KOUDID
AZEDDINE
212131044860
GRP1

MULTIPLICATION DE DEUX MATRICE

1/ Donner un algorithme itératif calculant le produit de deux matrices représentées sous forme d’un tableau à deux dimensions. Calculer la complexité de cet algorithme.

```
Fonction MatriceMul(A, B, n) {

   Pour (i=0; i<n; i++;){  // Boucle externe : parcourt chaque ligne de la matrice A

        Pour (j=0; j<n; j++){ // Boucle intermédiaire : parcourt chaque colonne de la matrice B

                C[i][j] = 0 // Initialiser C[i][j] à 0 ⇒ C cest la matrice résultant

                Pour (k=0; k<n; k++){ // Boucle interne : effectue le produit scalaire entre la
        ligne i de A et la colonne j de B
                    C[i][j] = C[i][j] + A[i][k] * B[k][j]
                }
            }
        }
   return C;
   }
```

La complexité de cette algorithme:
on a 3 boucle imbriquée tout les 3 sont indépendant donc on peut avoir la complexité par fair une sommation: Σ (i=0 au n-1) Σ (j=0 au n-1) Σ (k=0 au n-1) == $O(n^3)$

a/ Doit-on préciser dans quels cas (pire cas, meilleur des cas, cas moyen) cette complexité est obtenue ?

NON, la complexité $O(n^3)$ est la meme dans les 3 cas car chaque élément de la matrice résultante nécessite toujours de parcourir toutes les lignes de A et toutes les colonnes de B, indépendamment des valeurs spécifiques des éléments des matrices.

b/ Modifier l’algorithme précédent lorsque la matrice A est de dimension (m,n) et la matrice B de dimension (n, p). Quelle est alors la complexité de l’algorithme ?

```
Fonction MatriceMul(A, B, m, n, p) {

    Pour (i=0; i<m;i++;){// Parcourir les lignes de A (A est une matrice de dimension m * n )

     Pour (j=0; j<p; j++;){ // Parcourir les colonnes de B (B est matrice de dimension n * p )

            C[i][j] = 0  // Initialiser C[i][j] à 0 ⇒ C cest la matrice résultant de dimension m * p

            Pour (k=0; k<n; k++;) {   // Calculer le produit
                C[i][j] = C[i][j] + A[i][k] * B[k][j]
            }
        }
    }
       return C;
}
```

La complexité de cette algorithme:
ici on a 3 boucle imbriquée chacun indépendante de l'autre donc on peut fair une sommation pour trouver la complexité

la première boucle le i varie de 0 jusqu'à m-1   cela veut dire il ya m iteration
la deuxième boucle le j varie de 0 jusqu'à p-1 cela veut dire il ya p iteration
la troisième boucle le k varie de 0 jusqu'à n-1 cela veut dire il ya n iteration

on fai la somme des troi boucle Σ (i=0 au m-1) Σ (j=0 au p-1) Σ (k=0 au n-1) == O(m*p*n)

2/ Donner un algorithme récursif calculant le produit de deux matrices représentées sous forme d'un tableau à deux dimensions. Calculer la complexité de cet algorithme.

```
Fonction MatriceMulRecursiv(A, B, C, i, j, k, n){
    // Si l'indice i (ligne de A) dépasse la taille de la matrice (n), on termine la récursion.
    Si i >= n Alors
        return;  // Sortie de la fonction récursive (fin de l'opération)
    fsi
  // Si l'indice j (colonne de B) dépasse la taille de la matrice (n) on passe à la ligne suivante de A.
Si j >= n Alors
  MatriceMulRecursiv(A, B, C, i + 1, 0, 0, n);// Avancer à la ligne suivante de A réinitialiser j, k
        return;  // Sortie de la fonction récursive pour passer à la ligne suivante
    fsi;

    // Si k est inférieur à n, on effectue le produit entre la ligne i de A et la colonne j de B.
    Si k < n Alors
        C[i][j] = C[i][j] + A[i][k] * B[k][j]  // Calcul du produit
        MatriceMulRecursiv(A, B, C, i, j, k + 1, n);  //  passer au terme suivant
        return;              // Sortie de la fonction récursive pour avancer dans le produit scalaire
```

fsi;

MatriceMulRecursiv(A, B, C, i, j + 1, 0, n); // Avancer à la colonne suivante de B, réinitialiser k
}

La complexité de cette algorithme:
on a le nombre total d'appels récursifs dépend du nombre d'éléments de C et du nombre d'opérations nécessaires pour calculer chaque élément et a chaque étape la fonction parcourt les lignes de A et les colonnes de B ce qui signifie qu'il effectue un appel récursif pour chaque élément de la matrice C[i][j]. En d'autres termes il ya n^2 appels récursifs pour traiter tous les éléments de C en plus pour chaque élément de C (C[i][j]) on a n operation de d'addition et de multiplication donc le nombre total T(n)= $n * n^2 = n^3$ donc la complexité reste O(n^3)

3/ Supposons que les matrices A et B peuvent être décomposées en sous matrices carrées de dimension n/2 * n/2
Proposer un algorithme récursif (ou itératif) pour le produit matriciel en vous basant sur la décomposition précitée. Calculer la complexité de cet algorithme

```
Fonction MatriceMulRecursiv_V2(A, B, C, n, i, j, k){
   // Cas de base : lorsque la taille de la sous-matrice est 1
   SI n = 1 ALORS
      C[i][j] = C[i][j] + A[i][k] * B[k][j]  // Calcul du produit scalaire pour un seul élément
   SINON
      m = n / 2  // Taille des sous-matrices (diviser la matrice par 2)


      // Première sous-matrice de A et B
      MatriceMulRecursiv_V2(A, B, C, m, i, j, k)
      // Deuxième sous-matrice de A et B
      MatriceMulRecursiv_V2(A, B, C, m, i, j, k + m)
      // Première sous-matrice de A et B avec indices modifiés
      MatriceMulRecursiv_V2(A, B, C, m, i + m, j, k)
      // Deuxième sous-matrice de A et B avec indices modifiés
      MatriceMulRecursiv_V2(A, B, C, m, i + m, j, k + m)
   fsi;
}
```

La complexité de cette algorithme:
on a chaque appelle recursive on divise le la taille de la matrice en 2 comme on a 4 appelle alors on fait la division 4 fois donc le premier terme de l'expression de la récursivité T(n)= 4*T(n/2) en plus de sa on a avant de diviser le problemme la fonction vas fair un calcule de multiplication et addition des element de A et B le coût de cette étape est de O(n), car chaque multiplication scalaire (produit ligne-colonne) pour une ligne de A et une colonne de B nécessite n multiplications donc l'expression de recurrence est $T(n)=4*T(\frac{n}{2})+O(n)$

Nous allons supposer que la solution de la récurrence est de la forme T(n)=O(n^k), ou k est un exposant que nous allons déterminer.on remplace dans la récurrence **:**

Substituons T(n)=O(n^k)dans la récurrence pour voir ce qui se passe.

T(n)=4*T($\frac{n}{2}$)+O(n) ⇒ T(n) = 4*(O($\frac{n}{2}$)^k) +O(n)
Cela devient :
T(n)=O(n^k/2^k)+O(n) ⇒ on O(n^k/2^k) devient O(n^k) donc
T(n)=O(n^k)+O(n)

maintenant on cherche a dominer O(n) par O(n^k) et cela est valable pour chaque valeur de k=>1 on prendre k=2 pour satisfaire les choose et donc on a la complexite de l'algorithme est noter bien que dans le cas ou le k=1 les deux termes sont de la même forme, donc l'algorithme a une complexité O(n log(n))
mais dans notre cas ona prendre k=2 : et la complexiter est :
O(n^2)