Algorithmique TD2
L3 Informatique – ENS Cachan

28 juillet 2020

Exercice 1

Soit T un tableau de n cellules contenant des valeurs numériques et $k \leq n$ un entier.

1. Ecrire un algorithme naïf qui renvoie la $k$ième plus petite valeur du tableau T et l'indice correspondant dans le tableau.
2. Ecrire un algorithme alternatif avec tri.
3. Comparer la complexité des deux algorithmes. Indiquez les cas favorables à l'un ou l'autre des algorithmes.
4. Soit l'algorithme 1. Démontrer que cet algorithme renvoie la $k$ième plus petite valeur du tableau T. Quel type de partition rend l'algorithme le plus efficace ?
5. En tenant compte de la question précédente, on propose l'algorithme 2 qui est une adaptation du précédent. Démontrer que n_1 et n_3 sont tous les deux inférieurs à $3n/4$.
6. Soit $Time(n)$, le pire temps d'exécution de cet algorithme pour un tableau de taille inférieure ou égale à n. Démontrer que pour une constante c bien choisie :

$$\forall n < 24, \quad Time(n) \leq c \cdot n$$

$$\forall n \geq 24, \quad Time(n) \leq c \cdot n + Time(\frac{n}{5}) + Time(\frac{3n}{4})$$

En déduire la complexité de cet algorithme.

Exercice 2

1. Donner la complexité en temps de l'algorithme naïf pour multiplier deux entiers, donnés en binaire, de n et m bits respectivement.
2. En observant que

$$(a \times 2^k + b)(c \times 2^k + d) = ac \times 2^{2k} + (ac + bd - (a - b)(c - d)) \times 2^k + bd$$

proposer un algorithme de type « diviser pour régner » et calculer sa complexité en temps.

Exercice 3

1. Rappeler le principe de l'algorithme de tri rapide avec pivot.
2. Écrire formellement cet algorithme.
3. Analyser sa complexité dans le pire des cas.
4. En utilisant l'Exercice 1, proposer un algorithme de tri rapide avec pivot dont la complexité dans le pire cas est $n \log(n)$ où n est la taille du tableau à trier.

```
Select(T : int array,
       n : int, k : int)
  n_1 <- 0; n_2 <-0; n_3 <- 0;
  T_1 <- Array.make n 0; T_3 <- Array.make n 0;
  // On sélectionne une valeur du tableau
  valeur <- T[1];
  for i = 1 to n do
    if T[i] < valeur then
      n_1 <- n_1+1;
      T_1[n_1] <- T[i];
    else if T[i] = valeur then
      n_2 <- n_2+1;
    else
      n_3 <- n_3+1;
      T_3[n_3] <- T[i];
  done
  // On recherche l'élément dans un des tableaux par un appel récursif ou
  // on renvoie valeur suivant les valeurs de n_1, n_2 et n_3
  if n_1 >= k then
   return Select(T_1,n_1,k);
  else if n_1 + n_2 >= k then
   return valeur;
  else
    return Select(T_3,n_3, k - (n_1 + n_2 ) );
```

FIGURE 1 – Un algorithme par partition

```
Select(T : int array,
       n : int, k : int)
R <- Array.make n 0; S <- Array.make n 0;
T_1 <- Array.make n 0; T_3 <- Array.make n 0;
mediane <- 0; m <- 0; n_1 <- 0; n_2 <- 0; n_3 <- 0; i <- 0; j <- 0;
if n < 24 then
  // On applique l'algorithme de la question 3
  S <- Trie(T,1,n);
  return S[k];
else
  // On divise T en paquets de 5 éléments et
  // on range dans R, l'élément médian de chaque paquet
  m <- floor(n/5);
  for i = 0 to m -1 do
    for j = 1 to 5 do
      S[j] <- T[5*i+j]};
    done;
    S <- Trie(S,1,5);
    R[i+1] <- S[3];
  done
  // On calcule le médian des médians par un appel récursif
  mediane <- Select(R,m, ceil(m/2));
  for i = 1 to n do
    if T[i] < mediane then
      n_1 <- n_1+1;
      T_1[n_1] <- T[i];
    else if T[i] = mediane then
      n_2 <- n_2+1;
    else
      n_3 <- n_3+1;
      T_3[n_3] <- T[i];
  done;
  // On recherche l'élément dans un des tableaux par un appel récursif ou
  // on renvoie mediane suivant les valeurs de n_1, n_2 et n_3
  if n_1 >= k then
    return Select(T_1,n_1,k);
  else if n_1 + n_2 >= k then
    return mediane;
  else
    return Select(T_3,n_3, k - (n_1 + n_2) );
```

FIGURE 2 – Algorithme par partition optimisé