UVHC-ISTV

Licence 2 semestre 3 : ALGO3
Fiche de TD 1 d'Informatique-algorithmique

Exercice 1

On veut calculer la somme des n premiers termes de la série suivante :

$$1 + 1/ 1.5 + 1/ 2.25 + 1/ 3.375 + \quad ... \quad + 1/ (1.5)^n$$

1- Ecrire un programme effectuant ce calcul en utilisant une boucle ***for***.
2- Transformer la solution précédente en utilisant ***while*** puis ***do while*** .
3- Ecrire une fonction *fex1* effectuant ce calcul, puis la fonction principale .

Exercice 2

1- Ecrire une fonction nommée *f1* se contentant d'afficher « Vive l'informatique » (elle ne possédera aucun argument ni valeur de retour) .
2- Ecrire une fonction nommée *f2* qui affiche « Vive l'informatique» un nombre de fois égal à la valeur reçue en argument (*int*) et qui ne renvoie aucune valeur.
3- Ecrire une fonction nommée *f3* qui fait la même chose que f2 mais qui de plus renvoie la valeur (*int*) 1 si l'argument est strictement positif et 0 sinon . .
4- Ecrire un petit programme appelant successivement chacune de ces 3 fonctions après les avoir convenablement déclarées sous forme d'un prototype.

Exercice 3

1- Ecrire une fonction qui reçoit 3 arguments (2 nombres réels et un caractère) et qui fournit un résultat correspondant à l'une des 4 opérations appliquées à ses deux premiers arguments en fonction de la valeur du dernier : + - * / .
2- Que faut-il faire pour tenir compte des risques de division par zéro.
3- Ecrire un petit programme (*main*) utilisant cette fonction pour effectuer les 4 opérations sur 2 nombres fournis en entrée.

Exercice 4

1. Définir un type tableau de MAX réels (*Tmax*) où MAX est une constante.
2. Ecrire la fonction de lecture d'un tableau t de nb valeurs réelles (nb supposé connu).
   void lect1 (Tmax t , int nb)
3. Ecrire une autre fonction de lecture du tableau t où nb est à lire aussi.
   int lect2 (Tmax t)
4. Ecrire une fonction d'affichage des N valeurs du tableau en mettant 10 valeurs par ligne avec 2 digits décimaux (cf poly une introduction au C)
   void affich (Tmax t, int n)
5. Ecrire une fonction tri de tri des valeurs du tableau (ordre croissant) .
   void tri (Tmax t , int nb)

UVHC-ISTV

Licence 2 semestre 3 : ALGO3
Fiche de TD 2 d'Informatique-algorithmique

Exercice 1

Soient 2 tableaux a et b de type Tab (tableau de 10 entiers) déclarés ainsi : *Tab a , b ;*
Ecrire les 2 fonctions permettant de recopier, dans b, tous les éléments positifs de a, en complétant éventuellement a par des zéros : cas 1 : les éléments restent à la même place
cas 2 : les zéros sont mis à la fin du tableau.

Exercice 2

1- Définir un type Chaine de caractères.
2- Ecrire une fonction *length* qui calcule la longueur d'une chaîne *ch*.
3- Ecrire une fonction logique *egal* qui compare 2 chaînes de caractères *ch1* et *ch2*.
int egal (Chaine ch1 , Chaine ch2)
4- Ecrire une fonction *concat* qui recopie la chaîne *ch2* à la suite de la chaîne *ch1* dans une nouvelle chaîne *ch3* (attention de bien reconstituer correctement la chaîne finale avec le caractère de fin '*\0*')
void concat (Chaine ch1 , Chaine ch2 , Chaine ch3)
5- Ecrire le programme principal

Exercice 3 : La notion de Structure en C

La communauté européenne (CE) désire réaliser un outil informatique capable de manipuler des données géographiques concernant les 25 pays de la CE.
Chaque **pays** est représenté par les informations suivantes :le nom du pays (chaîne de 25 caractères), le nom de la capitale (chaîne de 20 caractères), la superficie en km^2 (entier) , la population en million d'habitants (réel), la monnaie locale (chaîne de 20 caractères).

Exemple

France	Paris	550000	64.5	euro
Nom	Capitale	superficie	habitants	monnaie

a) Déclarer le type **Pays**.
b) Ecrire une fonction **Pays saisie1 ( )** permettant de saisir les données concernant un pays
c) Ecrire une fonction **void affiche (Pays p)** permettant d'afficher les données concernant un pays p.
d) L'ensemble des données sur les 15 pays européens est représenté par un tableau de type **Pays15**:
typedef Pays Paysce [MAX] ;
Ecrire une fonction saisie **void saisie_ totale (Paysce ce)** utilisant la fonction saisie1 de la question 1 et permettant de saisir la totalité des données sur les 25 pays de CE.
e) Ecrire une fonction **void resultats1 (Paysce ce)** permettant de calculer et d'afficher les valeurs suivantes : la superficie totale des 15 pays de CE, la population totale de la CE, la densité moyenne (en nombre d'habitants/Km2)

Exercice 4 facultatif

Ecrire un sous-programme qui fusionne deux tableaux triés d'entiers ayant n1 et n2 valeurs en un troisième de taille à calculer.

UVHC - Institut des Sciences et Techniques ISTV

Licence Sciences, Technologies et Santé 2 semestre 3
TP d'Algorithmique et Programmation N° 1

Exercice 0

Se connecter dans l'environnement Linux (identifiant et mot de passe) , ouvrir la fenêtre terminal, lister ses fichiers et répertoires (ls), créer un répertoire Tpapri3 (mkdir Tpapri3) , lister (ls) , changer de répertoire (cd Tpapri3) , lister ce nouveau répertoire vide (ls) , lancer un éditeur (gedit &), enfin positionner les deux fenêtres sur votre écran pour occuper tout l'espace. S'habituer avec la souris à passer de la fenêtre Editeur à celle de Terminal.

Exercice 1

```
#include<stdio.h>            /*  texte à saisir sous le nom   tp1_ex1.c  */
int  length ( char ch [ ] )  {      int i = 0;
                                    while ( ch[ i ] != '\0')  i ++;
                                    return i;
                             }
main ( ) {     char chaine [ 80 ] ; printf ("entrer une chaine     ") ;
               scanf ( "%s", chaine);
               printf ( "\nlongueur de %s = %d\n", chaine, length (chaine)   ) ;
         }
```

1) Saisir sous éditeur puis dans la fenêtre Terminal lister (ls) et compiler (cc).

cc -o tp1_ex1 tp1_ex1.c

2) S'il n'y a pas d'erreurs, exécuter ce programme pour différentes valeurs de n

./tp1_ex1

3) Saisir la fonction suivante et modifier le programme principal pour la tester.

```
int  egal ( char c1 [ ], char c2 [ ])
                        {       int i =0, tmp=1 ;
                                while ( tmp && c1[ i ] != '\0'  )
                                        if (c1[i] !=c2[i] ) tmp =0 ; else i++ ;
                                return tmp &&  c1[i] == c2[i];
                        }
```

4) Exécuter plusieurs fois le programme avec différentes chaînes.
5) Modifier le programme en introduisant une boucle POUR de 3 répétitions.
6) Rajouter la fonction concat vue en TD 2 et compléter le programme.

Exercice 2

Soient 2 tableaux a et b de type Tab (tableau de MAX entiers) déclarés ainsi : *Tab a , b ;*

1. Ecrire les 2 fonctions permettant de recopier, dans b, tous les éléments positifs de a, en complétant éventuellement a par des zéros :
   cas 1 : les éléments restent à la même place
   cas 2 : les zéros sont mis à la fin du tableau.
2. Après avoir écrit deux sous-programmes de lecture et affichage, tester ces deux fonctions.
3. Ecrire un sous-programme qui fusionne deux tableaux triés d'entiers ayant n1 et n2 valeurs en un troisième de taille à calculer et retourner.
4. Compléter la fonction principale qui permettra de lire deux tableaux, les fusionner et les afficher.

UVHC-ISTV

Licence 2 semestre 3 : ALGO3
Fiche de TD 3 d'Informatique-algorithmique

Exercice 1 : Les pointeurs

Nous allons reprendre l'exercice précédent 3 du TD 2 en utilisant les pointeurs.

a) Ainsi au lieu d'avoir dans la question b une fonction renvoyant un pays, nous allons utiliser un pointeur **ppy** sur un pays. La fonction saisie devient :

void saisie2 (Pays *ppy)

Ecrire la fonction saisie2

b) Reprenant maintenant la question e de l'exercice 1, réécrire la fonction resultats1 en renvoyant les 3 résultats (superficie totale, population totale et densité) au lieu de les afficher.

Exercice 2 : Les pointeurs

Dans l'exercice 3 du TD1 nous avons écrit une fonction **int operation (float nb1, float nb2, char oper)** permettant de calculer le résultat de l'application de l'opérateur oper aux deux nombres nb1 et nb2. Réécrire une fonction permettant de calculer les 4 résultats nb1-nb2, nb1+nb2, nb1*nb2 et nb1/nb2 dans les variables pointées par les 4 paramètres résultats.

Exercice 3 : Pointeurs et tableaux

Reprendre dans la fiche 2 les exercices suivants en remplaçant le paramètre tableau par un paramètre pointeur sur ...

1- exercice 1 questions 2 et 3

- Ecrire une fonction *length* qui calcule la longueur d'une chaîne *ch*.
- Ecrire une fonction logique *egal* qui compare 2 chaînes de caractères *ch1* et *ch2*.

*int egal (char * ch1 , char * ch2)*

- Ecrire une fonction *concat* qui recopie la chaîne *ch2* à la suite de la chaîne *ch1* dans une nouvelle chaîne *ch3* (attention de bien reconstituer correctement la chaîne finale avec le caractère de fin *'\0'*)

*void concat (char * ch1 , char * ch2 , char * ch3)*

- Ecrire le programme principal

2- exercice 4 questions

Ecrire un sous-programme qui fusionne deux tableaux triés d'entiers ayant n1 et n2 valeurs en un troisième de taille à calculer.

*void fusion (Tab a, int na, Tab b, int nb, Tab c, int * nc)*

Ecrire un sous-programme qui fusionne deux tableaux triés de caractères (chaînes) en un troisième

*void fusion2 (char * a, char * b, char *c)*

UVHC - Institut des Sciences et Techniques ISTV

Licence Sciences, Technologies et Santé 2 semestre 3
TP d'Algorithmique et Programmation N° 2

Exercice 0

Se connecter, ouvrir la fenêtre Terminal et aller dans le répertoire Tpapri3 **cd Tpappri3**

Puis lancer l'éditeur **gedit &**

Exercice 1: Les pointeurs

Reprendre l'exrecice 1 du TP 1 et remplacer, dans les fonctions, systématiquement le tableau par un pointeur sur caractère.

Exercice 2: Les structures

Le garage LaKass a décidé d'informatiser la gestion de son parc de voitures afin de connaître à tout moment l'état de ce parc. Cette gestion devrait permettre, par exemple, de savoir combien de voitures sont présentes dans le garage à une date donnée, de faire des statistiques sur les flux des voitures qui entrent ou qui sortent du garage, ou bien encore de repérer les bons clients. Vous êtes chargés de faire cette informatisation.

Chaque voiture est caractérisée par sa marque, son modèle, sa date de mise en service, sa puissance, son numéro d'immatriculation, son propriétaire et la liste des interventions qu'elle a subies dans le garage. La marque d'une voiture est une chaîne de caractères. Son modèle comporte une séquence de 4 lettres, suivies de 10 chiffres suivis de 2 lettres. Une date se compose d'un numéro de jour (de 1 à 31), d'un numéro de mois (de 1 à 12) et de l'année (en 4 chiffres pour éviter les problèmes du passage à l'an 2000 !). Un numéro d'immatriculation comporte 9 caractères alphanumériques. Les numéros d'immatriculation sont tous différents. Le propriétaire d'une voiture est défini par son nom et son prénom, son adresse (numéro, rue, ville et code postal). On supposera que tous les propriétaires ont des noms différents.

Une intervention est définie par un numéro d'un intervenant (qui travaille dans le garage), une date de début et une date de fin, et le prix de la main d'œuvre ainsi que celui des pièces de rechange et enfin par le kilométrage de la voiture au moment de l'intervention. La liste des interventions est définie par le type **DesInterventions** :

```
TYPE DesInterventions =    ENREGISTREMENT   Nb: entier;
            Contenu: tableau[MAXINTERVENTIONS]de Intervention Fin ;
```

1) Définir les types ***Date, Adresse, Propriétaire, Intervention, Voiture*.**

2) Ecrire les sous-programmes de lecture et d'affichage des types précédents, soit sous la forme d'une fonction retournant le résultat ou d'une procédure ayant en entrée une valeur de ce type.

3) Ecrire le programme principal permettant de tester les sous-programmes.

4) Codifier le type Garage suivant
```
TYPE Garage =ENREGISTREMENT Nb:entier ;
                Contenu: tableau[MAXVOITURE] de Voiture; Fin ;
```
Dans la suite on utilise la variable laKass de type Garage pour représenter le garage considéré.

5) Ecrire le programme principal permettant de faire la saisie et l'affichage d'une variable *lakass* de type *Garage*

UVHC-ISTV

Licence Sciences, Technologies et Santé 2 semestre 3 : ALGO3

Fiche de TD 4 d'Informatique-algorithmique

Exercice 1 : Fonction puissance

1) Ecrire la fonction récursive qui renvoie la valeur de a^n avec a réel et n entier strictement positif.
2) Modifier la fonction pour prendre en compte également les valeurs négatives ou nulles de n.

Exercice 2 : Chaînes de caractères

1) Ecrire une fonction récursive qui renvoie la longueur d'une chaîne de caractères
2) Ecrire une fonction récursive qui affiche une chaîne à l'envers (en partant du dernier caractère)
3) Ecrire une fonction récursive qui vérifie l'égalité entre 2 chaînes
4) Ecrire une fonction récursive qui vérifie l'appartenance d'un caractère à une chaîne.

Exercice 3 : Tableaux et récursivité

Soit un tableau T de MAX entiers. Ecrire les fonctions récursives suivantes :

1) saisie récursive des N entiers de T .
2) calcul de la somme des N éléments de T
3) remplacer toutes les occurrences de l'élément e dans T par 0
4) vérification que T est trié (renvoie 1 si trie, 0 sinon)
5) renvoi de l'indice de l'élément minimal de T entre les positions i et N - 1
6) donner l'équivalent récursif de l'algorithme (itératif) de tri suivant :

```
pour i de 0  à N – 2  faire
    j ← indice du min de T [ i .. N ]
    si j ≠ i , permuter T[i] et T[j]
    i ++
```

Exercice facultatif : Tableaux et récursivité

Ecrire un sous-programme récursif qui fusionne deux tableaux triés t1 et t2 d'entiers ayant n1 et n2 valeurs en un troisième t3. Que se passe-t-il si on veut aussi calculer la taille du tableau résultant ?

TP d'Algorithmique et Programmation N° 3

Exercice 1 : La récursivité sur les chaînes

Reprendre l'exrecice 1 du TP 1 (ou ex 1 du TP 2) et remplacer, dans les fonctions, systématiquement la solution itérative par une solution récursive.

Exercice 2 : La récursivité sur les tableauxs

Reprendre l'exrecice 3 du TD 4 et saisir sous éditeur les deux premières fonctions, puis écrire le programme principal déclarant N et T, lisant N puis faisant appel aux 2 fonctions.
Compléter avec les fonctions suivantes.

UVHC-ISTV

Licence Sciences, Technologies et Santé 2 semestre 3 : ALGO3

Fiche de TD 5 d'Informatique-algorithmique

Exercice 1 : Fichier binaire de structures

1) Définir une structure étudiant comprenant un code entier,un nom, un prénom et une note.
2) Ecrire la fonction de création du fichier binaire avec un étudiant.
3) Ecrire la fonction d'ajout en fin de fichier.
4) Ecrire une fonction récursive de calcul du nombre d'étudiants.
5) Donner une solution itérative.

Exercice 2 Fichier Texte de noms (chaînes de 20 caractères)

typedef char Chaine [20]; typedef struct { int num; Chaine nom; } Enreg ; void creation (Chaine nomfichier) ; void visualiser (Chaine nomfichier) ; int recherche (Chaine nomfichier, Chaine nom) ; /* fonction booléenne */

1) Ecrire le programme principal utilisant les 3 fonctions avec un menu, un choix et un aiguillage (***switch***), le nom du fichier (« algo3») étant lu.
2) Ecrire le sous-programme de création d'un ficher de 10 enregistrements contenant chacun un numéro suivi d'une chaîne en utilisant ***fprintf*** ; le nom du fichier sera passé en paramètre .
3) Ecrire un sous-programme de visualisation de tout le fichier (numéro et nom).
4) Ecrire une fonction de recherche de l'existence d'un nom dans le fichier (pour effectuer la comparaison on utilisera la fonction ***strcmp***)
5) Sous UNIX vérifier que le fichier « algo3 » existe et regarder sa taille. Le copier sous un autre nom. Accéder par l'éditeur au fichier « algo3a» et regarder son organisation …
6) Ecrire la fonction d'ajout d'un enregistrement en fin de fichier.
void ajout (Enreg e, Chaine nomfichier)
7) Ecrire une fonction itérative de calcul du nombre d'enregistrements dans le fichier.
int nombre (Chaine nomfichier)
8) Donner une solution récursive.

TP d'Algorithmique et Programmation N° 4

Exercice 1 Fichier de voitures (suite du TP 2)

1) Ecrire une fonction permettant de sauvegarder un tableau de voitures dans un fichier
2) Ecrire une fonction permettant de transférer dans un tableau de voitures le fichier.
3) Compléter le programme principal pour prendre en compte ces deux nouvelles procédures.

Exercice 2 Fichier Texte de noms (chaînes de 20 caractères)

typedef char Chaine [20]; typedef struct { int num; Chaine nom; } Enreg ; void creation (Chaine nomfichier) ; void visualiser (Chaine nomfichier) ; int recherche (Chaine nomfichier, Chaine nom) ; /* fonction booléenne */

Reprendre les questions vues en TD avec le fichier texte …

UVHC-ISTV

Licence Sciences, Technologies et Santé 2 semestre 3 : ALGO3

TP d'Algorithmique et Programmation N° 5

Exercice 1 : Comparaison d'algorithmes de recherche dans un tableau

#define MAX 1000
typedef int Vec [MAX+1] ;
int f1 (int c, Vec e, int n); /*prototype de fonction retournant le nombre de comparaison*/
int f2 (int c, Vec e, int n); /*prototype de fonction retournant le nombre de comparaison*/
int f3 (int c, Vec e, int n); /*prototype de fonction retournant le nombre de comparaison*/
int f4 (int c, Vec e, int n); /*prototype de fonction retournant le nombre de comparaison*/

1) Ecrire le programme principal qui remplira tout le tableau t de valeurs (par exemple t[i]=2*i , i = 1… MAX) puis calculera le nombre moyen de comparaisons (moyenne effectuée sur MAX valeurs différentes) pour les 4 fonctions précédentes dans 3 cas :

- la valeur recherchée est toujours dans le tableau (c = 2*i , i = 1… MAX)
- la valeur recherchée est une fois sur 2 dans le tableau (c = 2*i , puis 2*i-1, i = 1… MAX)
- la valeur recherchée n'est jamais dans le tableau (c = 2*i -1 , i =1 … MAX)

```
main () { int c, i, tot1, tot2, tot3, tot4 ;          Vec e;
          for  ( i = 0; i<= MAX; i ++ )   e [ i ] = 2 * i ;
          tot1= tot2 = tot3 = tot4 = 0;
          for (  i = 1; i <= MAX; i ++ )
          {   c = 2 * i;
              tot1 += f1 ( c, e,  MAX +1);  tot2 += f2 ( c, e, MAX +1 );
              tot3 += f3 ( c, e, MAX +1 );  tot4 += f4 ( c, e, MAX + 1 );
          }
          printf  ("%d %d %d %d\n", tot1/ MAX, tot2/ MAX, tot3/ MAX, tot4/ MAX);
    }   /*  dupliquer les 7 lignes précédentes  pour le cas 2 puis le cas 3  */
```

2) Ecrire deux fonctions (*f1* récursive et *f2* itérative) qui recherchent l'existence de l'entier ***c*** dans un tableau ***e,*** supposé non trié, de type ***Vec*** ayant ***n*** valeurs. Les deux fonctions retourneront le nombre de comparaisons (ou nombre d'appels récursifs) pour obtenir la réponse.
3) Ecrire deux fonctions récursives *f3* et *f4* qui recherchent l'existence de l'entier ***c*** dans un tableau ***e*** trié (ordre croissant) de type ***Vec*** ayant n valeurs : *f3* recherche séquentielle et *f4* recherche dichotomique. Les deux fonctions retourneront le nombre de comparaisons (nombre d'appels récursifs) pour obtenir la réponse.
4) Peut-on conclure ?

Exercice 2 : Recherche du PGCD de a et b (a <= b)

p1 (a, b) = a si a | b sinon p1 (a – 1, b)
p2 (a, b) = a si a = b sinon si a < (b–a) alors p2 (a, b- a) sinon p2 (b - a, a)
p3 (a, b) = a si a | b sinon si a < (b–a) alors p3 (a, b- a) sinon p3 (b - a, a)
p4 (a, b) = b si a = 0 sinon p4 (b mod a, a)

1) Ecrire ces quatre fonctions récursives en renvoyant le nombre d'appels récursifs avant d'arriver à la solution.
2) Ecrire le programme principal, qui pour b ∈ [981, 1000] et pour a ∈ [2, b] calcule le nombre moyen et le nombre max d'appels récursifs pour chaque fonction.
3) Peut-on conclure ?

Licence Sciences, Technologies et Santé 2 semestre 3 : ALGO3

Fiche de TD 6 d'Informatique-algorithmique

Exercice 1 : complexité d'algorithmes sur fichiers

1) Evaluer la complexité Minimale et Maximale de la recherche de l'inclusion d'un fichier de taille n dans un autre de taille m :

- si les 2 fichiers ne sont pas ordonnés
- si les 2 fichiers sont ordonnés

2) Evaluer la complexité Minimale et Maximale de l'identité de 2 fichiers de taille n1 et n2

- si les 2 fichiers ne sont pas ordonnés
- si les 2 fichiers sont ordonnés

3) Evaluer la complexité de l'algorithme de fusion (vu aussi en cours) de deux fihiers de taille n1 et n2

Exercice 2 : complexité de calcul *float f (float x, int n)*

1. Ecrire une fonction <u>récursive</u> f1 calculant x^n en utilisant la récurrence f(x,n) = x* f(x,n-1), n≥0 . Evaluer la complexité de la fonction en nombre de multiplications.
2. Ecrire une fonction <u>récursive</u> f2 calculant x^n en utilisant la récurrence f(x, n)= f(x,n/2)*f(x, n/2) si n pair et f(x, n) = x* f(x,n/2)* f(x, n/2) si n impair,. Calculer la complexité dans ce cas.
3. Ré écrire la fonction f2 en faisant attention à sa complexité, c-a-d en ne faisant qu'un seul appel récursif à chaque fois ; recalculer la complexité dans ce cas.
4. Que penser de la complexité en nombre de multiplications et divisions dans les 3 cas ?
5. Facultatif : Proposer une solution itérative f4 pour f2bis.

Exercice 3 : complexité

```
int f ( int n )
{       int i = 1, j, s = 0 , m = n;
        while ( i < n )
                {       j = 1 ; while ( j  < m )
                                        { j++;s = s + i * j;}
                        i ++ ;
                }
        return  s;
}
```

1 Evaluer la complexité exacte de cet algorithme en termes de nombre d'itérations puis son ordre de grandeur.

2 Idem avec le nombre de multiplications.

3 Idem avec le nombre d'additions et multiplications.

4 Transformer l'algorithme en remplaçant les boucles *while* par des *for*. Réévaluer la complexité dans ce cas.

5 Donner une solution récursive de f en gardant une boucle.

Donner une solution récursive en supprimant les 2 boucles

Licence Sciences, Technologies et Santé 2 semestre 3 : ALGO3

Fiche de TP d'Informatique-algorithmique 6

Exercice 1

```
int max ( int t [ ] , int n , int * p )
        {     *p++;
              if (n ==0 || t[max (t, n-1,p)] < t[n])    return n;
                                          else          return max ( t, n-1,p);
        }
```

1. Transformer cette fonction en ajoutant un parmètre résultat p calculant la complexité de l'algorithme en termes de nombre de comparaisons. Ecrire le programme principal qui va remplir un tableau (t[i] = i , i=0, 99) puis calculer le nombre moyen de comparaisons pour 100 valeurs.
2. Transformer la fonction *max* en supprimant le double appel récursif afin d'améliorer la complexité. Retester cette version 2.
3. Proposer une solution itérative . Tester cette version 3.
4. Ecrire une fonction <u>récursive</u> de tri d'un tableau de n entiers en utilisant la fonction *max*. Evaluer la complexité de l'algorithme de tri (complexité exacte et ordre de grandeur).

Exercice 2

```
int f  ( int n ) { int i = 1, j , s = 0 ;
          while (  i < n )      {     j = 1;
                                      while (  j  < n ) {     s = s + i * j ;   j ++;  }
                                      i ++  ;
                                }
                return  s;
          }
```

1) Ajouter deux compteurs calculant le nombre d'additions et multiplications. Tester.
2) Transformer f en une solution récursive en supprimant le premier TQ et en gardant le second et en introduisant un paramètre supplémentaire i.
3) Ajouter deux paramètres résultat pour retourner les deux compteurs. Tester.
4) Transformer **j = 1 ; while (j < n){ s = s + i *j ; j ++ ; }** en une fonction itérative g et modifier f. Tester.

Exercice facultatif 3

On désire calculer le nombre C_n défini pour tout n entier naturel comme suit :

$$C_n = [\, C_{n-k} + C_{n-k+1} + C_{n-k+2} + \ldots + C_{n-1} \,] \qquad \text{et } C_n = n \text{ si } n <= k$$

1) Ecrire une fonction récursive ***calc1*** réalisant ce calcul :

int calc1 (int n, int k)

2) Ecrire le programme principal permettant de calculer C_n pour n et k lus.

3) L'opération fondamentale étant l'addition, modifier la fonction en ajoutant un paramètre résultat permettant de compter le nombre d'additions et ainsi de calculer la complexité pour n variant de k à k+10. Qu'en déduire ?

4) Ecrire en C une fonction récursive ***calc2*** qui fournit la valeur de C_n en mémorisant la suite des k valeurs précédentes dans un tableau

int calc2 (int n , int k, int c [ ])

5) Modifier la fonction pour calculer la complexité de *calc2* pour n variant de k à k+10.

UVHC-ISTV

Licence Sciences, Technologies et Santé 2 semestre 3 : ALGO3

Fiche de TD 7 d'Informatique-algorithmique

Exercice 1 Recherche dichotomique

```
int dicho ( Tab e , int c, int g ,int d )
{ int m = (g+d) / 2 ; if (g <= d )    if (c == e[m] ) return m ;
                                        else if ( c < e[m] ) return dicho (e, c, g, m-1);
                                                else return dicho ( e, c, m+1, d);
                    else return -1 ;
}
```

1) Dérouler la fonction *dicho (e, c, 0, 9)* avec e= [1 2 2 5 5 6 9 12 15 15] pour c=5, puis 6.
2) Transformer la fonction pour trouver la place de la première occurrence de *c* dans *e* (indication: si on trouve une valeur *m* tq *e[m]=c*, il faut continuer la recherche à gauche de *m*, *m* compris). Expliquer sur l'exemple.
3) Ecrire une version itérative de *dicho*, en la dérécursivant.

Exercice 2 : tri d'un petit nombre de valeurs

Le but de cet exercice est d'étudier les algorithmes de tri de n valeurs entières pour n= 4, 5 et 8. On ne compte que les comparaisons entre valeurs et on note comp(n) leur nombre.

1) liste de 4 valeurs rangées dans 4 variables a, b, c, d . ***void tri4 (int *pa, int *pb, int *pc, int *pd)***

1.1 Algo 1 : donner un algorithme pour trier a, b, c, d avec comp(4) = 6 (algo 1). Justifier.

1.2 Algo 2 : donner un algorithme pour trier a, b, c, d avec comp(4) = 5 (algo 2). Justifier.

Sans refaire l'algo, donner la valeur de comp(5) pour les 2 algorithmes (algo 1 et algo 2).

2) On suppose que n=8. On demande de préciser le nombre de comparaisons pour chacun des 3 algorithmes basés sur les principes suivants (mais on ne demande pas d'écrire les algorithmes) et de conclure:

2.1 Faire deux listes de 4 valeurs consécutives, utiliser deux fois l'algorithme algo 2 et fusionner les deux listes triées obtenues.

2.2 Faire deux listes de 4 valeurs : les positions paires et les positions impaires, utiliser 2 fois l'algorithme algo 2 puis fusionner les listes triées obtenues.

2.3 Faire 2 listes consécutives : une de 5 et une de 3, utiliser 2 fois l'algorithme algo 2 puis fusionner les listes triées obtenues.

Exercice facultatif 3 : tri par insertion

Soit une liste de N valeurs entières représentées linéairement par un tableau de type *Tab* .

1) Donner l'algorithme de la fonction permettant d'insérer à sa place la $(I+1)^{ème}$ valeur *t[i]* dans les $(I+1)^{ères}$ positions du tableau t supposé trié entre 0 et i-1: Par comparaisons et permutations la valeur insérée remonte dans le tableau jusqu'à sa place : ***void inserer1 (Tab t , int i)***

Donner les complexités minimale et maximale en nombre de comparaisons.

2) Donner un autre algorithme d'insertion : on cherche en séquence la place k de *t[i]* entre o et i, puis on décale les positions k à i-1, enfin on insère *t[i]*. Donner les complexités minimale et maximale.
3) Donner un autre algorithme d'insertion dichotomique : on cherche par dichotomie la place k de *t[i]* entre o et i, puis on décale les positions k à i-1, enfin on insère *t[i]*. Donner les complexités minimale et maximale.
4) En utilisant le sous-programme 1 ou 2 ou 3 :
   4.1 Donner un algorithme de tri récursif par insertion .

6) 4.2 Donner la complexité au mieux et au pire.

UVHC-ISTV

Licence Sciences, Technologies et Santé 2 semestre 3 : ALGO3

Fiche de TD 8 d'Informatique-algorithmique

Exercices sur le TRI

1. QUICKSORT

```
int partition ( Tab t, int i, int j, Element piv )
 {      int g = i , d = j ;
        do      {       perm ( & t [ g ], & t [ d ] ) ;
                        while ( t [ g ] < piv ) g = g+1;
                        while ( t [ d ] > piv ) d = d - 1;
                }   while  ( g < d );
         return  g ;
 }
int p2 ( Tab t, int i, int j, Element piv)
  {     if ( i > j ) return i;
                else if ( t[i] < piv )  return p2 (t, i+1, j, piv);
                                else    if (t[j]  >= piv)       return  p2 ( t, i, j-1, piv);
                                                                else    {       perm ( & t[i], & t[j]) ;
                                                                                return p2 ( t, i, j, piv);
                                                                        }
 }
```

1.1 Dérouler l'algorithme *partition* sur l'exemple suivant de tableau ayant 10 valeurs avec pivot=10 :
3 17 9 15 2 7 10 8 12 11 qui devient 3 8 9 7 2 10 15 17 12 11
Quelle est la valeur renvoyée par *partition* ?

1.2 Que se passe-t-il avec une autre valeur de pivot (exemple 3 ou 17 ou 2) ?

1.3 Quelle est la complexité de partition en nombre de comparaisons dans le pire et le meilleur des cas ?

1.4 Quelle est la complexité de partition en nombre de permutations dans le pire et le meilleur des cas ?

1.5 Comment transformer partition en partition2 pour que la valeur du pivot se trouve au point de césure ?

1.6 Dérouler p2 sur le même exemple de tableau. La fonction *p2* fait-elle la même chose que *partition* ?

1.7 Comment transformer p2 pour avoir le même résultat qu'en 1.3 ?

2. HEAPSORT

```
void ajouter (Tab t, int p )
 {      int i=p;
        while (i>0 && t[i] <t[ (i-1)/2] )
                        {       perm (&t[i], &t[ (i-1)/2]);             i= (i-1)/2;         }
 }
void detruire ( Tab t, int p)
        {       int i =0, j;
                t[0] = t[p];
                while ( i <= p/2 )
                        {       if (2*i == p-2 ||  t[2*i+1]< t[2*i+2])    j = 2*i+1 ;       else  j = 2*i+2 ;
                                if ( t[i] > t[j] )      {               perm ( &t[i], &t[j] ) ;         i=j;
                                                        }
                                        else    exit ;
                        }
        }
void heapsort ( Tab t, int n ) {        int     p = 1 , min ;
                                        while ( p < n ){  ajouter ( t, p ) ; p++ ; }
                                        p= n-1;
                                        while ( p >= 0 ) { min= t[0]; detruire ( t, p );    t[p] = min;         p-- ;   }
                        }
```

2.1 Dérouler la 1ère boucle de l'algorithme *heapsort* sur l'exemple suivant de tableau ayant 10 valeurs :
3 17 9 15 2 7 10 8 12 11 qui devient 2 3 7 8 11 9 10 17 12 15

2.2 En supposant que ce tableau est la représentation d'un arbre binaire parfait, reconstruire l'arbre binaire parfait pour les 2 tableaux précédents. Chaque arbre est-il un tas ?

2.3 Dérouler l'algorithme *detruire* sur l'exemple suivant de tableau ayant 10 valeurs avec p=9:
2 3 7 8 11 9 10 17 12 15 qui devient 3 8 7 12 11 9 10 17 15 15

2.4 Dérouler la 2ème boucle de l'algorithme *heapsort* sur l'exemple suivant de tableau ayant 10 valeurs :
2 3 7 8 11 9 10 17 12 15 qui devient 17 15 12 11 10 9 8 7 3 2

UVHC-ISTV

Licence Sciences, Technologies et Santé 2 semestre 3 : ALGO3

Fiche de TP d'Informatique-algorithmique 7

Exercice 1 : tri Shell

C'est une variante améliorée de la méthode de tri par insertion séquentielle .

Au lieu d'insérer chaque élément à sa place dans la sous-liste triée de tous les éléments qui le précèdent, on l'insère dans une sous-liste d'éléments qui le précèdent, distants d'un certain incrément *incr*, que l'on diminue au cours de passages successifs sur la liste. C'est pourquoi on appelle parfois cette méthode, tri par incréments décroissants.

Au premier passage, on forme des sous-listes d'éléments distants de n/2 que l'on trie séparément par insertion. Au deuxième passage, on forme des sous-listes d'éléments distants de n/4 que l'on trie séparément, et ainsi de suite. A chaque passage, l'incrément est divisé par 2. Le tri s'arrête lorsque l'incrément est nul.

```
void trishell ( Tab t, int n, int incr)  { int i;  if ( incr > 0 )
                                    {   for ( i = incr ; i < n ; i++)  inserer (t, i - incr, incr ) ;
                                     trishell ( t, n, incr/2) ; }                    }
```

1) Ecrire la procédure qui permet d'insérer *t [i - incr]* à sa place dans le tableau t entre le début et *i-incr* . Ajouter des compteurs de nombres de comparaisons et de permutations.
2) Ajouter un appel d'une procédure d'affichage du tableau pour voir l'évolution de ce tableau.
3) Ecrire le programme principal qui va faire appel à la procédure *trishell*.
4) Illustrer le déroulement de la procédure avec n=11 et t = [5 12 25 38 20 7 9 23 5 4 8].
5) Dérécursiver (ou proposer une solution itérative pour) *trishell*.

Exercice 2 : Tri rapide

On partage la liste à trier en 2 sous-listes, telles que tous les éléments de la première liste soient inférieurs à tous les éléments de la seconde. On recommence jusqu'à avoir des sous-listes réduites à un élément.

Principe de **Quicksort** = ordonner le tableau t entre les positions 0 et n-1, en cherchant dans celui-ci une **clé pivot** autour de laquelle réorganiser ses éléments.

```
void tri_rap (Tab t , int n) { int i=0, j= n-1, x= t[j/2], z;
                                do {      while ( t[ i] <x) i++ ; while ( t[ j] > x) j-- ;
                                          if ( i<j ) { z=t[i] ; t[i]= t[j] ; t[j]= z ;}
                                          } while ( ++i <= ++j );
                                 if ( i == j +3){ i -- ; j++;}
                                 if ( j>0 ) tri_rap (t, j+1) ; if ( ( i< n-1) tri_rap (t+i, n-i) ;  }
```

1. Tester le déroulement de cette procédure avec 23 398 34 100 57 67 55 320 Ajouter des affichages intermédiaires du tableau pour voir son évolution.
2. Ajouter des compteurs du nombre de comparaisons et du nombre de permutations et afficher les à la fin.
3. Prendre un exemple de tableau déjà trié à l'endroit, puis à l'envers.
4. *Facultatif*

Le pivot joue un rôle important, ici c'est la valeur centrale *t[(n-1)/2*]. On pourrait choisir l'élément médian entre t[0], t[n-1], t[(n-1)/2]. Transformer l'algorithme.

UVHC-ISTV

Licence Sciences, Technologies et Santé 2 semestre 3 : ALGO3

Exercices supplémentaires d'Informatique-algorithmique

On suppose connues les déclarations suivantes :

```
typedef  int  Tab [1000] ;
void perm ( Element *px, Element *py) /* permute les valeurs pointées par px et py */
```

Exercice 1 : Recherche auto-adaptative

On veut effectuer la recherche de la valeur x dans le tableau non trié t : Tab ayant N valeurs.

1) Ecrire l'algorithme de recherche de l'existence de x dans t. Donner la complexité en moyenne.
2) Transformer l'algorithme précédent en plaçant l'élément éventuellement trouvé en début du tableau.
3) La complexité est-elle améliorée ? Expliquer.

Exercice 2 : tri ou pas tri

On considère deux tableaux A et B de type Tab de n éléments tous différents. On dit que A est contenu dans B si tous les éléments de A figurent au moins une fois dans B.

1) Ecrire une fonction qui, sans trier les tableaux , dit si A est contenu dans B. Quelle est la complexité au pire de l'algorithme en nombre de comparaisons ?
2) Améliore-t-on cette complexité si l'on trie l'un des deux tableaux et si oui préciser lequel ?
3) Améliore-t-on à nouveau la complexité si l'on trie les deux tableaux ?

Exercice 3 : Amélioration des performances d'un algorithme

```
void  algo1 ( Tab t , int n  )
      {      int  i,  j;
             for ( i=0 ; i<n-1 ; i++)
                   for (  j = n-2;  j >= 0;  j - - )
                         if  ( t [ j ] < t [ j+1 ] )  perm ( & t [ j+1 ] , & t [  j ] );
      }
void  algo2 ( Tab  t,  int  n )
      {      int i, j ;
             for ( i=0 ; i < n-1 ; i++)
                   for (  j = 0;  j < n-1;  j ++ )
                         if (  t [  j ] < t [  j+1 ] )   perm ( & t [ j+1 ] , & t [ j ] );
      }
```

1) Illustrer le déroulement de ***algo1*** sur un exemple de tableau de 5 valeurs par exemple 5 3 9 7 6
2) Vérifier que la fonction ***algo2*** fait des opérations différentes pour arriver au même résultat sur l'exemple.
3) Calculer la complexité en nombre de comparaisons pour ***algo1.***
4) On peut voir sur l'exemple précédent que cet algorithme effectue des parcours sans échange. Une première amélioration consiste à éviter ces parcours inutiles. Modifier ***algo1*** en introduisant un booléen et en modifiant POUR en TQ. Calculer la nouvelle complexité de ***algo1*** dans le meilleur des cas et dans le pire des cas.
5) On peut aussi mémoriser le plus grand indice (ou le plus petit) à partir duquel les échanges ne se font plus, c'est-à-dire l'indice en dessous (ou au-dessus) duquel la liste est triée. Par exemple
   8 5 2 6 4

Modifier ***algo1*** en ajoutant une variable entière supplémentaire. Comment se modifie la complexité ?

6) Enfin on remarquera une certaine dissymétrie dans l'exécution de l'algorithme sur les 2 listes suivantes : 6 5 4 2 8 et 2 8 6 5 4

Dans les deux cas, un seul élément n'est pas à sa place, mais la complexité n'est pas la même : calculer la pour ***algo1*** et ***algo2*** .

Pour remédier à cet inconvénient, il faut alterner la direction des parcours consécutifs.

Ecrire une procédure ***algo3***, basée sur le principe de ***algo1,*** qui prenne en compte ces améliorations.

Exercice 4 : Amélioration du tri à bulles

```
void  bulle1 ( Tab t , int n , int i )
      {      int j; if ( i<n-1)
```

```
        {       for ( j=n-2; j>=i; j - - )       if (t[j]>t[j+1]) perm(&t[j+1],&t[j]);
                bulle1(t,n,i+1);        }
    }
void  bulle2( Tab t, int n, int i, int j)
     {    if ( i<n-1 )
            {       if ( t[j]>t[ j+1] ) perm(&t[j+1],&t[j]);
                    if (j>i) bulle2(t,n,i,j-1); else bulle2(t,n,i+1,n-1);          }
    }
```

1) Les 2 fonctions précédentes vues en cours effectuent un tri à bulles : illustrer leur déroulement sur un exemple de tableau de 5 valeurs par exemple 4 2 6 8 5
2) Calculer la complexité en nombre de comparaisons pour *bulle1*.
3) On peut voir sur l'exemple précédent que cet algorithme peut effectuer des parcours sans échange. Une première amélioration consiste à éviter ces parcours inutiles. *Modifier la fonction bulle1* en introduisant un booléen. Calculer la nouvelle complexité de *bulle1* dans le meilleur des cas et dans le pire des cas.
4) On peut aussi mémoriser le plus grand indice à partir duquel les échanges ne se font plus, c'est-à-dire l'indice en dessous duquel la liste est triée. Par exemple 8 5 2 6 4

Modifier *bulle1* en ajoutant une variable entière supplémentaire. Comment se modifie la complexité ?

5) Enfin on remarquera une certaine dissymétrie dans l'exécution de l'algorithme de tri à bulles sur les 2 listes suivantes : 4 5 6 8 2 et 8 2 4 5 6

Dans les deux cas, un seul élément n'est pas à sa place, mais la complexité n'est pas la même : calculer la.

Pour remédier à cet inconvénient, on peut alterner la direction des parcours consécutifs.

Ecrire une procédure de tri basée sur le principe du tri à bulles qui prenne en compte ces 3 améliorations.

Un tel tri s'appelle **tri shaker.**

Exercice 5 : Insertion et tri

Soit une liste de N valeurs entières représentées linéairement par un tableau de type *Tab* .

1) Voici 3 algorithmes (3 fonctions) permettant d'insérer à sa place la $(I+1)^{ème}$ valeur *t[i]* dans les $(I+1)^{ères}$ positions du tableau t supposé trié entre 0 et i-1

```
void decaler  (Tab t , int k, int j )     /*   décale d'1 position les positions k à j de t    */
void inser1 ( Tab t, int i ) {  while ( i >0 &&  t[ i-1 ] > t[ i ] )  { perm ( &t[ i ], &t[ i-1]  );  i --;  }  }
void inser2 ( Tab t, int i ) { int k= i-1, z= t[i]; while ( i >=0 &&  z< t[ i] ) i--; decaler ( t, k, i) ; t[k] = t[i] ; }
void inser3 ( Tab t, int i ) { int g=0, d= i, k, z; while ( g<d ) { k=(g+d)/2 ; if ( t[i] < t[k] ) d = k ; else g = k ; }
                               z= t[ i ]; decaler ( t, k ,i) ; t[ k] = z ; }
```

Donner les complexités minimale et maximale en nombre de comparaisons pour chaque fonction.

2) En utilisant le sous-programme 1 ou 2 ou 3, donner un algorithme récursif et un autre itératif de tri par insertion . Calculer les complexités extrémales en nombre de comparaisons.
3) Au lieu d'insérer chaque élément à sa place dans la sous-liste triée de tous les éléments qui le précèdent, on l'insère dans une sous-liste d'éléments qui le précèdent, distants d'un certain incrément *k*

```
void inser4 (Tab t, int i, int k) {  while ( i - k >= 0 && t[ i - k ] > t[ i] ) { perm(&t[ i],&t[ i - k ] ); i= i - k ;  } }
```

Illustrer le déroulement de *inser4* avec i=4 et k=4 sur l'exemple 6 8 1 4 3 7 2 9

Evaluer les complexités extrémales de *inser4* en fonction de k et i.

4) Une méthode de tri par incrément successif ou tri shell s'appuie sur cette insertion avec un incrément k (ayant une valeur initiale = n/2) que l'on diminue au cours de passages successifs sur la liste.

```
void trishell (Tab t, int n) { int  k= n/2, i= k;  while ( k>0) { while ( i<n ) inser4 ( t, i++, k);   k = k / 2 ;  } }
```

Evaluer les complexités extrémales de ***trishell*** en fonction de n.