SÉRIE 4

Question # 1

A) Écrivez un algorithme diviser pour régner qui trouve le plus grand et le plus petit élément d'un tableau de n nombres.

B) Trouvez la récurrence qui décrit le nombre de comparaisons entre deux éléments du tableau fait par votre algorithme (on suppose ici que $n = 2^k$). Donnez l'ordre de croissance de cette récurrence.

C) Comparez votre algorithme avec l'algorithme de force brute qui résout le même problème.

Question # 2

A) Écrivez un algorithme diviser pour régner qui calcul la valeur de a^n où $a > 0$ et $n \geq 0$.

B) Trouvez et résolvez la récurrence qui décrit le nombre de multiplications fait par votre algorithme (on suppose ici que $n = 2^k$).

C) Comparez votre algorithme avec l'algorithme de force brute qui résout le même problème.

Question # 3

Des joueurs de Chess-Boxing se réunissent pour un tournoi. Chaque partie est constituée d'exactement deux joueurs et d'un unique gagnant. Le but est de faire le moins de parties possibles pour déterminer un unique grand gagnant.

A) Écrivez un algorithme diviser pour régner qui détermine le grand gagnant du tournoi de Chess-Boxing. Vous pouvez faire appel à l'algorithme ChessBoxing(joueur1, joueur2) qui retourne le gagnant parmi une paire de joueurs en $\Theta(1)$.

B) Analysez l'efficacité de votre algorithme en A).

C) Allez voir le pseudo-code de la solution en A). Quelle est l'efficacité de cet algorithme si les tableaux en paramètres sont passés par copie. Quelle est l'efficacité s'ils sont passés par référence ?

Question # 4

Lorsque nous avons parlé de la conception et de l'analyse des algorithmes au chapitre 2, nous avons mentionné que la base d'un logarithme n'est pas importante dans la majorité des contextes qui découlent de l'analyse d'algorithmes. Est-ce vrai pour les deux parties du théorème général qui inclus des logarithmes ?

Question # 5

Trouvez l'ordre de croissance de la fonction $T(n)$ qui est exprimée selon la récurrence (utilisez le théorème général pour résoudre la récurrence) :

A) $T(n) = T(n/2) + c, T(1) = 0$

B) $T(n) = 4T(n/2) + n, T(1) = 1$

C) $T(n) = 4T(n/2) + n^2, T(1) = 1$

D) $T(n) = 4T(n/2) + n^3, T(1) = 1$

Question # 6

Soit l'algorithme Quicksort($A[l \ldots r]$) présenté à la figure 1 (voir p. 128 de A. Levitin) :

```
ALGORITHME QuickSort(A[l..r])
//Entrée: le sous tableau A[l..r] de A[0..n-1]
//Sortie: le sous tableau A[l..r]  trié
if l < r
   s ← Partition(A[l..r])
   QuickSort(A[l..s-1])
   QuickSort(A[s+1..r])
```

FIGURE 1 – Tri rapide

A) Est-ce que les tableaux formés d'éléments tous égaux doivent être considérés comme des instances : du pire cas, du meilleur cas, ou aucun des deux ?

B) Est-ce que les tableaux triés en ordre décroissant doivent être considérés comme des instances : du pire cas, du meilleur cas, ou aucun des deux ?

Question # 7

A) En considérant Quicksort avec la technique de sélection du pivot "median-of-three" (qui consiste à choisir comme pivot l'élément médian parmi l'élément le plus à gauche, celui le plus à droite et celui au centre), est-ce que les tableaux triés en ordre croissant doivent être considérés comme des instances : du pire cas, du meilleur cas, ou aucun des deux ?

B) Répondez à la même question qu'en A mais pour des tableaux triés en ordre décroissant.

***Question # 8**

On vous donne une collection de n boulons de tailles différentes ainsi que les n écrous correspondants. Il est permis d'essayer un boulon avec un écrou dans le but de savoir si l'écrou est trop large pour le boulon, pas assez large pour le boulon ou exactement de la bonne taille pour le boulon. Cependant, il n'est pas possible de comparer deux boulons ensemble ni deux écrous ensemble. Le problème est de trouver, pour chaque boulon, l'écrou qui lui convient. Décrivez votre algorithme en français, puis en pseudo-code. Votre algorithme doit avoir une efficacité moyenne de $\Theta(n \log n)$.

Question # 9

Écrivez un algorithme qui est une version récursive de la recherche binaire présentée à la figure 2 (voir l'algorithme BinarySearch($A[0 \ldots n-1], K$) à la p. 134 de A. Levitin).

```
ALGORITHM BinarySearch(A[0..n-1], K)
  l ← 0; r ← n-1
  while l ≤ r do
    m ← ⌊(l + r)/2⌋
    if K = A[m] then return m
    else if K < A[m] then r ← m – 1
    else l ← m + 1
  return -1
```

FIGURE 2 – Recherche binaire