Durée : 01h

T E S T

Mercredi 22 Novembre 2023

ALGORITHMIQUES ET COMPLEXITE

Exercice n°1 : (4.5pts)

Sort the following functions in ascending order of asymptotic growth rate:

Trier les fonctions suivantes par ordre croissant de l'ordre de grandeur asymptotique :

$f1(n) = 5^n$ $\quad f2(n) = n^{1/3}$ $\quad f3(n) = n^2 log_2 n$

$f4(n) = n^{log_2 n}$ $\quad f5(n) = n^3$ $\quad f6(n) = 4^{log_2 n}$

$f7(n) = n^2\sqrt{n}$ $\quad f8(n) = 2^{2n}$ $\quad f9(n) = \sqrt{log_2 n}$

Exercice n°2 : (9.5pts)

Consider the following algorithm that takes an array of numbers A of size n:

Considérons l'algorithme suivant qui prend en entrée un tableau de nombres A de taille n :

```
for (i=2; i<n; i++)
  for (j=1; j<=i-1; j++)
    for (k=0; k<=j-1; k++) {
      x = A[i]; y = A[j]; z = A[k];
      if (x > y) swap(x,y);
      if (y > z) { swap(y,z);
                   if (x > y) swap(x,y); }
      if (y-x == z-y) return true; }
return false;
```

Analyze the complexity of this algorithm and write another algorithm that does the same but with a strictly better asymptotic time complexity and with the same space complexity.

Analysez la complexité de cet algorithme et écrivez un autre algorithme qui fait la même chose mais avec une complexité temporelle asymptotique strictement meilleure et avec la même complexité spatiale.

Exercice n°3 : (6pts)

The partition function is executed on each iteration of the quick sort.

La fonction partition est exécutée à chaque itération du tri rapide.

a/ Give the array obtained after executing this function on the following array. We consider that the chosen pivot is the last element of the array. Explain the result obtained by giving the intermediate steps (give the contents of the array after each swapping).

a/ Donner le tableau obtenu après exécution de cette fonction sur le tableau suivant. On considère que le pivot choisi est le dernier élément du tableau. Expliquer le résultat obtenu en donnant les étapes intermédiaires (donner le contenu du tableau après chaque permutation).

150	147	192	178	176	17	160	136	159	130	150	143

b/ What would be the complexity (in term of comparisons and swapping) if we apply this sorting to an array already sorted by taking the last element of the array as a pivot? Justify.

b/ Quelle serait la complexité (en termes de comparaisons et permutations) si on appliquait ce tri sur un tableau déjà trié en prenant comme pivot le dernier élément du tableau ? Justifier.

Bon courage…

Solution & Barème

Exercice n°1 : **(4.5pts: 0.5pt for each well classified function)**

Sort the following functions in ascending order of asymptotic growth rate:

1	$f9(n) = \sqrt{log_2 n}$
2	$f2(n) = n^{1/3}$
3	$f6(n) = 4^{log_2 n} = n^2$
4	$f3(n) = n^2 log_2 n$
5	$f7(n) = n^2\sqrt{n}$
6	$f5(n) = n^3$
7	$f4(n) = n^{log_2 n}$
8	$f8(n) = 2^{2n} = 4^n$
9	$f1(n) = 5^n$

Exercice n°2 : **(9.5pts)**

Consider the following algorithm that takes an array of numbers A of size n:

Analyze the complexity of this algorithm and write another algorithm that does the same but with a strictly better asymptotic time complexity and with the same space complexity.

```
for (i=2; i<n; i++).....n-2=O(n)
  for (j=1; j<=i-1; j++).....1+2+…+(n-2)=(n-2)(n-1)/2=O(n²)
    for (k=0; k<=j-1; k++) {.....1+(1+2)+(1+2+3)+…(1+2+…+(n-2))=O(n³)*
      x = A[i]; y = A[j]; z = A[k];
      if (x > y) swap(x,y);
      if (y > z) { swap(y,z);
                   if (x > y) swap(x,y); }
      if (y-x == z-y) return true; }
return false;
```

$$* \sum_{i=1}^{n-2}\sum_{j=1}^{i-2} j = \sum_{i=1}^{n-2} \frac{(i-2)(i-1)}{2} = \frac{1}{2}\left[\sum_{i=1}^{n-2} i^2 - 3\sum_{i=1}^{n-2} i\right] + \sum_{i=1}^{n-2} 1$$

$$= 1/2\left[\frac{(n-2)(n-1)(2(n-2)+1)}{6} - 3\frac{(n-2)(n-1)}{2}\right] + (n-2) = O(n^3)$$

Boucle1 : 0.5pt
Boucle2 : 0.75pt
Boucle3 : 1.5pts
Résultat final : 0.75pt

Une idée :
Trier le tableau (en $O(n^2)$ ou O(nlogn), puis générer les paires ($O(n^2)$) au lieu des triplets, et pour chaque paire faire une recherche dichotomique (O(logn)) des 3 éléments suivants :

max(A[i],A[j]) + abs(A[i]-A[j])
min(A[i],A[j]) - abs(A[i]-A[j])
min(A[i],A[j]) + (abs(A[i]-A[j])/2)

Ce qui donne un algorithme de $O(n^2 \log n)$

```
sort(A,n);
for (int i=0; i<n-1; i++) {
  for (int j=i+1; j<n; j++) {
   if (binarySearch(A,n,max(A[i],A[j])+diff(A[i],A[j])))   return true;
   if (binarySearch(A,n,min(A[i],A[j])-diff(A[i],A[j])))   return true;
   if (binarySearch(A,n,min(A[i],A[j])+diff(A[i],A[j])/2)) return true;
}
}
return false;
```

Algorithme : 5pts (Principe seul : 50%)
(Tri 2pts, génération de paires 1pt, recherche dichotomique 2pts)
Complexité : 1pt

Exercice n°3 : (6pts)

The partition function is executed on each iteration of the quick sort.

a/ Give the array obtained after executing this function on the following array. We consider that the chosen pivot is the last element of the array. Explain the result obtained by giving the intermediate steps (give the contents of the array after each swapping).

150	147	192	178	176	17	160	136	159	130	150	143

Etapes : (3pts : 0.75pt pour chaque tableau ou permutation**)**

130	147	192	178	176	17	160	136	159	150	150	143
130	136	192	178	176	17	160	147	159	150	150	143
130	136	17	178	176	192	160	147	159	150	150	143
130	136	17	143	176	192	160	147	159	150	150	178

b/ What would be the complexity (in term of comparisons and swapping) if we apply this sorting to an array already sorted by taking the last element of the array as a pivot? Justify. **(3pts :** 1 réponse + 2pts justification**)**

$O(N^2)$,
Même si le dernier élément est déjà à sa place, la fonction partition fait N itérations pour décider de sa place (N comparaisons et 0 permutations). Puis le tri continue avec les (N-1) éléments restants, et ainsi de suite ce qui donne $N + (N-1) + \ldots + 1 = N(N+1)/2 = O(N^2)$ comparaisons et 0 permutations.