Complexité

Fiche TD – Extraits d'exams

1. Carrelage (examen 2008-2009)

On aimerait carreler une surface de dimension NxN carreaux en utilisant D différentes sortes de carreaux (bleu, rouge, etc). Il faut 1 seconde pour placer un carreau.

1. Combien y a-t-il de combinaisons possibles de carrelage ?
2. Combien de temps faut-il pour essayer toutes les combinaisons de carrelage ?
3. Pour D=N. Quelle est la complexité et la classe de complexité du programme suivant : générer tous les carrelages de couleur uniforme.

2. Recherche dans un tableau

Déterminer la complexité dans le meilleur cas, le pire cas et en moyenne de l'algorithme suivant, qui permet de rechercher une valeur x dans un tableau t.

```
trouvé=faux
i=1
tant que (trouvé = faux) et (i<=n)
        si t[i] = x
                trouvé = vrai
        Fsi
        i=i+1
FTantque
```

3. Produit scalaire sur R^n

```
Algorithme
P=0
Pour i de 1à n
        P=P+x[i]*y[i]
FPour
```

4. Multiplication de deux matrices carrées de taille n

C = A*B ;

```
Algorithme
Pour i=1 à n
        Pour j=1 à n
                C[i][j] = 0 ;
                Pour k=1à n
                        C[i][j]=C[i][j]+A[i][k]*B[k][j];
                FPour
        FPour
FPour
```

5. Calcul de puissance

Donnez la complexité des algorithmes de calcul de puissance suivant (on considère que $n=2^k$)

```
A1 :
y=1
pour i=1 à n faire
        y=y*x
FPour
```

```
A2 : puiss(x,n)
Si (n=1)
        renvoyer x
Sinon
        renvoyer y*puiss(x,n-1)
FSi
```

```
A3 : puiss(x,n)
Si (n=1)
        renvoyer x
Sinon
        tmp=puiss(x,n/2)
        renvoyer tmp*tmp
```

6. La dichotomie (examen 2006-2007)

Principe : on cherche à deviner une valeur comprise entre 1 et n en faisant un minimum de propositions. Pour chaque proposition la réponse peut être : « c'est la bonne réponse », « la valeur cherchée est supérieure à la valeur proposée » ou « la valeur cherchée est inférieure à la valeur proposée ».
La meilleure stratégie pour trouver la solution est de proposer la valeur n/2 en première valeur puis si la valeur cherchée est supérieure (resp. inférieure) de proposer la valeur située au milieu de l'intervalle [n/2,n] (resp. [1,n/2]) et ainsi de suite jusqu'à trouver la bonne valeur.
Calculer la complexité de cet algorithme en justifiant votre réponse.

7. Sudoku (examen 2006-2007)

Principe : un sudoku est une grille de 9 par 9 cases, rempli de chiffres de 1 à 9. Chaque ligne doit contenir tous les chiffres, chaque colonne doit contenir tous les chiffres et si on décompose la grille de 9 par 9 en 9 carrés de 3 par 3, chaque carré doit contenir les neuf chiffres.

1. Essayez d'estimer le nombre de grilles possibles d'un sudoku (en majorant par exemple ce nombre)
2. Quelle est la complexité du problème du sudoku ?
3. On généralise le sudoku à des grilles de n^2 par n^2 cases décomposées en n^2 carrés de taille n par n. Les nombres présents dans les cases vont de 1 à n.
4. Voici une approche pour résoudre une telle grille de sudoku

Approche par back-tracking

On parcourt les cases libres de haut en bas et de gauche à droite.
Pour chaque case, on essaye les nombres de 1 à n^2 dans l'ordre.
Si le chiffre proposé génère un conflit avec un autre chiffre, on passe au chiffre suivant.
Si pour une case donnée tous les chiffres ont été testés et génèrent un conflit, on retourne à la case précédente et on change le chiffre.
On procède ainsi jusqu'à ce qu'une solution soit trouvée

5. Qu'elle est la complexité de cet algorithme? Faites bien attention que la complexité que vous calculez soit la complexité dans le pire cas et que ce pire cas soit bien un cas qui puisse apparaître. On suppose que le nombre de cases déjà remplies est de $n^4/6$.

8. Calcul de l'inverse d'une matrice (examen 2007-2008)

Calcul directe par la méthode de Gauss : $A*A^{-1} = I$ avec A de taille nxn, I matrice Identité (des 1 sur la diagonale, des zéros ailleurs). On résous n systèmes d'équations linéaires du type Ax=b

```
Algorithme de résolution d'un système d'équations linéaires Ax = b

Pour i = 1 à n-1
  pour j = i+1 à n
    a[i,j] = a[i,j] / a[i,i]
  fin pour
  b[i] = b[i] / a[i,i]
  pour k = i+1 à n
```

```
    pour j = i+1 à n
      a[k,i] = a[k,i] - a[k,j]*a[i,i]
    fin pour
    b[k] = b[k] - a[k,i]*b[i]
  fin pour
fin pour
b[n] = b[n] / a[n,n]
```

Questions :
1. Quelle est la complexité de l'algorithme de résolution de Ax=b ?
2. Quelle est la complexité de l'algorithme d'inversion de matrice ?

Calcul par décomposition LU

On résous également n systèmes d'équations linéaires du type Ax=b avec la méthode présentée ci-après mais en décomposant (une seule fois) A en un produit L.U

```
Décomposition LU
Pour i=1 à n
     Pour j=1 à n
          u[i,j] = a[i,j]
     fin pour
fin pour
Pour k=1 à n
    Pour i=k+1 à n
    Pour j=k+1 à n
        u[i,j] = u[i,j] – u[k,j]* u[i,k]/ u[k,k]
    fin pour
    l[i,k]= u[i,k]/ u[k,k]
  fin pour
fin pour


Résolution de Ax = b, en posant A = LU, LUx =b, y= Ux, résolution de Ly = b
puis Ux=y
y[1]=b[1]
Pour i=2 à n
   s=b[i]
   Pour j=1 à i
       s = s-l[i,j]*y[j]
   Fin pour
   y[i]=s
fin pour
b[n]=y[n]/u[n,n]
Pour i=n-1 à 1
  s=y[i]
  Pour j=i+1 à n
     S = s – u[i,j]*b[j]
     b[i]=s/u[i,i]
  fin pour
fin pour
```

Questions
1. Quelle est la complexité de la décomposition LU ?
2. Quelle est la complexité de la résolution de Ax=b ?
3. Quelle est la complexité de l'algorithme d'inversion par la méthode LU

4. A quelle classe de complexité appartient l'algorithme ?

9. Calcul de tan(x) (examen 2007-2008)

Enoncé : on utilise le principe suivant pour calculer tan(x) : $\tan(x) = 2t/(1-t^2)$ avec $t=\tan(x/2)$ et $\tan(x) = x$ pour x petit

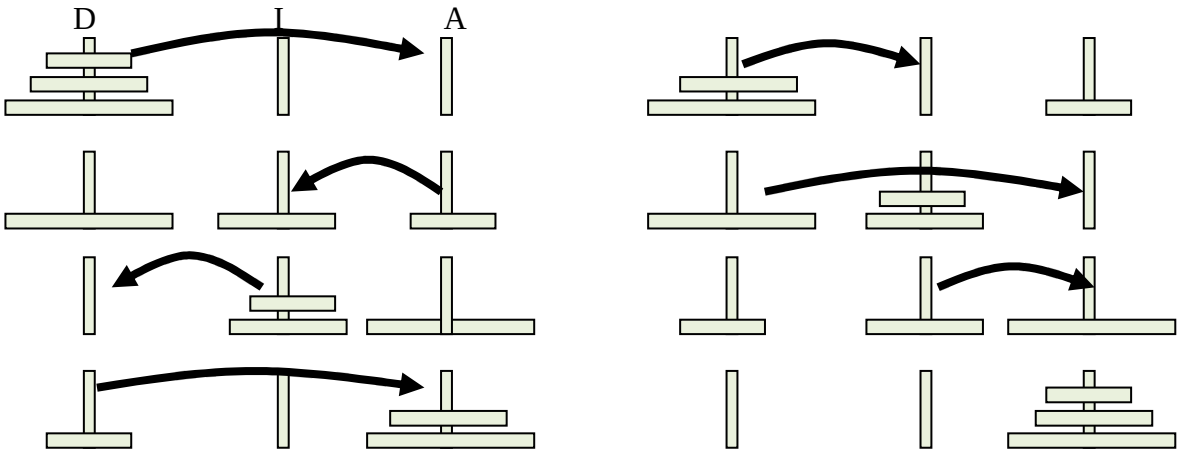
```
Algorithme
Function Res=tan(x)
      si (x < 2^-n)
            Res = x
      Sinon
            t=tan(x/2)
            Res = 2*t / (1-t²)
      Fsi
      Renvoyer Res
Fin
```

Questions
1. Calculer la complexité de cet algorithme en fonction de n

10. Les tours de Hanoï (examen 2007-2008)

Le problème des tours de Hanoï est posé comme suit :
On souhaite déplacer des disques de diamètres différents d'une tour de départ (D) à une tour d'arrivée (A) en passant par une tour intermédiaire (I). Les règles suivantes doivent être respectées : on ne peut déplacer qu'un disque à la fois, on ne peut placer un disque que sur un disque plus grand ou sur un emplacement vide.

Exemple avec 3 disques



Voici l'algorithme

Pour résoudre le problème avec n+1 disques, il suffit de déplacer une tour de taille n de « D » vers « I », puis un disque de « D » vers « A », puis une tour de taille n de « I » vers « D ».

```
Hanoi(n, D, I, A)
      Si n = = 1
            On déplace le disque de « D » vers « A »
      Sinon
            Hanoi(n-1, D, A,I)
            On déplace un disque de « D » vers « A »
            Hanoi(n-1, I, D, A)
      Fin
```

Calculez la complexité du problème des tours de Hanoï. A quelle classe de complexité l'algorithme appartient il ?

11. Ordre de complexité

Classez les complexités suivantes par ordre croissant : n^3, log(n), nlog(n), n!, e^n, n^n.

12. Fibonacci

```
La suite de Fibonacci est définie de la manière suivante :
F(n) = F(n-1) + F(n-2), avec F(0) = 0, F(1) = 1 et F(-1) = 1
```

1. Calculer la complexité de l'algorithme naïf suivant

```
fibo(n):
        if (n <= 1) # cas de base
              return n    # si n=0 return 0, si n=1 return 1
        else         # récurrence
              return fibo(n - 1) + fibo(n - 2)
```

2. Calculer la complexité de l'algorithme suivant

```
fibo(n)
 f_n_1 = 1  # F_{-1} = 1
 f_n = 0    # F_0 = 0
 for i in n
     f_n_1 = f_n
     f_n = f_n + f_n_1
 return f_n
```

13. Comparaison d'algorithmes (examen 2008-2009)

1. Calculer en justifiant la complexité des deux fonctions suivantes

Rec1(n) = Si n=1 renvoyer(1) sinon renvoyer (2*Rec1(n-1))
Rec2(n) = Si n=1 renvoyer(1) sinon renvoyer (Rec2(n-1) + Rec2(n-1))

2. Calculer en justifiant la complexité de la fonction suivante

test(n)
s=0
tant que (n>=1) faire
Si (n est pair) alors n=n/2
Sinon n=n-1
s:=s+1
FTq
Renvoyer(s)

14. Boulons et écrou

Vous disposez de n boulons et n écrous. Chaque boulon correspond à un seul écrou. Vous pouvez déterminer si un boulon est plus petit, plus grand ou de la même taille qu'un écrou mais vous ne pouvez pas comparer deux écrous ensembles ou deux boulons ensembles.
Donnez un algorithme qui permette d'associer tous les boulons et les écrous.
Calculer sa complexité (pire cas).
Si vous pouvez comparer les boulons et les écrous entre eux, trouvez un nouvel algorithme et donner sa complexité.

15. Ordres de complexité

1. Soit 2 programmes A et B qui utilisent 2 algorithmes différents pour résoudre le même problème, à savoir trier une liste de N objets.

Le programme A a besoin d'un temps t = 10 000 * N pour trier la liste.
Le programme B a besoin d'un temps t = 2 * N^2 pour trier la même liste.

a. Quel programme est plus rapide pour trier une liste de 5 objets?
b. Quel est la longueur critique N_c telle que pour tout $N > N_c$, le programme A est plus rapide que le B?
c. Comparer les performances des programmes A et B avec un troisième programme C qui trie la liste en un temps t = 10 + 0.5 * N^3.

2. Soit 2 programmes E et F qui utilisent 2 algorithmes différents pour trier une liste de N objets.

Le programme E a besoin d'un temps t = 8^N.
Le programme F a besoin d'un temps t = N!.
Quel programme est plus rapide? Discuter les cas où N = 2,4,8,10,12

16. Le problème du sac à dos

On cherche à remplir un sac-à-dos avec un certain nombre d'objets de façon à le remplir exactement. On peut modéliser ce problème de la façon suivante: on se donne *n* entiers strictement positifs a_i et un entier *S*. Existe-t-il des nombres $x_i \in \{0, 1\}$ tels que

$$S = x_0\, a_0 + x_1\, a_1 + \cdots + x_{n-1}\, a_{n-1}?$$

Si x_i vaut 1, c'est que l'on doit prendre l'objet a_i, et on ne le prend pas si x_i=0.

1. **Version 1 : donner la complexité de l'algorithme suivant**

```
   // a[0..n[ : existe-t-il x[] tel que
   // somme(a[i]*x[i], i=0..n-1) = S ?
 static void sacADosn(int[] a, int S){
     int n = a.length, N;
     int[] x = new int[n];

     for(int i = 0; i < 2^n; i++){
         // reconstruction de N = somme x[i]*a[i]
         N = 0;
         for(int j = 0; j < n; j++)
             if(x[j] == 1)
                 N += a[j];
         if(N == S){
             System.out.print("S="+S+"=");
             for(int j = 0; j < n; j++)
                 if(x[j] == 1)
                     System.out.print("+"+a[j]);
             System.out.println("");
         }
         // simulation de l'addition
         for(int j = 0; j < n; j++){
             if(x[j] == 1)
                 x[j] = 0;
             else{
                 x[j] = 1;
                 break; // on a fini
             }
         }
     }
 }
```

2. **Version recursive : donner la complexité de l'algorithme suivant**

```
// on a déjà calculé Si = sum(a[j]*x[j], j=0..i-1)
    static void sacADosrec(int[] a, int S, int[] x,
```

```
                            int Si, int i){
         nbrec++;
         if(Si == S)
             afficherSolution(a, S, x, i);
         else if((i < a.length) && (Si < S)){
             x[i] = 0;
             sacADosrec(a, S, x, Si, i+1);
             x[i] = 1;
             sacADosrec(a, S, x, Si+a[i], i+1);
         }
     }
 static void sacADos(int[] a, int S){
         int[] x = new int[a.length];

         nbrec = 0;
         sacADosrec(a, S, x, 0, 0);
         System.out.print("# appels=" + nbrec);
         System.out.println(" // " + (1 << (a.length + 1)));
     }
```

17. Rechercher dans du texte

1. Algorithme naïf

```
static boolean occurrence(char[] T, char[] M, int i){
    for(int j = 0; j < M.length; j++)
        if(T[i+j] != M[j]) return false;
    return true;
}

static void naif(char[] T, char[] M){
    System.out.print("Occurrences en position :");
    for(int i = 0; i < T.length-M.length; i++)
        if(occurrence(T, M, i))
            System.out.print(" "+i+",");
    System.out.println("");
}
```

Donner la complexité de cet algorithme en notant n la taille de T et m la taille de M.

2. Algorithme linéaire de Karp-Rabin : 1ère version

```
static long signature(char[] X, int m, int i){
        long s = 0;

        for(int j = i; j < i+m; j++)
            s += (long)X[j];
        return s;
    }
static void KR(char[] T, char[] M){
    int n, m;
    long hT, hM;

    n = T.length;
    m = M.length;
    System.out.print("Occurrences en position :");
    hM = signature(M, m, 0);
    for(int i = 0; i < n-m; i++){
        hT = signature(T, m, i);
        if(hT == hM){
            if(occurrence(T, M, i))
                System.out.print(" "+i+",");
            else
                System.out.print(" ["+i+"],");
        }
```

```
        }
        System.out.println("");
    }
```

Donner la complexité de cet algorithme

3. Algorithme linéraire de Karp-Rabin : 2ème version

```
static long B = ((long)1) << 16, p = 2147483647;

 // calcul de S(X[i..i+m-1])
 static long signature2(char[] X, int i, int m){
     long s = 0;

     for(int j = i; j < i+m; j++)
         s = (s * B + (int)X[j]) % p;
     return s;
 }

 // S(X[i+1..i+m]) = B S(X[i..i+m-1])-X[i] B^m + X[i+m]
 static long signatureIncr(char[] X, int m, int i,
                           long s, long Bm){
     long ss;

     ss = ((int)X[i+m]) - ((((int)X[i]) * Bm)) % p;
     if(ss < 0) ss += p;
     ss = (ss + B * s) % p;
     return ss;
 }

 static void KR2(char[] T, char[] M){
     int n, m;
     long Bm, hT, hM;

     n = T.length;
     m = M.length;
     System.out.print("Occurrences en position :");
     hM = signature2(M, 0, m);
     // calcul de Bm = B^m mod p
     Bm = B;
     for(int i = 2; i <= m; i++)
         Bm = (Bm * B) % p;
     hT = signature2(T, 0, m);
     for(int i = 0; i < n-m; i++){
         if(i > 0)
             hT = signatureIncr(T, m, i-1, hT, Bm);
         if(hT == hM){
             if(occurrence(T, M, i))
                 System.out.print(" "+i+",");
             else
                 System.out.print(" ["+i+"],");
         }
     }
     System.out.println("");
 }
```

Donner la complexité de cet algorithme

18. Produit de convolution et transformée de Fourier

1. La convolution : $(f * g)(n) = \sum_{m=-\infty}^{+\infty} f(m)g(n-m)$

Algorithme : soit f et g deux fonction, on calcule le produit de convolution f*g pour i=1 à n avec g à valeur non nulles sur l'intervalle [–p,p].

```
Pour i=1 à n
        P[i]=0
        Pour j=-p à p
                P[i] =P[i]+f(j).g(i-j)
        FPour
FPour
```

Calculer la complexité de l'algorithme.
Qu'en est il si p=n ?

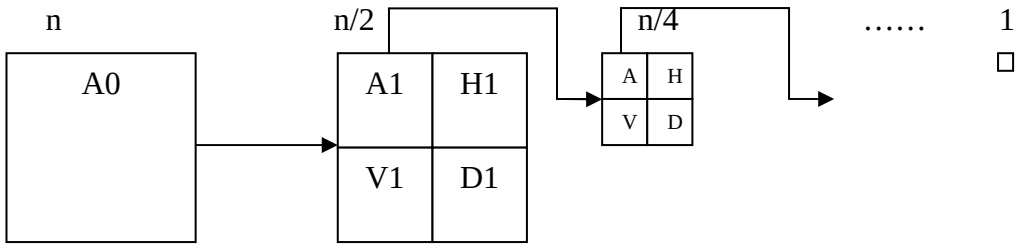
2. La transformée de Fourier : $\widehat{f(k)} = \sum_{p=1}^{n} f(p).e^{-2i\pi kp/n}$
   Algorithme :

```
Pour k=1 à n
        F[k]=0
        Pour j=1 à n
                F[k] =F[k]+f(k)*exp(k,n)
        FPour
FPour
```

3. Convolution et TF : $\widehat{(f*g)} = \hat{f}.\hat{g}$ donc $(f*g) = (\hat{f}.\hat{g})^{-1}$
   Calculer la complexité de cet algorithme

19. Transformée en ondelettes (examen 2007-2008)

Enoncé : la transformée en ondelettes permet de décomposée une image (fonction de N^2 dans R) dans une série d'espaces orthogonaux dont les bases sont dérivées d'une fonction appelée ondelette. Cette transformation permet l'analyse en espace et en fréquence de l'image. On calcule ainsi une série d'images d'approximations (A0, A1, …, An), de détails horizontaux (H1, H2, …,Hn), verticaux (V1, V2, ..., Vn) et diagonaux (D1, D2, …, Dn). La taille de chacune des images est divisée par 2 entre deux niveaux successifs. Le calcul des images est réalisé par un produit de convolution entre l'image et un filtre passe bas (PB) et un filtre passe haut (PH).



On rappelle qu'une série géométrique de premier terme **a** et de raison **q** a comme terme général $\mathbf{a.q^n}$. Sa somme partielle est $\mathbf{S_n = a*(1-q^n)/(1-q)}$
Exemple : la suite géométrique 1, 2, 4, 8, …, 2^n a pour premier terme 1, pour raison 2 (Car $U_{n+1} = 2*U_n$) et donc pour somme partielle jusqu'à n $1+2+4+\ldots+2^n = (1-2^n)/(1-2) = 2^n-1$

```
Algorithme
PH et PB sont des tableaux de taille 1xk
Le premier appel à ondelette est le suivant : O = Ondelette(A,n) n étant
une puissance de 2

Fonction Res = ondelette(A,p)
      Si p == 1
            Renvoyer A
      Sinon
            H = convolution(A,p,PH,PB,k)
            V = convolution(A,p,PB,PH,k)
```

```
            D = convolution(A,p,PH,PH,k)
            A1 = ondelette(convolution(A,p,PB,PB,k), p/2)
            Pour i=1 à p/2
                  Pour j=1 à p/2
                        Res[i,j] = A1[i,j]
                        Res[i+p/2,j] = V[i,j]
                        Res[i,j+p/2] = H[i,j]
                        Res[i+p/2,j+p/2] = D[i,j]
                  Fin pour
            Fin pour
      Renvoyer Res
Fin

Fonction Res = convolution(A,p,F1,F2,k)
      Pour i=1 à p
            Pour j=1 à p
                  Tmp[i,j] = 0
                  Pour m = 1 à k
                        Tmp[i,j] = Tmp[i,j] + F1[m]*A[i,j+m-k/2]
                  Fin pour
            Fin pour
      Fin pour
      Pour i=1 à p
            Pour j=1 à p
                  Tmp2[i,j] = 0
                  Pour m = 1 à k
                        Tmp2[i,j] = Tmp2[i,j] + F2[m]*Tmp[i+m-k/2,j]
                  Fin pour
            Fin pour
      Fin pour

      Pour i=1 à p/2
            Pour j=1 à p/2
                  Res[i,j] = Tmp2[i*2,j*2]
            Fin pour
      Fin pour
      Renvoyer Res
Fin
```

Questions
1. Quelle est la complexité en temps de la fonction convolution en fonction de p et k ?
2. Quelle est la complexité en temps de la fonction ondelette (en fonction de n et k)?
3. Quelle est la complexité en espace de l'algorithme (en fonction de n)?
4. A quelle classe de complexité appartient l'algorithme ?

20. Algorithmes approchés (examen 2007-2008)

Un problème NP-difficile peut être abordé de deux manières :

1. De manière exacte : on applique l'algorithme de complexité non polynomiale qui trouve la solution exacte en un temps très long
2. De manière approchée : on applique un algorithme plus simple qui n'est pas sur de trouver la solution exacte mais qui s'exécute en un temps plus court.

On trouve parmi ces algorithmes les algorithmes génétiques. Ils sont basés sur un codage des solutions potentielles et une évaluation de celles-ci. On cherche la meilleure solution (celle ayant la plus grande évaluation) en utilisant des opérateurs de sélection, croisement et mutation sur une population de solutions potentielles.

L'algorithme génétique générique est décrit ci-dessous

On considère que le codage d'une solution potentielle **e** est un tableau de valeurs de taille n (e= | e(1) | e(2) | | | | | e(n) |)

```
Fonction Sol = AG(p)
Recherche de la solution par algorithme génétique
      Génération = 0
      Pop = générer(p)

      Tant que ( Génération < G ) faire
            V = évaluation(Pop, n, p)
            Sel = sélection(Pop,V,p, p/4)
            E = croisement(Sel, n, p/4)
            E = mutation(E, n, p/4, tx)
            VE = évaluation(E, n, p/4)
            Pop = sélection(E+P,V+VE, p)
            Génération = Génération+1
      FTantque
```

```
Fonction P=générer(k)
Génération aléatoire de k solutions potentielles au problème
```

```
Fonction V=évaluation(E,n,k)
Calcule la valeur des k solutions potentielle de E, on considère que
l'évaluation d'une solution est linéaire par rapport à sa taille n.
```

```
Fonction Sel=sélection(E, V, m, k)
Sélection des k meilleures solutions de l'ensemble E de m solutions
potentielles. V est la valeur des k solutions
      Trier l'ensemble V dans l'ordre décroissant
      Sel = les k solutions potentielles de E correspondant aux k premières
valeurs de V
      Retourner (Sel)
```

```
Fonction E=croisement(P,n,k)
On croise ici les k solutions potentielles de P deux par deux
Pour i=1 à k/2
            s = valeur aléatoire comprise entre 2 et n-1
            Pour m=1 à s
                  E[i,m] = P[i, m]
                  E[k-i+1,m] = P[k-i+1, m]
            FinPour
            Pour m=s+1 à n
                  E[i,m] = P[k-i+1, m]
                  E[k-i+1,m] = P[i, m]
            FinPour
      FPour
      Retourner(E)
```

```
Fonction E=mutation(E, n, k, tx)
change aléatoirement une ou plusieurs solutions parmi les k solutions de E,
      Pour i=1 à k faire
            Pour j=1 à n faire
                  V=valeur aléatoire entre 0 et 1
                  Si (V<tx)
                        E[i,j] = valeur aléatoire
                  FSi
            FPour
      FPour
      Retourner(E)
```

Donnez la complexité en temps et en espace de cet algorithme pour p=10, p=n, p=2n, $p=n^2$ (précisez la complexité de chaque fonction).

21. Valeur majoritaire : Algorithme naïf (examen 2008-2009)

Soit un tableau A de taille n (n étant une puissance de 2 : $n=2^k$)
Un élément x est dit majoritaire si et seulement si x est présent au moins $\frac{n}{2}+1$ fois dans A.

1. Occurrence : calculer en justifiant la complexité de la fonction suivante en fonction de i et j

```
Occurrences(x,A,i,j)
    Cpt = 0
    Pour k=i à j faire
        Si A[k] = x alors  Cpt = Cpt +1  FSi
    FPour
    Renvoyer Cpt
```

2. Majoritaire : calculer en justifiant la complexité de la fonction suivante

```
Majoritaire(A)
    Pour i=1 à n/2 faire
        Si Occurrences(A[i],A,i,n)>n/2 alors Renvoyer vrai FSi
    FPour
    Renvoyer Faux
```

22. Valeur majoritaire : Algorithme diviser pour régner version 1(examen 2008-2009)

Soit un tableau A de taille n (n étant une puissance de 2 : $n=2^k$)
Un élément x est dit majoritaire si et seulement si x est présent au moins $\frac{n}{2}+1$ fois dans A.

```
Majoritaire(A,i,j)
Si i=j alors  Renvoyer(vrai,A[i])
Sinon
    (rx,x)=Majoritaire(A,i,(i+j-1)/2)
    (ry,y)=Majoritaire(A,(i+j+1)/2,j)
    Si rx = faux et ry=faux alors  Renvoyer(faux,0)  FSi
    Si rx = vrai et ry=vrai alors
        Si x=y alors  Renvoyer(vrai,x)
        Sinon
            Cx = Occurrences(x,A,i,j)
            Cy = Occurences(y,A,i,j)
            Si cx > (j-i+1)/2 alors  Renvoyer(vrai,x)  FSi
            Si cy>(j+i+1)/2 alors  Renvoyer(vrai,y)  FSi
            Renvoyer(faux,0)
        Fsi
    Sinon si rx=vrai alors
        Si Occurences(x,A,I,j)>(j+i+1)/2 alors  Renvoyer(vrai,x)
        Sinon  Renvoyer(faux,0)
        FSi
    Sinon
        Si Occurences(y,A,I,j)>(j+i+1)/2 alors  Renvoyer(vrai,y)
        Sinon  Renvoyer(faux,0)
        FSi
    FSi
FSi
```

23. Valeur majoritaire : Algorithme diviser pour régner version 2 (examen 2008-2009)

Soit un tableau A de taille n (n étant une puissance de 2 : $n=2^k$)

Un élément x est dit majoritaire si et seulement si x est présent au moins $\frac{n}{2}+1$ fois dans A.

1. Calculer en justifiant la complexité de l'algorithme suivant

```
Pseudo Majoritaire(A,i,j)
    Si i=j alors     Renvoyer(vrai,A[i],1)
    Sinon
        (rx,x,cx)=PseudoMajoritaire(A,i,(i+j-1)/2)
        (ry,y,cy)=PseudoMajoritaire(A,(i+j+1)/2,j)
        Si rx = faux et ry=faux alors          Renvoyer(faux,0,0)
        Sinon si rx = vrai et ry=faux alors    Renvoyer(vrai,x,cx+(j-i+1)/4)
        Sinon si rx = faux et ry=vrai alors    Renvoyer(vrai,y,cy+(j-i+1)/4)
        Sinon si rx=vrai et ry=vrai alors
            Si x=y alors                       Renvoyer(vrai,x,cx+cy)
            Sinon si cx=cy alors               Renvoyer(faux,0,0)
            Sinon si cx>cy alors               Renvoyer(vrai, x, (j-i+1)/2+cx-cy)
            Sinon                              Renvoyer(vrai, y, (j-i+1)/2+cy-cx)
            FSi
        Fsi
    FSi
```

2. Calculer en justifiant la complexité de l'algorithme suivant

```
Majoritaire(A)
    (rep,x,cx)=PseudoMajoritaire(A,1,n)
    Si rep=faux alors      Renvoyer faux
    Sinon
        Si Occurrences(x,A,1,n) > n/2 alors     Renvoyer(vrai)
        Sinon                                   Renvoyer(faux)
        FSi
    FSi
```