
TP2 - Question 2

réalisé par Claire Bouttes - 536793110
IFT-3001

Table des matières

Introduction	2
1 Algorithme de partitionnage	3
2 Algorithme de tri	3
3 Algorithme appelé	4
Conclusion	4

Introduction

Pour cette question, nous avons un tableau A de n résultats séparés en w pages de k éléments. Le but de l'algorithme qu'il est demandé de designer est qu'il trie les éléments de la p ième page tout en restant suffisamment efficace.

On peut déjà observer que le tri ne s'effectue pas uniquement parmi des nombres. Au vu des données à notre disposition, il me semble intéressant de sélectionner le tri rapide. La particularité qu'on rajoutera un filtre empêchant de continuer la récurrence pour les éléments non inclus dans le résultat attendu.

L'intérêt de cet algorithme est de s'intéresser au maximum aux k éléments à retourner.

La fonction `RetournePage` fera donc un appel au `QuickSort` et retournera l'ensemble des k éléments dans la page demandée p . `QuickSort` faisant lui-même appel à la fonction de partition. Nous analyserons premièrement cette dernière fonction avant d'analyser le `QuickSort` et pour enfin analyser `RetournePage`.

Algorithm 1: Partition($A[l..r]$)

```
 $s \leftarrow \text{Uniforme}(l, r);$   
 $\text{interchange}A[l] \text{ et } A[s];$   
 $j \leftarrow l;$   
 $p \leftarrow A[j];$   
for  $i = j + 1..r$  do  
    if  $A[i] < p$  then  
         $j \leftarrow j + 1;$   
         $\text{interchange}A[i] \text{ et } A[j];$   
    end  
end  
 $\text{interchange}A[j] \text{ et } A[l];$   
return  $j;$ 
```

Algorithm 2: QuickSort($A[l..r]$, d , k)

```
if  $r > l$  then
     $s \leftarrow PartitionRand(A[l..r]);$ 
    if  $s > d$  then
        |  $QuickSort(A[l..s - 1]);$ 
    end
    if  $s < d + k$  then
        |  $QuickSort(A[s + 1..r]);$ 
    end
end
```

1 Algorithme de partitionnage

Il est intéressant d'indiquer que la fonction d'interchange s'effectue en temps constant et peut donc être considérée comme opération élémentaire. L'opération de base choisie pour cet algorithme de partitionnage est la condition $A[i] < p$ effectuée à chaque tour de boucle.

L'exécution de l'opération de base dépend uniquement de n , la taille de l'instance. Il n'y a donc pas de meilleur ou pire cas.

$$C(n) = \sum_{i=l+1}^r 1$$
$$C(n) = l - 1 - r + 1 \quad C(n) = l - r$$

or $l - r$ correspond à notre taille d'instance $n - 1$. Ainsi on a :

$$C(n) = n - 1$$

2 Algorithme de tri

On rappellera que $C_{partition}(n) = n$ L'exécution de la récurrence dépend uniquement de la taille de l'instance.

$$C(A[1..n - 1]) = \frac{1}{n} \left[\sum_{s=1}^d (C(A[s + 1..n - 1])) + \sum_{s=d}^{d+k} (C(A[1..s - 1]) + C(A[s + 1..n - 1])) + \sum_{s=d+k}^{n-1} (C(A[1..s - 1])) \right] + C_{partition}(n)$$

$$C(A[1..n - 1]) = \frac{1}{n} \left[\sum_{s=1}^n C(s) + \sum_{s=0}^k C(s) \right] + (n - 1)$$

$$C(n) = \frac{1}{n} \left[\sum_{s=1}^{n+k} C(s) \right] + (n - 1)$$

$$C(n - 1) = \frac{1}{n-1} \left[\sum_{s=1}^{n+k-1} C(s) \right] + (n - 2)$$

$$C(n - 2) = \frac{1}{n-2} \left[\sum_{s=1}^{n+k-2} C(s) \right] + (n - 3)$$

Après x substitutions, on a alors :

$$C(n - x) = \frac{1}{n-x} \left[\sum_{s=1}^{n+k-x} C(s) \right] + (n - 1 - x)$$

Après plusieurs essais, je n'ai pas réussi à refaire le même raisonnement vu en classe, et à l'appliquer pour cette analyse. Pour autant je sens que nous sommes en $k \log(k) + n$. Étant donné qu'on élimine théoriquement à chaque itération les parties qui ne sont pas concernées. En revanche je ne peux être sûr.

En revanche, on sait que $C(n) < n \log(n)$. On restera donc sur $C(n) \in \omega(n \log(n))$

3 Algorithme appelé

Algorithm 3: `RetournePage(R[0..n-1],k,p)`

```

 $d \leftarrow k \times p;$ 
 $A \leftarrow R // S'effectueen\theta(n);$ 
 $QuickSort(A[0..n-1], d, k);$ 
return  $A[d..d+k] // S'effectueen\theta(k);$ 

```

On a alors :

$$C(n) = \omega(n \log(n)) + \omega(n) + \omega(k)$$

Selon les règles du maximum, on a alors

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

Conclusion

En conclusion, nous n'arrivons pas à la conclusion attendue qui était un algorithme en $\omega(n, k \log(k))$. Il faudrait donc pouvoir réussir à continuer le calcul de la seconde partie pour pouvoir prouver si oui ou non l'algorithme est assez rapide.