TD 4

Complexité d'algorithmes classiques

Exercice 1 1. Simuler sur le tableau suivant l'algorithme de tri à bulles :

8 4 6 1 5 3 2 7.

On peut constater que les éléments les plus grands se trouvant dans la partie basse du tableau (dits *lièvres*) atteignent assez rapidement leur position définitive, tandis que les éléments les plus petits se trouvant dans la partie haute du tableau (dits *tortues*) atteignent leur place plus lentement.

2. On inverse maintenant le sens de parcours de l'algorithme, autrement dit, on balaye le tableau de la droite vers la gauche en échangeant $T[i]$ avec $T[i-1]$ si $T[i] < T[i-1]$. Simuler cette variante du tri à bulles sur le même tableau. Qu'arrive-t-il aux lièvres et aux tortues ?
3. On en déduit un algorithme amélioré pour le tri à bulles : on alterne les passes, en balayant une fois le tableau de la gauche vers la droite et la fois successive de la droite vers la gauche. Ecrire un algorithme de tri basé sur ce principe.
4. Justifier sa terminaison grâce à un invariant de boucle. Donner la complexité de cet algorithme.

Exercice 2 Considérons l'algorithme suivant de tri d'un tableau $T[i..j]$:

```
Tri(T,i,j)
   Si i >= j alors
     sortir
   k := i
   pivot := T[i]

   Pour l de i+1 à j faire
    Si T[l] <= pivot alors
    T[k] := T[l]
    T[l] := T[k+1]
    T[k+1] := p
    k++

   Tri(T,i,k-1)
   Tri(T,k+1,j)
```

1. Exécuter cet algorithme sur le tableau $T = [5, 9, 1, 6, 3, 2, 4]$ avec $i = 1$ et $j = 7$
2. Quel est sa complexité dans le pire cas ?
3. Donner un des pires cas.

Cours : ce tri fonctionne bien sur la plupart des entrées. On peut d'ailleurs montrer que sa complexité *moyenne* est $O(n\log(n))$.

Exercice 3 Soit A et B deux matrices $n \times n$, le produit de ces deux matrices $C = A * B$ est défini par $C_{i,j} = \sum_{0 \leq \alpha \leq n-1} A_{i,\alpha} B_{\alpha,j}$.

1. Ecrire un programme qui effectue le produit de deux matrices.
2. Donner la complexité de votre algorithme.
3. On veut calculer la puissance n d'une matrice A. Proposer un algorithme qui calcule A^n en utilisant votre procédure de produit.
4. Améliorer cet algorithme en vous inspirant de l'exponentiation rapide des nombres. Quel est alors sa complexité ?

Exercice 4 La suite de Fibonacci est définie par la relation de récurrence : $F(n+1) = F(n) + F(n-1)$ et $F(0) = F(1) = 1$.

1. Proposer un algorithme récursif pour résoudre ce problème
2. Montrer par récurrence que sa complexité est un $\Omega(\frac{3}{2}^n)$.
3. Proposer un algorithme itératif qui calcule les valeurs de F en prtant de 0 pour aller jusqu'à $F(n)$
4. Evaluer sa complexité.
5. Est il possible d'écrire un algorithme récursif aussi performant que l'algorithme itératif ? Indice : calculer les couples $(F(i), F(i+1))$ plutôt que $F(i)$
6. La relation matricielle suivante est une définition de Fibonacci

$$\begin{pmatrix} F(i+1) \\ F(i) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F(i) \\ F(i-1) \end{pmatrix}$$

   Utiliser l'exponentiation rapide de matrice pour obtenir un algorithme de calcul du $n^{\text{ème}}$ terme de la suite de Fibonacci.
7. Quel est sa complexité ?

Exercice 5 Soit T un tableau trié d'entiers (positifs et négatifs) tous distincts. On sait que ce tableau est circulairement triée dans l'ordre croissant, c'est-à-dire qu'il existe un indice i tel que en commençant la lecture à $T[i]$ jusqu'a $T[n]$ et recommençant à $T[1]$ jusqu'à $T[i-1]$ on obtient une suite croissante. Par exemple, la suite $[15, 17, 20, 2, 3, 5, 8, 10]$ est circulairement triée, et l'indice de départ à considerer est $i = 4$.

1. Donner un algorithme en temps linéaire qui permette de trouver l'indice i du plus petit élément du tableau.
2. Proposez un invariant qui aide à montrer que votre algorithme est correct.
3. Expliquer pourquoi a monotonie d'un sous tableau peut se tester en comparant ses bornes. De plus, quand on coupe en deux le tableau T, si on obtient une seule moitié monotone alors la solution recherchée se trouve dans l'autre moitié. Verifiez cette propriété sur le tableau donné en exemple.
4. Donner un algorithme inspiré de la recherche dichotomique qui trouve l'indice i du minimum en temps logarithmique.

Exercice 6 On considère un tableau A contenant des entiers signés (positifs et négatifs). On se propose de rechercher un sous-tableau de A (i.e. un ensemble d'éléments consécutifs de A) ayant la somme maximale.

1. Ecrire une procédure qui si on lui donne i et j renvoie la somme des éléments du sous-tableau de borne i et j.
2. Ecrire un algorithme qui éxecute la procédure précédente pour tout les couples (i, j) afin de trouver la somme maximale.
3. Donner la complexité de votre algorithme
4. Améliorer cet algorithme naïf en essayant de ne pas refaire tous les calculs. Indice : utiliser le résultat de (i, j) pour calculer $(i, j + 1)$
5. Donner la complexité de cette amélioration
6. Proposer un algorithme linéaire. Indice : prétraiter votre tableau en remplacant les séquences de nombres positifs et négatifs par leur somme.