POLYTECH PARIS-SUD

Et2-Info

Algorithmique

Travaux Dirigés N° 3

(Correction)

Les algorithmes de ce TD sont à écrire en pseudocode.

1 Factorielle

Écrivez l'algorithme récursif permettant de calculer la factorielle d'un entier naturel n passé en paramètre. La définition mathématique de la fonction factorielle est :

$$\begin{cases} n! = \prod_{i=1}^{n} i = 1 \times 2 \times 3 \cdots (n-1) \times n \text{ pour } n \geq 1 \\ 0! = 1 \end{cases}$$

Correction :
Algorithme CalculerFactorielle(n : entier naturel)

Début :
    Si n = 0 :
        Retourner 1
    Sinon :
        Retourner n × CalculerFactorielle(n − 1)
    Fin Si
Fin

---

2 Fibonacci

De la même manière que pour l'exercice précédent, écrivez l'algorithme récursif permettant de calculer la suite de Fibonacci pour un entier naturel n passé en paramètre. La définition mathématique de cette suite est :

$$\begin{cases} \mathcal{F}_0 = 0 \\ \mathcal{F}_1 = 1 \\ \mathcal{F}_{n+2} = \mathcal{F}_{n+1} + \mathcal{F}_n \end{cases}$$

Écrivez maintenant une version itérative de cet algorithme. Calculez la complexité en temps de ces deux algorithmes, lequel est le plus performant ?

Correction : *Algorithme CalculerFibonacciRecursif(n : entier naturel)*
Début :
    Si n ≤ 1 :
        Retourner n
    Sinon :
        Retourner CalculerFibonacci(n − 1) + CalculerFibonacciRecursif (n − 2)
    Fin si

Fin

Algorithme CalculerFibonacciIteratif(n : entier naturel)
Début :
    Variables moinsdeux, moinsun, résultat, i, : entiers
    moinsdeux ← 0
    moinsun ← 1
    Pour i ← 3 à n faire :
        résultat ← moinsdeux + moinsun
        moinsdeux ← moinsun
        moinsun ← résultat
    Fin pour
    Retourner résultat
Fin

La complexité est en $\mathcal{O}(n)$ pour la version itérative et $\mathcal{O}(2^n)$ pour la version récursive.

---

3 Tri des tableaux

On considère trois algorithmes de tri par ordre croissant : le tri par sélection, le tri par insertion et le tri bulle. Décrivez le comportement et la complexité de ces algorithmes dans les deux cas les plus extrêmes. Dans le cas le plus favorable, le tableau est déjà trié dans l'ordre croissant et dans le pire cas, il est trié dans l'ordre décroissant.

Correction :
- Le tri sélection fait autant de comparaisons dans tous les cas. La seule différence tient au nombre de mises à jour du minimum. Ce tri a cependant l'avantage de ne faire que peu d'échanges, ce qui peut être important pour des données de grande taille. Dans le pire cas ou en moyenne, la complexité (ici : nombre de comparaisons) du tri par sélection est en $O(n^2)$.

- Le tri par insertion : une complexité de $O(n^2)$ dans le pire cas et en moyenne, et linéaire $O(n)$ dans le meilleur cas (ne fait qu'un test par tour de boucle si le tableau est trié). Avantage pour les tableaux presque triés.

- Pour le tri bulle, le comportement dépend du fait que l'on ait ou pas modifié l'algorithme pour qu'il s'aperçoive qu'un tableau est trié (ce qui est le cas si aucune permutation n'a été faite au cours d'une passe). La complexité est de $O(n^2)$ dans le pire cas et en moyenne, et linéaire $O(n)$ dans le meilleur cas.

---

4 Le plus grand commun diviseur (PGCD)

Étant donné deux entiers n et m sans connaître leur factorisation,

1) Écrivez l'algorithme itératif permettant de calculer le PGCD(n,m) et donnez la complexité de cet algorithme.
2) Écrivez l'algorithme récursif permettant de calculer le PGCD(n,m)
Indice : PGCD(n, 0) = n et PGCD(n, m) = PGCD(m, n mod m).

Correction : *Algorithme PGCD(n :entier, m : entier) :*
Début :
    Si m==0
        Retourner n

Sinon
Retourner PGCD(m,n mod m)
Fin Si
Fin

---

3) On sait que si l'algorithme récursif du dessus est terminé en N étapes avec une paire d'entiers $n > m > 0$, on a $n \geq \mathcal{F}_{N+2}$ et $m \geq \mathcal{F}_{N+1}$ (Théorème Lamé). Donnez la complexité de cet algorithme récursif.
Indice : $\mathcal{F}_N \geq (\frac{1+\sqrt{5}}{2})^{N-2}$, $\log_{10} \frac{1+\sqrt{5}}{2} > 1/5$

Correction : $m \geq \mathcal{F}_{N+1} \geq (\frac{1+\sqrt{5}}{2})^{N-1} \Rightarrow N-1 \leq \log_{\frac{1+\sqrt{5}}{2}} m \Rightarrow (N-1)/5 < \log_{10} \frac{1+\sqrt{5}}{2} \times \log_{\frac{1+\sqrt{5}}{2}} m \Rightarrow N \leq 5 \log_{10} m$.
La complexité : $O(\log_{10} m)$

---

5 Complexité de code

```
1) for( int i = n; i > 0; i /= 2 ) {
   for( int j = 1; j < n; j *= 2 ) {
        for( int k = 0; k < n; k += 2 ) {
               ... // le nombre constant d'operations
        }
    }
}

2) for ( i=1; i < n; i *= 2 ) {
    for ( j = n; j > 0; j /= 2 ) {
       for ( k = j; k < n; k += 2 ) {
             sum += (i + j * k );
       }
     }
}

3) for( int i = n; i > 0; i-- ) {
     for( int j = 1; j < n; j *= 2 ) {
        for( int k = 0; k < j; k++ ) {
          ... // le nombre constant d'operations
        }
      }
}
```

Correction : *1. In the outer for-loop, the variable i keeps halving so it goes round log2n times. For each i, next loop goes round also log2n times, because of doubling the variable j. The innermost loop by k goes round n^2 times. Loops are nested, so the bounds may be multiplied to give that the algorithm is $O(n(logn)^2)$. 2. Running time of the inner, middle, and outer loop is proportional to n, logn, and logn, respectively. Thus the overall Big-Oh complexity is $O(n(logn)2)$.*

3. The outer for-loop goes round n times. For each i, the next loop goes round $m = log2n$ times, because of doubling the variable j. For each j, the innermost loop by k goes round j times, so that the two inner loops together go round $1 + 2 + 4 + +2m - 1 = 2m - 1 = n$ times. Loops are nested, so the bounds may be multiplied to give that the algorithm is $O(n^2)$

---

6 Temps d'éxecution

Les trois algorithmes A, B, et C ont la complexité de temps $O(n^2)$, $O(n^{1.5})$, et $O(n \log n)$. n est le nombre de données. En testant leurs codes, chaque algorithme prend 10 secondes pour processer 100 données. Calculez le temps que chaque algorithme prend pour processer 10,000 données.