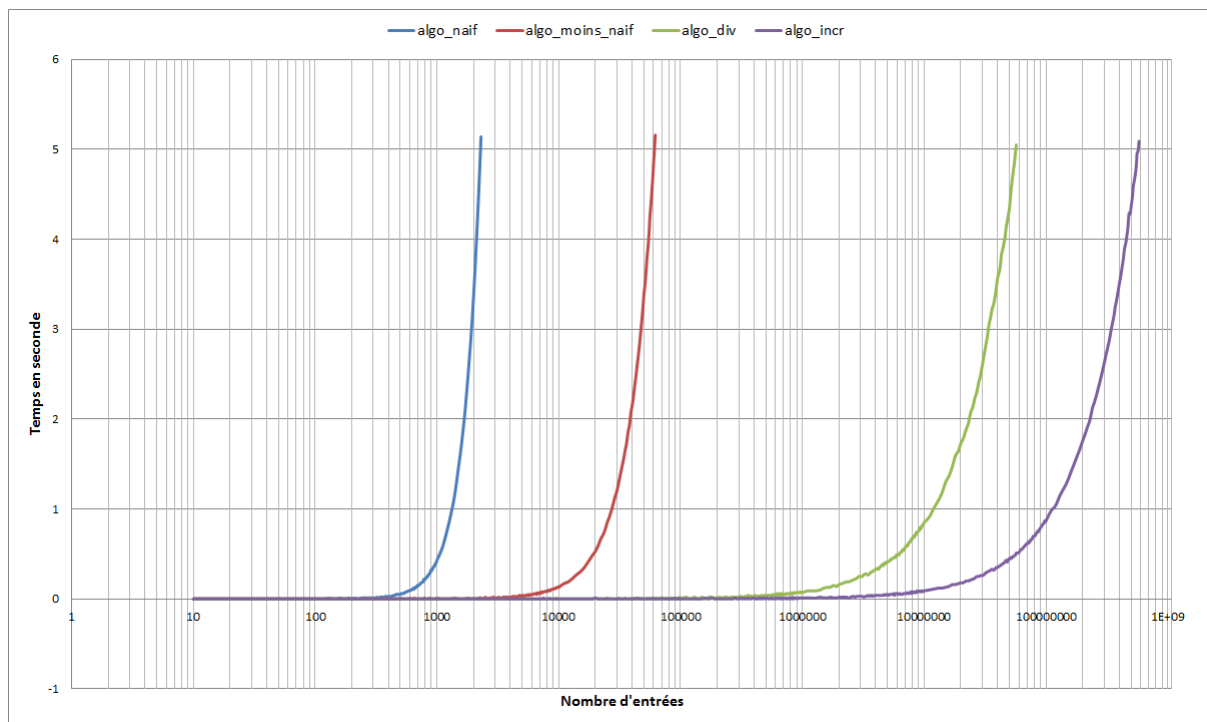


FIGURE 1 – Évolution des performances des algorithmes