**1.** Donner un exemple d'algorithme dont la complexité spatiale vérifie une relation $t(n) = t(n-1) + t(n-2) + f(n)$ On suppose que $f$ est positive croissante d'ordre polynomial. Chercher une estimation asymptotique de $t$.

**2.** Étudier la complexité temporelle de la recherche par dichotomie. En faire l'étude asymptotique.

**3.** Résoudre la récurrence $u_n = 4u_{n/2} + n^2$ pour $n = 2^k$.

Solution proposée
On pose $v_k = u_{2^k}$ et on a $v_k = 4v_{k-1} + 2^{2k}$, d'où $u_{2^k} = v_k = (u_1 + k)2^{2k}$.

**4.** Résoudre asymptotiquement les complexités des relations de récurrence :

**1°** $t(n) = 2t(\lceil n/2 \rceil) + n \log_2 n$

**2°** $t(n) = 2t(\lfloor (n-1)/2 \rfloor) + n$

**5.** On définit une suite $(u_n)$ par les relations :

$$u_0 = 1, \qquad u_n = u_{\lfloor n/2 \rfloor} + u_{\lfloor n/3 \rfloor} \quad \text{pour } n \geqslant 1.$$

Les deux algorithmes suivants calculent $u_n$ :

```
let rec u_1(n) =
  if n = 0 then 1 else u_1(n/2) + u_1(n/3)
;;

let u_2(n) =
  let v = make_vect (n+1) 0 in
  v.(0) <- 1;
  for i=1 to n do v.(i) <- v.(i/2) + v.(i/3) done;
  v.(n)
;;
```

Comparer les complexités spatiales et temporelles de ces deux fonctions Caml.

**6.** Résoudre asymptotiquement les relations de récurrence :

$$t(n) = 5t(n-1) + 3n + 2 \quad t(0) = 7$$
$$c(n) = 7c(E(n/2)) + O(n^2).$$

(le $O(\cdot)$ est supposé être une fonction croissante).

Solution proposée
On pose $u(n) = t(n)/5^n$ d'où $t(n) = 5^n[7 + \sum_{k=1}^{n} \frac{3k+2}{5^k}] = O(5^n)$.
On résout pour $n = 2^p$. On pose $v(k) = \frac{c(2^k)}{2^{k \log_2 7}}$. On trouve $c(n) = \Theta(n^{\log_2 7})$.

**7.** Montrer l'assertion suivante :

*Pour $A, B \leqslant N$, le nombre de divisions euclidiennes effectuées pour calculer le PGCD de $A$ et $B$ par l'algorithme d'Euclide est majoré par $\log_\phi(N) + 2$, où $\phi = \dfrac{1+\sqrt{5}}{2}$. Le pire des cas est obtenu lorsque $A$ et $B$ sont des nombres de Fibonacci consécutifs.*

Indication : on pourra choisir un nombre d'étapes $e$ et chercher quelle est la plus petite valeur $N_e$ pour laquelle on peut trouver deux entiers $A, B \leqslant N_e$ pour lesquels le calcul du pgcd nécessite $e$ divisions.

**8.** On considère la suite $f$ de Dijkstra définie par :

$$f(0) = 0,\ f(1) = 1,\ f(2k) = f(k),\ f(2k+1) = f(k) + f(k+1).$$

En considérant une suite auxiliaire à valeurs dans $\mathbb{N}^2$, déterminer un algorithme récursif de calcul de $f(n)$ de complexité logarithmique.

Indications

Définition récursive brute :

```
let rec suite1 n =
     match n with
         |  0 -> 0
         |  1 -> 1
         |  _ -> if (n mod 2)=0 then suite1 (n/2)
                 else let m = n/2 in  suite1 m + suite1 (m+1);;
```

Supposons l'existence d'une fonction `suite2` telle que

$$suite2\ k = \begin{pmatrix} f_{2k} \\ f_{2k+1} \end{pmatrix}$$

qui ne soit programmée que par un appel récursif à `suite2 (k/2)`. Sa complexité sera ainsi logarithmique.
Voyons son cahier des charges
- si k est pair (k=2p) `suite2 k` doit retourner

$$\begin{pmatrix} f_{4p} \\ f_{4p+1} \end{pmatrix}$$

sachant que par l'appel récursif elle connaît

$$suite2(k/2) = suite2(p) = \begin{pmatrix} f_{2p} \\ f_{2p+1} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

dans ce cas `suite2` retourne

$$\begin{pmatrix} f_{4p} \\ f_{4p+1} \end{pmatrix} = \begin{pmatrix} f_{2p} \\ f_{2p} + f_{2p+1} \end{pmatrix} = \begin{pmatrix} x \\ x + y \end{pmatrix}$$

- si k est impair (k=2p+1) `suite2 k` doit retourner

$$\begin{pmatrix} f_{4p+2} \\ f_{4p+3} \end{pmatrix}$$

sachant que par l'appel récursif elle connaît toujours

$$suite2(k/2) = suite2(p) = \begin{pmatrix} f_{2p} \\ f_{2p+1} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

Cette fois ci `suite2` retourne

$$\begin{aligned} \begin{pmatrix} f_{4p+2} \\ f_{4p+3} \end{pmatrix} &= \begin{pmatrix} f_{2p+1} \\ f_{2p+1} + f_{2p+2} \end{pmatrix} = \begin{pmatrix} f_{2p+1} \\ f_{2p+1} + f_{p+1} \end{pmatrix} \\ &= \begin{pmatrix} f_{2p+1} \\ f_{2p+1} + f_{2p+1} - f_p \end{pmatrix} = \begin{pmatrix} f_{2p+1} \\ f_{2p+1} + f_{2p+1} - f_{2p} \end{pmatrix} = \begin{pmatrix} y \\ 2y - x \end{pmatrix} \end{aligned}$$

Voici la traduction Caml de ces considérations :

```
let suite2 n =                (* en se ramenant a une suite vectorielle *)
    let rec suitevect p =
              if p=0 then (0,1)
              else let (x,y) = suitevect (p/2) in
                   if (p mod 2) =0 then (x,x+y) else (y,2*y-x)
    in let (u,v) = suitevect (n/2) in
       if (n mod 2)=0 then u else v;;

(* suite1 4321, suite2 4321;;   *->   int * int = 85, 85 *)
```

**9.** Vous vous trouvez face à un mur, trop haut et trop lisse pour que vous puissiez l'escalader. À gauche comme à droite, le mur semble continuer jusqu'à l'infini. Vous savez qu'une porte est percée dans ce mur, mais vous ne savez pas si elle est à votre gauche ou à votre droite, vous ne connaissez même pas la distance $d$ à laquelle elle se trouve de vous ; et la porte est à peine visible, si bien qu'il faut être juste devant elle pour la voir. Décrire une stratégie vous permettant de trouver cette porte, tout en parcourant une distance $\mathcal{O}(d)$.

Solution proposée

Parcourir un km vers la gauche, revenir au point de départ ; parcourir deux km vers la droite, revenir au point de départ ; et ainsi de suite en doublant à chaque fois la distance. Soit $n$ tel que $2^n < d \leqslant 2^{n+1}$ ; au pire, on devra parcourir $2(1 + 2 + \ldots + 2^{n+1}) + d = 2^{n+3} - 2 + d < 9d$.

**10.** Soit $n \geqslant 1$ un entier. Le but de l'exercice est d'évaluer le nombre de multiplications requises pour calculer $a^n$ connaissant $a$. Ainsi, on peut calculer $a^{10}$ en 4 multiplications de la façon suivante : $a.a = a^2$, $a^2.a = a^3$, $a^3.a^2 = a^5$, $a^5.a^5 = a^{10}$. On note $P(n)$ le nombre minimal de multiplications requises pour calculer la puissance n-ième d'un nombre et $B(n)$ le nombre de bits égaux à 1 dans la représentation de $n$ en base 2.

**1°** Établir les inégalités :
- (1) $P(nm) \leqslant P(n) + P(m)$,
- (2) $\lceil \log_2 n \rceil \leqslant P(n) \leqslant \lfloor \log_2 n \rfloor + B(n) - 1$

**2° a.** En déduire les valeurs de $P(n)$ pour $n$ différent de 7 et inférieur ou égal à 10 en justifiant votre calcul.

**b.** Montrer que $P(15) \leqslant 5$. En déduire que l'inégalité $P(n) \leqslant \lfloor \log_2 n \rfloor + B(n) - 1$ n'est pas optimale.

**11.** Que fait la fonction Caml suivante :

```
let rec f = function
 | [] -> failwith "liste vide"
 | [t] -> (t,t)
 | t::q -> let (a,b) = f q in
    if t<a then (t,b) else
    if t>b then (a,t) else (a,b);;
```

Prouvez votre réponse, étudiez la complexité de cette fonction. Qu'en pensez vous ?

Solution proposée

Cette fonction calcule le couple (minimum, maximum) d'une liste. On prouve le résultat par récurrence immédiate sur la longueur de la liste considérée.
La complexité de cette fonction en nombre de comparaisons vérifie $c(0) = c(1) = 0$ et $c(n) \leqslant 2 + c(n-1)$. On en déduit que $c(n) = O(n)$ (plus précisément $\forall\ n \geqslant 2$, $c(n) \leqslant 2(n-1)$).
La complexité linéaire était attendue (il faut de toute façon lire toute la liste). La complexité est ici inférieure à celle de l'algorithme naïf qui agit en deux passes. Il existe des raffinements de cet algorithme pour faire diminuer les constantes (cf. cours).

**12.** La fonction Caml suivante utilise une méthode *diviser pour régner* en vue de calculer la plus petite et la plus grande composante d'un vecteur d'entiers :

```
let minmax t =
 let rec minmax_rec t i j =
     if i=j then (t.(i),t.(i))
            else if i+1=j then begin
                                  if t.(i)<t.(j) then (t.(i),t.(j))
                                                 else (t.(j),t.(i))
                               end
                          else let k=(i+j)/2 in
                               let (min1,max1) = minmax_rec t i k
                               and (min2,max2) = minmax_rec t (k+1) j in
                               ((min min1 min2),(max max1 max2))
 in minmax_rec t 0 (vect_length t - 1);;
```

Justifier la validité de cette fonction puis, notant $C_n$ le nombre de comparaisons entre éléments de `t` effectuées lorsque la fonction est appliquée à un vecteur de $n$ éléments, déterminer des constantes $a$ et $b$ optimales telles que :

$$\forall n \geqslant 2 : an \leqslant C_n + 2 \leqslant bn$$

Solution proposée

La suite $(C_n)_{n\geqslant 1}$ est définie par :

$$C_1 = 0, C_2 = 1,\ C_{2n} = 2C_n + 2 \text{ pour } n \geqslant 2, \text{ et } C_{2n+1} = C_n + C_{n+1} + 2 \text{ pour } n \geqslant 1$$

En posant $D_n = C_n + 2$, on a les relations plus simples :

$$D_1 = 2, D_2 = 3,\ D_{2n} = 2D_n \text{ pour } n \geqslant 2, \text{ et } D_{2n+1} = D_n + D_{n+1} \text{ pour } n \geqslant 1$$

Restent à trouver les constantes $a$ et $b$ optimales pour l'encadrement $an \leqslant D_n \leqslant bn$.

$(D_n)$ est une suite croissante comme on le vérifie sans peine par récurrence.
L'examen de la suite de terme général $\frac{D_n}{n}$ (avec Maple par exemple) suggère les valeurs $a = \frac{3}{2}$ et $b = \frac{5}{3}$.
Le cas minimal semblant être atteint pour $(D_{2^n})$. Le calcul fournit $D_{2^n} = 3 \cdot 2^{n-1}$, $\forall\ n \geqslant 1$. On vérifie ensuite par récurrence (forte[1]) que $\forall\ n \geqslant 2$, $D_n \geqslant \dfrac{3}{2}n$.

Ainsi $a = 3/2$ convient.
Pour $b$, il semblerait que le maximum, $\dfrac{5}{3}$, soit atteint pour $n = 3 \cdot 2^n$, $\forall\ n \geqslant 0$.

Il suffit alors de le vérifier en 2 temps :
– $D_{3\cdot 2^n} = 5 \cdot 2^n$,
– $\forall\ n \geqslant 2$, $D_n \leqslant \dfrac{5}{3}n$.
La première assertion est immédiate puisque $D_{3\cdot 2^n} = 2D_{3\cdot 2^{n-1}} = \ldots = 2^n \cdot D_3 = 5 \cdot 2^n$ ; la seconde se vérifiant alors par récurrence (”forte” à nouveau).
Pour info, voici le code Maple

```
u := proc(n)
local k;
option remember;
    if n = 1 then 2
    elif n = 2 then 3
    else
```

[1] J'entends par récurrence ”forte” la proposition :
$\big(p(0)$ et $(\forall\ n \in \mathbb{N},\ [\forall k \leqslant n,\ p(k)]) \Longrightarrow p(n+1)\big) \Longrightarrow (\forall\ n \in \mathbb{N},\ p(n))$.
alors que dans la preuve par récurrence usuelle, on suppose seulement $p(n)$ pour montrer $p(n+1)$.

```
        k := iquo(n, 2);
        if type(n, odd) then u(k) + u(k + 1)
        else 2*u(k)
        end if
    end if
end proc

> for i to 30 do i,‘    ‘, (u(i)/i), ‘    ‘, evalf(u(i)/i) od;
                    1,     , 2,     , 2.
                    2,     , 3/2,     , 1.500000000
                    3,     , 5/3,     , 1.666666667
                    4,     , 3/2,     , 1.500000000
                    5,     , 8/5,     , 1.600000000
                    6,     , 5/3,     , 1.666666667
                    7,     , 11/7,     , 1.571428571
                    8,     , 3/2,     , 1.500000000
                    9,     , 14/9,     , 1.555555556
                    10,     , 8/5,     , 1.600000000
                              18
                    11,     , --,     , 1.636363636
                              11
                    12,     , 5/3,     , 1.666666667
                              21
                    13,     , --,     , 1.615384615
                              13
                   14,     , 11/7,     , 1.571428571
                              23
                    15,     , --,     , 1.533333333
                              15
                    16,     , 3/2,     , 1.500000000
                              26
                    17,     , --,     , 1.529411765
                              17
```

**13.** Deuxième plus grand élément.
Soit V un vecteur d'entiers de taille $n \geqslant 1$. On cherche à déterminer l'indice du deuxième plus grand élément de V (celui qui viendrait en deuxième position si on rangeait les éléments de V par ordre décroissant).

**1°** Proposer une solution naïve en Caml à ce problème. Quel nombre de comparaisons entre entiers de V effectue cette fonction ?

**2°** Pour imaginer une meilleure solution, penser à un tournoi de Tennis. Le deuxième meilleur joueur n'est pas forcément le finaliste mais figure parmi les adversaires du gagnant.
En déduire une nouvelle solution et analyser sa complexité.

**14.** *Recherche linéaire du k-ième plus petit élément d'une liste*

**1°** *Méthode élémentaire.*
On se donne une liste non triée d'éléments d'un ensemble ordonné (nous travaillerons sur des entiers) dont il s'agit de chercher le $k^{\text{ième}}$ élément dans l'ordre croissant.
Proposer en Caml un algorithme naïf permettant d'atteindre ce but.
En préciser la complexité en nombre de comparaisons.

**2°** *Une méthode linéaire*
Supposons la liste partagée en deux morceaux autour d'un pivot, les éléments plus petits que le pivot d'une part ($liste_1$ de taille $n_1$) et les éléments plus grand (strictement) d'autre part ($liste_2$ de taille $n_2$).
Si $n_1 > k$, on applique la procédure à la première liste sinon, on applique la procédure à la deuxième liste en remplaçant $k$ par $(k - n_1)$.
La procédure est donc récursive.
Écrire une fonction `Decoupe liste pivot` qui prend en argument une liste et un pivot et renvoie le quadruplet $(liste_1, n_1, liste_2, n_2)$

**3°** Bien sûr, l'algorithme tournera d'autant plus vite que l'on aura choisi le pivot proche de l'élément médian[2] de la liste, mais il faut le trouver lui aussi de façon linéaire.
Une bonne idée consiste à découper la liste de départ en paquets de 5 éléments (que l'on peut trier en temps constant) puis à appliquer récursivement la méthode de recherche du pivot à la liste des médians de ces sous-listes.

**a.** Écrire en Caml l'algorithme de tri par insertion.

**b.** Écrire une fonction `Quintuplifie liste` dont le résultat est la liste des éléments de `liste` groupés en quintuplets (un peu moins peut-être pour le dernier paquet).

**c.** Écrire une fonction `Medians liste` qui aprés avoir trié toutes les sous listes de longueur 5 (on utilisera un `map`) renvoie la liste de leurs éléments médians (*i.e.* les 3èmes éléments des quintuplets triés).
Si le dernier paquet n'a pas 5 éléments, on se contentera de renvoyer son premier élément.

**d.** Écrire une fonction `nth n liste` qui retourne le $n^{\text{ième}}$ élément d'une liste quelconque.

**e.** L'algorithme récursif est alors le suivant :

```
let rec nth_lin\'{e}aire  i liste=
    let choisit_pivot l n = nth_lin\'{e}aire ((n+4)/10) (M\'{e}dians l) in
    let n=list_length liste in
    if i>n then failwith "liste trop courte"
           else if n<=5 then nth i (Tri_insertion liste)
                else let l1,n1,l2,n2=D\'{e}coupe liste (choisit_pivot liste n) in
    if i<n1 then nth_lin\'{e}aire i l1
            else nth_lin\'{e}aire (i-n1) l2;;
```

Commenter ce code puis montrer que la récurrence vérifiée par le coût $T(n)$ de l'algorithme vérifie :

$$T(n) \leqslant T(n/5) + T(7n/10) + c \cdot n$$

et en déduire que cet algorithme est bien linéaire.

Solution proposée

**15.** En considérant qu'il y a 256 caractères possibles on veut comparer 2 textes composés de $n$ caractères. On les compare jusqu'à épuiser une chaine ou trouver 2 caractères différents. Majorer le nombre moyen de comparaisons à effectuer pour comparer 2 textes de longueur au plus $n$.

**16.** **1°** Quelle est la complexité du calcul d'un déterminant d'ordre $n$, développé récursivement suivant la première colonne ?

**2°** Quelle est cette complexité temporelle si on mémorise tous les déterminants mineurs, afin d'eviter de les calculer plusieurs fois ? Quelle est alors la complexité spatiale ?

**17.** La multiplication classique de deux matrices $2 \times 2$ requiert 8 multiplications et 4 additions scalaires, la multiplication de deux matrices $n \times n$ nécessitant $\mathcal{O}(n^3)$ opérations arithmétiques. Le nombre de multiplications peut être réduit à l'aide du paradigme *diviser pour régner*.

[2] Celui qui réalise $|liste_1| = |liste_2|$ à 1 près.

**1°** L'algorithme de Strassen repose sur la décomposition des deux matrices à multiplier en 4 blocs :

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \quad N = \begin{pmatrix} E & F \\ G & H \end{pmatrix},$$

La matrice produit vaut $MN = \begin{pmatrix} -P_2 + P_4 + P_5 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 - P_3 + P_5 - P_7 \end{pmatrix}$ avec :

$$\begin{aligned} P_1 &= A(F-H), & P_2 &= (A+B)H, \\ P_3 &= (C+D)E, & P_4 &= D(G-E), \\ P_5 &= (A+D)(E+H), & P_6 &= (B-D)(G+H), \\ P_7 &= (A-C)(E+F). \end{aligned}$$

Quel est le nombre d'opérations arithmétiques effectuées ?

**2°** Pour multiplier des matrices $n \times n$, on applique l'égalité précédente *récursivement* (attention, la multiplication de matrices n'est pas commutative !) (on peut supposer que $n$ est une puissance de deux). On note $c(n)$ le coût pour effectuer la multiplication de deux matrices $n \times n$. Exprimer $c(n)$ en fonction de $c(\frac{n}{2})$.

**3°** En déduire $c(n)$. Estimer le gain par rapport à la méthode naïve.

## 18. Multiplication Rapide.

On considère deux polynômes $P = \sum_{i=0}^{n} a_i X^i$ et $Q = \sum_{j=0}^{m} b_j X^j$ de $\mathbb{Q}[X]$. On note $N = max(\deg P, \deg Q) + 1$ (ainsi $N$ désigne un majorant de la place mémoire allouée par polynôme).

**1°** Le produit de ces deux polynômes peut se faire suivant l'algorithme naïf :

$$P \cdot Q = \sum_{i=0}^{N} \left( \sum_{j=0}^{N} (a_i \times b_j) X^{i+j} \right)$$

Quelle est la complexité $C(N)$ de cet algorithme en nombre de multiplications élémentaires $\times$ ?

**2°** Soit $M = \lceil N/2 \rceil$ (partie entière supérieure). On écrit chacun des deux polynômes sous la forme

$$P = P_1 + X^M P_2 \qquad Q = Q_1 + X^M Q_2$$

avec $P_1$, $P_2$, $Q_1$, $Q_2$ quatre polynômes de degrés inférieurs strictement à $M$.

**a.** On a alors

$$P \cdot Q = P_1 \cdot Q_1 + X^M (P_1 \cdot Q_2 + P_2 \cdot Q_1) + X^{2M} P_2 \cdot Q_2$$

On note $C'(N)$ le nombre de multiplications élémentaires nécessaires au calcul du produit sous cette forme. $C'(1) = 1$. Vérifier que

$$C'(N) = 4C'(\lceil N/2 \rceil) + O(N)$$

(on considère en effet que les coûts "autres" sont un $O(N)$). En déduire $C'(N)$. Commentaires ?

**b.** On écrit

$$P \cdot Q = R_1 + X^M (R_2 - R_1 - R_3) + X^{2M} R_3$$

avec $R_1 = P_1 \cdot Q_1$, $R_2 = (P_1 + P_2) \cdot (Q_1 + Q_2)$ et $R_3 = P_2 \cdot Q_2$. Déterminer la nouvelle complexité $C''(N)$.

**3°** On désire appliquer ce qui précède au produit de grands entiers écrits en base $B$ (algorithme de Karatsuba).

$$n = aB^m + b \qquad m = cB^m + d$$

**a.** Que se passe-t-il qui ne se produisait pas avec le produit de deux polynômes ?

**b.** Ceci change-t-il la complexité en nombre de multiplications élémentaires ?

Solution proposée

**1°** Le calcul du produit naïf nécessite deux boucles for imbriquées (une par $\sum$) et le nombre de multiplications élémentaires est $(degP+1)(degQ+1)$ soit $O(N^2)$.

**2° a.** La résolution de cette équation de récurrence est un classique du cours : on trouve $C'(N) = O(N^{\log_2 4}) = O(N^2)$. On n'a donc rien gagné en ordre de grandeur par rapport à la méthode brutale.

**Remarque(s).** Comme c'est la première fois dans cette épreuve, vous avez tout intérêt à redémontrer le résultat du cours. Il faudra donc
- commencer par signaler que la fonction $C'(N)$ est croissante en fonction de $N$.
- supposer donc dans un premier temps que $N = 2^p$
- poser $u_p = C'(2^p)$ et déterminer $u_p$
- enfin conclure dans le cas général pour $N$ par monotonie (*i.e.* en encadrant).

**b.** Cette fois ci $C''$ vérifie :

$$C''(N) = 3C''(\lceil N/2 \rceil) + O(N)$$

On obtient donc $C''(N) = O(N^{\log_2 3})$ et $\log_2 3 \simeq 1,58$.

**3°** Dans le cas d'entiers, apparaissent les retenues qui n'existaient pas au niveau des polynômes. Cela ne change pas le nombre de multiplications élémentaires mais complique quelque peu la vie au programmeur.
Notons finalement que si on compare dans la pratique la multiplication naïve des grands entiers avec la multiplication de Karatsuba, le gain en complexité ne se fait sentier que pour de très grands entiers...

## 19.

On suppose connue une fonction `ppcm2` qui calcule le ppcm de 2 entiers. On veut généraliser cette fonction au calcul du ppcm de $n$ entiers. Proposer une solution Caml itérative et une solution Caml récursive à l'aide d'une méthode "diviser pour régner". Comparer les performances des deux approches.

## 20. *Analyse en moyenne d'un algorithme.*

**1°** Donner un algorithme qui trouve le plus petit et le plus grand élément d'un tableau contenant $n$ entiers naturels en effectuant un nombre de comparaisons de l'ordre de $\frac{3n}{2}$ (prouver votre programme et sa complexité, bien sûr !).
*Indication : considérer les éléments du tableau 2 par 2 et commencer par comparer 2 éléments successifs.*

**2°** Un étudiant a proposé la solution suivante :

```
let minmax  v =
  let min = ref 0 and max = ref 0 in
  for i=1 to vect_length(v)-1 do
     if v.(i) < v.(!min) then min:= i
     else if v.(i) > v.(!max) then max:=i done; (!min,!max);;
```

Ainsi le nombre de comparaisons effectuées pour chaque $i$ est soit 1 soit 2. L'étudiant se demande si sa solution bien que fausse dans le cas le pire ne serait pas correcte en moyenne.
Le but de cet exercice est de montrer qu'il n'en est rien. Pour ce faire on retient le modèle des permutations et au lieu d'examiner les tableaux à trier on se penche sur la permutation associée, c'est à dire que l'on suppose que l'algorithme est exécuté sur un tableau $\alpha$ contenant tous les entiers naturels entre 1 et $n$, et que toutes les permutations de ces nombres sont équiprobables.

**a.** Pour toute permutation $\alpha$ on on appelle minimum local un entier $j$ tel que :

$$\forall\, i \in [\![1, j[\![,\ \alpha(i) > \alpha(j)$$

On convient que 1 est toujours un minimum local de $\alpha$. Exprimer le nombre de comparaisons effectuées par l'algorithme précédent en fonction du nombre $k$ de minima locaux de la permutation $\alpha$.

**b.** On note par $u_{n,k}$ le nombre de permutations sur $n$ éléments qui présentent $k$ minima locaux. Montrez que :

$$u_{n,0} = 0 \qquad u_{n,n} = 1$$

$$\forall\, k \in [\![1, n[\![ \quad u_{n,k} = (n-1)u_{n-1,k} + u_{n-1,k-1}$$

**c.** On considère la famille de polynômes

$$P_n(x) = \sum_{k=1}^{\infty} u_{n,k} x^k$$

Donnez une expression de $P_n(x)$ comme un produit de monômes du premier degré. Montrez que la dérivée $P'_n(x)$ satisfait

$$P'_n(x) = P_n(x)\left(\frac{1}{x} + \frac{1}{x+1} + \ldots + \frac{1}{x+n-1}\right)$$

**d.** Donnez une expression du nombre moyen $M_n$ de minima locaux dans les permutations sur $n$ éléments.

**e.** Quel est le comportement à l'infini du nombre moyen de comparaisons dans le programme de l'étudiant ; le comparer à $\frac{3n}{2}$.

Solution proposée

**1°** On suggère

```
let minmax v =
let n = vect_length v in let mini = ref v.(0) and maxi = ref v.(0) in
for i = 1 to (n/2) do
if v.(2*i) <= v.(2*i-1) then begin
     if v.(2*i) < !mini  then mini := v.(2*i);
     if v.(2*i-1) > !maxi then maxi := v.(2*i-1)
                          end else begin
     if v.(2*i) > !maxi  then maxi := v.(2*i);
     if v.(2*i-1) < !mini then mini := v.(2*i-1)
                                    end
done;
if (n mod 2)=1 then (min !mini v.(n-1),max !maxi v.(n-1))
else (!mini, !maxi);;
```

On vérifie ensuite par induction sur la longueur du vecteur que cette fonction est correcte.
On effectue $3E(n/2)$ comparaisons plus 1 éventuellement si $v$ est de longueur impaire.

**2°** **a.** Pour chaque $i$ entre 1 et n-1 il y a 2 comparaisons si i n'est pas un minimum local et une seule s'il en est un. Comme le nombre de minimum locaux de la permutation qui appartiennent à 1,2, ... n-1 est k-1 le nombre de comparaisons effectuées par l'algorithme vaut :

$$2(n-1-(k-1)) + k - 1 = 2n - k - 1$$

**b.** Toute permutation admet au moins un minimum local ainsi $u_{n,0} = 0$. Pour qu'une permutation $a_1$, $a_2$, ... $a_n$ présente $n$ minima locaux il faut que $a_1 > a_2 > \ldots > a_n$ il n'y en a donc qu'une seule ainsi $u_{n,n} = 1$.
Soit $\alpha$ une permutation sur $n$ éléments qui présente $k$ minima locaux. En supprimant $n$ de la suite, on obtient une permutation $\beta$ sur $n-1$ éléments. Celle-ci présente $k-1$ minima locaux si $n$ se trouvait en tête et $k$ minima sinon. Réciproquement étant donnée $\beta$ il y a $n-1$ façons d'insérer $n$ autrement qu'en tête et une seule de l'insérer en tête pour obtenir $\alpha$. Ainsi :

$$u_{n,k} = u_{n-1,k-1} + (n-1)u_{n-1,k}$$

**c.** On multiplie par $x^k$ l'égalité ci-dessus et on somme ces égalités pour $k = 1, 2...n$, on obtient

$$P_n = xP_{n-1} + (n-1)P_{n-1} = (x+n-1)P_{n-1}$$

Comme $P_1 = x$ on obtient $P_n = \prod_{i=0}^{n-1}(x+i)$ d'où l'expression de la dérivée.

**d.** Le nombre moyen $M_n$ de minima locaux dans l'ensemble des permutations sur $n$ éléments vaut :

$$M_n = \frac{1}{n!}\sum_{k=1}^{n} k u_{n,k} = \frac{1}{n!}P'_n(1) = \frac{n!}{n!}\sum_{k=1}^{n}\frac{1}{k}$$

**e.** Le nombre moyen de comparaisons de l'algorithme proposé est ainsi de $2n - ln(n)$ prépondérant devant $3n/2$.

**21.** Soit $x_1$, $x_2$, ..., $x_n$ une suite finie de nombre entiers non tous nécessairement distincts. La "multiplicité" d'une valeur $x$ dans la suite est égale au nombre de fois où $x$ apparaît dans la suite. Un entier $z$ est dit "valeur majoritaire" si sa multiplicité est supérieure ou égale à $n/2 + 1$.

**1°** Montrer que si $x_i$ est différent de $x_j$ et si l'on supprime $x_i$ et $x_j$ de la suite, la valeur majoritaire de la suite initiale, s'il en existe une, est aussi valeur majoritaire de la suite obtenue après suppression.

**2°** Montrer qu'en examinant successivement les éléments de la suite dans l'ordre $x_1$, $x_2$, ..., $x_n$, on peut "mettre à jour" deux variables $C$ et $M$ ayant la propriété suivante :
lorsque l'on considère $x_i$, $C$ contient une valeur qui est la seule candidate possible à être valeur majoritaire parmi $x_1$, $x_2$, ..., $x_{i-1}$ et $M$ contient le nombre de fois où la valeur $C$ est apparue jusqu'alors, si l'on exclue les fois où $C$ a été éliminé.

**3°** En déduire un algorithme qui, en deux "parcours" de la suite $x_1$, $x_2$, ..., $x_n$ détermine si la suite possède ou non une valeur majoritaire et donne cette valeur majoritaire quand elle existe.

Solution proposée

**1°** On considère une liste d'entiers $L$ de longueur $n$ (implémentée par exemple sous la forme d'un vecteur) qui a (ou non) une valeur majoritaire $m$.
Considérons les deux premiers éléments de la liste $x_1$ et $x_2$ et notons $L'$ la liste $L$ privée de $x_1$ et $x_2$ (de longueur $n-2$). Dans cette question on traite le cas où $x_1 \neq x_2$.
Si $x_1$ et $x_2$ sont tous les deux différents de $m$, on peut les supprimer de la liste : la nouvelle liste $L'$ a toujours $m$ pour valeur majoritaire ($m$ est même encore plus majoritaire !).
Si un des deux vaut $m$, le nombre d'occurrences de $m$ dans $L'$ a diminué de 1 mais la nouvelle longueur est $n-2$ donc $m$ est encore majoritaire dans $L'$.
Conclusion : en supprimant les deux éléments de tête d'une liste dans le cas ou ceux-ci sont différents, on ne change pas la valeur majoritaire d'une liste, si elle existe. Remarquons (en prenant par exemple la liste [1 ;2 ;3]) que cette suppression peut conduire à l'apparition d'une valeur majoritaire dans L' (ici 3) alors qu'il n'y en avait pas dans $L$.

**2°** $C$ est ainsi le candidat pour être une valeur majoritaire de la liste $L$ dont on a lu les $i$ premiers éléments. On initialise $C$ et $M$ avec $x_0$ et 1. On procède par examen successif des éléments de la suite. Quand on considère $x_i$ on le compare à $C$ et on ajoute ou on enlève 1 à $M$ selon que $x_i$ est égal ou non à $C$. Lorsque $M$ devient nul c'est qu'il n'y avait pas de valeur majoritaire dans le début de la liste et on donne alors à $C$ la valeur $x_{i+1}$ et à $M$ la valeur 1.
À la fin du parcours, il ne reste qu'un candidat $C$. On fait alors un deuxième parcours de la suite pour déterminer son nombre d'occurrences dans $L$ et décider si oui ou non c'est bien une valeur majoritaire.
La complexité de notre algorithme est ainsi bien linéaire.

```
(* valeur majoritaire *)


let x=[|1;2;5;1;3;2;4;7;6;4;2;2;0|];;

let y=[|1;2;5;2;2;2;4;2;6;4;2;2;0|];;

let candidat_val_maj x =
   let C= ref x.(0) and M = ref 1 in
   for i =1 to (vect_length x) -1 do
   if x.(i) = !C then M:=!M+ 1
   else M:= !M-1;
   if !M=0 then begin C:=x.(i) ; M:=1 end
   done;
   !C;;

let valmaj x =
```

```
    let C = candidat_val_maj x in
    let M = ref 0 in
    for i =0 to (vect_length x) -1 do
    if x.(i) = C then M:= !M+1
    done;
    if !M>((vect_length x) / 2) then C else -1;;


valmaj x;;

valmaj y;;
```

**22.** Décomposition en tranches d'un ensemble de points.
On considère un ensemble $E$ fini de points du plan qui n'ont jamais ni la même abscisse ni la même ordonnée. On ordonne cet ensemble par une relation de domination (ordre partiel) : Pour deux points $P$ et $Q$ du plan de coordonnées respectives $(x, y)$ et $(x', y')$, on dit que $P$ domine $Q$ et on note $P \succ Q$ si $x > x'$ et $y > y'$. On dit qu'un point est maximal s'il n'est dominé par aucun point.
On considère une partition de $E$ en des sous-ensembles $T_1, T_2, \ldots, T_k$ telle que :
– $T_1$ est est l'ensemble des points maximaux de $E$,
– pour $i > 1$, $T_i$ est l'ensemble des points maximaux de $E \setminus \{T_1, T_2, \ldots, T_{i-1}\}$.
On appelera une telle partition une décomposition en tranche.
On appelera pivot d'une tranche $T_i$, le point de $T_i$, dont l'abscisse est la plus petite.

**1°** Montrer que le pivot d'une tranche est aussi son point d'ordonnée la plus grande.

**2°** Soient $y_1, y_2, \ldots, y_k$ les ordonnées des pivots des tranches $T_1, T_2, \ldots, T_k$ respectivement. Montrer que :

$$y_1 > y_2 > \ldots > y_{k-1} > y_k$$

Soit $E$ un ensemble de points dont on a calculé la décomposition en tranches $E = T_1 \cup T_2 \cup \ldots \cup T_k$ dont les pivots ont pour ordonnées $y_1, y_2, \ldots, y_k$. Soit $Q = (x, y)$ un point du plan dont l'abscisse est strictement plus petite que les abscisses de tous les points de $E$ et dont l'ordonnée n'est égale à aucune des ordonnées des points de $E$.

**3°** Montrez que si $y > y_1$ alors $Q$ est un point maximal de $E' = E \cup \{Q\}$. Montrez qu'il existe un point de $T_j$ qui domine $Q$ si et seulement si $y < y_j$.

**4°** Dans quel cas $E'$ a-t-il une tranche de plus que $E$ ?

**5°** En déduire un algorithme qui détermine la décomposition en tranches de $E'$ connaissant celle de $E$ et les ordonnées des pivots de $E$.

**6°** Comment réaliser alors la décomposition en tranches d'un ensemble $E$ donné ?

**23.** Les deux points les plus proches
Le but de ce problème est d'identifier dans un ensemble de points quel est le couple de points les plus proches au sens de la distance euclidienne. Ce type d'algorithme voit son utilité dans les transports aériens ou maritimes.
L'ensemble de points est vu comme un vecteur T de points de taille N. Un point `T.(i)` est identifié par ses coordonnées dans le plan : `fst T.(i)` l'abscisse et `snd T.(I)` l'ordonnée.
Nous disposons d'une fonction `sqrt(a)` qui renvoie la racine carrée de a (`a ** 0.5`).

**1°** Donner un algorithme itératif qui détermine le couple de points les plus proches de l'ensemble T[0 ... N-1]. Prouver votre algorithme.
Évaluer le nombre d'additions, de multiplications et d'appels de la fonction sqrt.

L'ensemble de points est doorénavant stocké dans **deux** vecteurs de taille N : X où les points sont ordonnés suivant les abscisses croissantes et Y où les points sont ordonnés suivant les ordonnées croissantes.

**2°** Écrire une fonction `decoupe X Y Yp deb fin` qui place dans Yp[deb...med] les points de X[deb...med] classés par ordonnées croissantes et dans Yp[med + 1...fin] les points de X[med + 1... fin] classés par ordonnées croissantes, où med est la partie entière de $\frac{deb+fin}{2}$. On a donc coupé suivant les abcisses notre ensemble de points en deux moitiés où les points sont rangés suivant leur ordonnée.
Évaluer le nombre de transferts effectués lors de l'utilisation de cette fonction (il doit être linéaire !).

**3°** On suppose que A et B sont les points les plus proche de X[deb...med] et, C et D sont les points les plus proches de X[med + 1 ... fin].
Ecrire un algorithme linéaire qui détermine les deux points les plus proches de l'ensemble X [deb ... fin] connaissant A, B, C, D et Y[deb ... fin]. (on montrera que l'on peut se limiter à considérer les points d'une bande médiane, où chaque point est considéré avec au plus 7 autres points de la bande).

**4°** En déduire une fonction `cherche` récursive qui répond à notre problème avec une complexité $\Theta(n \ln n)$.

**24.** On considère le "jeu des chiffres" dans lequel on donne une suite $w[1]$, $w[2]$, .... $w[n]$ de $n$ nombres entiers (`int` de Caml) et un nombre s. On demande de trouver un sous-ensemble $I$ des indices tel que $\sum_{i\in I} w[i] = s$.
Donner un algorithme résolvant ce problème. Quelle est sa complexité ?

Indications

Proposer une solution récursive utilisant le fait qu'un algorithme $A(s,k)$ (qui résout le problème de somme $s$ en utilisant $w[k]$, ..., $w[n]$) vérifie $A(s,k) = A(s,k+1)$ ou $A(s-w[k],k+1)$ suivant que dans la solution on utilise ou non $w[k]$.

Solution proposée

```
let w=[|2;4;8;12;25;7|];;

let rec chiffre s k w n=
 if s=0 then true
 else if (s<0) || (k>=n) then false
 else if chiffre (s-w.(k)) (k+1) w n then  begin print_int w.(k); true end
 else chiffre s (k+1) w n;;

let jeu s w = chiffre s 0 w (vect_length w);;
```

Le cas le pire est obtenu lorsqu'il n'y a pas de solution ce qui entraine que tout l'arbre des possibilités est parcouru (rq. : une branche de l'arbre conduit à une partie de $w$). Dans le programme ci-avant, `chiffre (s-w.(k)) (k+1) w n` retourne false et on évalue forcément `chiffre s (k+1) w n`. Ainsi, en notant $c_k$ le nombre d'appels à `chiffre s k w n`, on a :

$$c_k \leqslant 2c_{k+1} \quad \text{et} \quad c_n = 1$$

D'où $c_k \leqslant 2^{n-k}$ et la complexité exponentielle dans le cas le pire attendue.
Voici une version plus claire pour utiliser `trace` sur `chiffre`.

```
let rec chiffre (s,l)   =
 try
 if s=0 then true
 else if (s<0) then false
 else if chiffre ((s-(hd l)) , (tl l)) then  begin print_int (hd l); true end
 else chiffre (s,(tl l))
 with Failure "tl" -> false;;



let jeu s l =  chiffre (s,l);;
```

**25.** À chaque exécution de la procédure Hanoï on associe une chaîne de caractères de la façon suivante :
– Un déplacement d'une rondelle du piquet 1 vers le piquet 3 est noté par le symbole a,
– du piquet 1 vers le piquet 2 par le symbole b
– et du piquet 2 vers le piquet 3 par c.
Les déplacements inverses sont notés par les mêmes lettres avec une barre dessus ($\bar{a}$, $\bar{b}$, $\bar{c}$). Ainsi la chaîne de caractères correspondant au déplacement de deux rondelles du piquet 1 vers le piquet 3 selon l'algorithme décrit en cours est : $h2 = bac$. Celle correspondant au déplacement de trois rondelles du piquet 1 vers le piquet 3 est :

$$h3 = ab\bar{c}a\bar{b}ca$$

**1°** Donner la chaîne correspondant au déplacement de 4 rondelles.

**2°** Quelle est la longueur de la chaîne h, correspondant au déplacement de n rondelles ?

**3°** En s'aidant des transformations suivantes sur les caractères donnez un procédé de calcul de la chaîne $h_{n+1}$ en connaissant $h_n$ :

| | $a$ | $b$ | $c$ | $\bar{a}$ | $\bar{b}$ | $\bar{c}$ |
|---|---|---|---|---|---|---|
| $\Phi$ | $b$ | $a$ | $\bar{c}$ | $\bar{b}$ | $\bar{a}$ | $c$ |
| $\Psi$ | $c$ | $\bar{b}$ | $a$ | $\bar{c}$ | $b$ | $\bar{a}$ |

**4°** Montrez que lorsque l'on supprime les barres au dessus des caractères intervenant dans la chaîne $h_n$ on obtient une chaîne très régulière.

**5°** En déduire une procédure itérative permettant de résoudre le problème des tours de Hanoï.

**26.** Soit $P$ un polygône à $n$ sommets notés dans l'ordre $A_i$, $i = 0 \dots n-1$, leurs coordonnées étant $(x_i, y_i)$ (les indices se comprennent modulo $n$).
On note $p = \sum_{i=1}^{n-1} d(A_{i-1}, A_i)$ le périmètre de $P$. Le but est de déterminer le couple $(A_{i_0}, A_{j_0})$ découpant $P$ en deux parties de longueurs les plus semblables possibles. Ce couple est réalisé par :

$$\inf_{(i,j)} \left| \sum_{k=i}^{j-1} d(A_k, A_{k+1}) - \sum_{k=j}^{i+n-1} d(A_k, A_{k+1}) \right|$$

On suppose donnés `n: int` et `P : (int * int) vect` tel que $P.(i)$ contienne les coordonnées de $A_i$.

**1°** La force brute.

**a.** Écrire une fonction `dist_plus` renvoyant

$$dist\_plus(i,j) = \sum_{k=i}^{j-1} d(A_k, A_{k+1})$$

et `dist_moins` renvoyant

$$dist\_moins(i,j) = \sum_{k=j}^{i+n-1} d(A_k, A_{k+1})$$

en convenant que $dist\_plus(A_i, A_i) = 0.0$ et $dist\_moins(A_i, A_i) = p$.

**b.** Écrire un programme utilisant les fonctions précédentes et renvoyant une matrice $M = [M_{i,j}]$ de taille $n \times n$ avec

$$M_{i,j} = \left| \sum_{k=i}^{j-1} d(A_k, A_{k+1}) - \sum_{k=j}^{i+n-1} d(A_k, A_{k+1}) \right|$$

et conclure à un algorithme permettant de trouver la paire $(i_0, j_0)$. Quelle est l'ordre de grandeur de sa complexité (en appels à $d$) ?

**2°** Amélioration de l'algorithme.

**a.** Écrire une fonction qui à $i$ donné renvoit Opposé(i), ce dernier étant le plus grand $j \geqslant i$ tel que

$$\sum_{k=i}^{j-1} d(A_k, A_{k+1}) \leqslant \frac{p}{2}$$

Écrire un programme renvoyant un vecteur J et un vecteur D tel que J.(i) = Opposé(i) mod n et $D.(i) = d(A_i, A_{Opposé(i)})$.
Quelle est sa complexité ? Peut-on s'arranger pour avoir une complexité en $O(n)$ ? (remarquer que Opposé(i + 1) succède à Opposé(i) mod n).

**b.** Montrer que $j_0 = J.(i_0)$ ou $(J.(i_0) + 1)$ mod n. Conclure à un nouvel algorithme permettant de trouver $(i_0, j_0)$. Quelle est sa complexité ?

Solution proposée

```
(* COUPER LA POIRE EN DEUX *)

let n=6;;
let P=[|0.,2.; 0.,0.; 3.,1. ; 4.,3.; 4.,4. ; 2.,5.|];;


let ind i = i mod n;;
let point i = P.(ind i);;
let norme (x,y) = (sqrt ((x *. x) +. (y *. y)));;

let dist_suiv i =
 let (x,y) = (point i) and (u,v) = (point (i+1)) in
 norme ((x -. u),(y -. v));;

let distance i j =
let d = ref 0.0 in
if (i >= j) then 0.0 else
( for k = i to j-1 do d:= !d +. (dist_suiv k);done; !d; );;

let dist_plus i j =
if (j < i) then (distance i (j+n)) else (distance i j);;

let dist_moins i j = if (i <= j)
then (distance j (i+n)) else (distance j i);;

let M =
let m = (make_matrix n n 0.0) in
for i = 0 to (n-1) do
for j = 0 to (n-1) do
m.(i).(j) <- abs_float ((dist_plus i j) -. (dist_moins i j)); done; done;
m;;

let (i0,j0) =
let point = ref (1,1) and d = ref (dist_moins 1 1) (* soit p *) in
for i = 0 to (n-1) do
for j = i to (n-1) do
if (M.(i).(j) <. !d) then (d:=M.(i).(j); point:=(i,j))
done; done;
!point;;

(* calcul de p *)
let p = let d = ref 0.0 in for i = 0 to (n-1) do d:= !d +. (dist_suiv i);done; !d;;
```

```
let oppose i =
let j = ref i and d = ref 0.0 and psur2 = (p /. 2.0) in while (!d <=. psur2) do
(d:= !d+.(dist_suiv !j);
j:= !j+1;) done; (* j vaut oppose(i) +1 *)
(!j - 1) mod n;;

(* calcul de D et J *)

let D = make_vect n 0.0 and J = make_vect n 0;;

let psur2 = (p /. 2.0) in
while ((D.(0) +. (dist_suiv J.(0)))<=. psur2) do
D.(0) <- D.(0) +. (dist_suiv J.(0));
J.(0) <- (J.(0)+1) mod n;done; (* J.(0) est calcul'e *)

for i = 1 to n-1 do
D.(i) <- D.(i-1) -. (dist_suiv (i-1));
J.(i) <- J.(i-1);
while ((D.(i) +. (dist_suiv J.(i)))<=. psur2) do
D.(i) <- D.(i) +. (dist_suiv J.(i));
J.(i) <- (J.(i)+1) mod n;done; (* J.(i) est calcul'e *)
done;;

D;;J;;


let res = ref (0,0) and d = ref p;;
 (* choix de i0 et j0 *)
for i = 0 to n-1 do
let dd = (p -. 2.0*.D.(i)) in
( if (dd <. !d) then
(res := (i,J.(i)); d:= dd;);
if ((2.0 *. (dist_suiv J.(i)) -. dd) <. !d) then
(res := (i,(J.(i) +1) mod n); d := dd;); );
done;;

res;;
```

**27.** On peut programmer le tri fusion avec un nombre de comparaison qui vérifie la relation

$$C_N = C_{\lfloor n/2 \rfloor} + C_{\lceil n/2 \rceil} + N$$

A partir d'un calcul de $C_{N+1} - C_N$, Montrer que $C_N = \sum_{k=0}^{N-1} (\lfloor \ln_2 k + 2 \rfloor)$
En déduire que $C_N = N\lceil \ln_2 N \rceil + N - 2^{\lceil \ln_2 N \rceil}$

**28.** Le tri des mécanographes.
Le but de cet exercice est d'implémenter en Caml l'algorithme des mécanographes pour trier les cartes perforées. Une carte perforée est représentée par un mot de $N$ lettres. Les lettres sont prises dans un alphabet de cardinal $p$. L'ordre choisi est l'ordre lexicographique.
Soit $S$ la séquence (de longueur $n$) des mots à trier suivant l'ordre croissant. L'algorithme consiste à effectuer sur $S$ successivement $N$ tris partiels ; plus précisément à chaque étape on ne trie les mots que suivant leur i-ième lettre sans modifier l'ordre relatif dans $S$ des mots ayant la même i-ième lettre.
Prenons un exemple avec $p = 3$, $N = 3$ et la suite initiale $S_0$ de 6 mots suivants : 112, 321, 113, 232, 233, 312. L'algorithme procède comme suit :
– on emploie trois corbeilles auxiliaires (une par lettre de l'alphabet) initialement vides $P_1$, $P_2$, $P_3$,

– on place tour à tour chacun des 6 mots dans la corbeille correspondant à sa dernière lettre ; cela donne : $P_1 = \{321\}$, $P_2 = \{112, 232, 312\}$ et $P_3 = \{113, 233\}$.
On récupère ensuite, dans cet ordre, les mots et la nouvelle suite $S_1$ de mots à traiter est ainsi : 321, 112, 232, 312,113, 233.
– on place ensuite chacun des 6 mots dans la corbeille suivant sa deuxième lettre ; cela donne : $P_1 = \{112, 312, 113\}$, $P_2 = \{321\}$ et $P_3 = \{232, 233\}$ (l'ordre relatif dans $S_1$ de deux mots ayant la même deuxième lettre n'a pas été modidié).
La nouvelle suite $S_2$ de mots à traiter est cette fois : 112, 312, 113, 321, 232, 233.
– Enfin on itère le procédé sur la première lettre ; on a : $P_1 = \{112, 113\}$, $P_2 = \{232, 233\}$ et $P_3 = \{312, 321\}$.

Et la suite finale triée est bien : 112, 113, 232, 233, 312, 321.

**1°** Justifier la validité de l'algorithme.

**2°** Proposez des structures de données *ad hoc* pour traiter le problème.

**3°** Donner une version Caml de l'algorithme (on pourra utiliser le type prédéfini `queue`).

**4°** Quel est sa complexité en fonction de $N$ et $n$ ? Ce résultat vous surprend-il ?

Solution proposée

**1°** On peut faire une récurrence sur $N$ :
– Pour des mots d'une lettre, l'algorithme effectue un tri en une itération.
– On suppose qu'après la i-ième étape les mots sont correctement triés dans la suite $S_i$ suivant leurs i dernières lettres. La (i+1)-ième étape classe alors les mots suivant la lettre suivante (la (N-i)-ième) et, puisqu'en cas d'égalité elle ne modifie pas l'ordre des mots de $S_i$, les mots sont correctement rangés suivant leur (i+1) dernières lettres dans $S_{i+1}$.

**2°** On choisit par commodité de représenter chaque mot par un vecteur d'entiers. L'algorithme nécessite de connaitre toute la suite de mots dès le départ : on choisit également une structure vectorielle pour $S$. Chaque corbeille doit recevoir les mots en conservant leur ordre d'arrivée : on choisit donc une structure FIFO, les files (`queue` en Caml).

**3°** Voici une solution Caml :

```
let L= [|  [|1;1;2|] ; [| 3; 2; 1|] ; [|1;1;3|] ; [|2;3;2|] ; [|2;3;3|] ; [|3;1;2|] |];;


let N=3;;

#open "queue";;

let p1 = new ()
and p2 = new ()
and p3 = new ();;
(* les files de rangements *)



let ajoute i e  =
 match e.(i) with
 | 1 -> add e p1
 | 2 -> add e p2
 | _ -> add e p3 ;;


let rec tri_indice i  v =
for j= 0 to ((vect_length v) -1) do
ajoute i v.(j) done;;


let merge v =
```

```
  let lp1 = length p1 and lp2 = length p2 and lp3 = length p3 in
  for i =0 to (lp1-1) do
  v.(i) <- take p1 done;
  for i=lp1 to (lp1 + lp2 -1) do
  v.(i) <- take p2 done;
  for i = (lp1 + lp2) to (lp1 + lp2 +lp3 -1) do
  v.(i) <- take p3 done;;
```

```
let tri_mecano v =
  for i=(N-1) downto 0 do
  tri_indice i v; merge v
  done; v;;
```

**4°** On a vu dans le cours que la complexité optimale en moyenne et dans le cas le pire d'un tri par comparaisons d'une suite de $n$ mots est en $n \log n$. Ici l'exécution d'une passe de tri a un coût linéaire en $n$ et l'on effectue $N$ itérations soit un coût total en $nN$. Ceci n'a rien de surprenant :
- d'une part on sait que si les données à trier ne sont pas quelconques (par exemple évoluant dans un ensemble de cardinal fini connu d'avance), on peut les trier avec un coût linéaire en temps (mais linéaire en espace également),
- d'autre part l'ordre lexicographique à ceci de particulier qu'il ne nécessite pas la connaissance de toutes les lettres pour comparer deux mots.

## 29. *Le tri selon les Pieds Nickelés.*

Les célèbres professeurs Croquignol, Filochard et Ribouldingue ont inventé une toute nouvelle technique de tri, qu'ils ont (subtilement) baptisée *méthode CFR*.
En voici une mise en œuvre en Caml :

```
let \'{e}change u i j = let x = u.(i) in u.(i)<-u.(j);u.(j)<-x;;

let CFR t =
 let rec CFR_aux u i j =
  if i=j then () else
  if i+1=j then begin if u.(i)>u.(j) then \'{e}change u i j end else
  let k=(j+1-i)/3 in
    CFR_aux u i (j-k); CFR_aux u (i+k) j; CFR_aux u i (j-k)
 in CFR_aux t 0 (vect_length t - 1);;
```

**1°** Montrer que le programme précédent est correct, *id est* : il trie bien le tableau transmis en argument, dans l'ordre croissant de ses valeurs.
On se propose d'analyser cette méthode. Soit $C_n$ le coût *maximal* exprimé en nombre de comparaisons d'éléments du tableau, lorsque la taille de celui-ci est $n$.

**2°** Écrire des relations qui permettent de calculer, de proche en proche, les valeurs de $C_n$.

**3°** Compléter le tableau suivant des premières valeurs de $C_n$ :

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $C_n$ | 0 | 1 | 3 | 9 | 27 | ? | ? | ? | 81 | 243 |

**4°** Montrer qu'à partir d'un rang $n_0$ que l'on précisera, on a $C_n \geqslant n^2$.

**5°** A-t-on $\forall\ n \geqslant 1$, $C_n \leqslant n^3$ ?

**6°** Justifier : il n'existe aucun exposant $\alpha$ tel que $C_n$ soit équivalent à $n^\alpha$ lorsque $n$ tend vers l'infini.

**7°** (difficile) exhiber un exposant $\beta$ tel que $C_n = o(n^\gamma)$ pour tout $\gamma > \beta$, et $n^\gamma = o(C_n)$ pour tout $\gamma < \beta$.

Solution proposée

Le tri selon les Pieds Nickelés.

**1°** La fonction trie en place les deux premiers tiers du vecteur, puis les deux derniers et recommence le tri des deux premiers tiers.
Prouvons sa validité par récurrence sur $j - i \geqslant 1$ : pour $j - i = 1$ et $j - i = 2$, c'est clair. Sinon, supposons la fonction correcte pour $j - i \leqslant n$ et prouvons qu'elle l'est encore pour $j - i = n + 1$. Notons $k$ la partie entière de $n/3$ ; on a $3k \leqslant n$, donc $n - 2k \geqslant k$. Après le premier appel récursif, les $k$ plus grand éléments sont dans $t[i+k..j]$ ; après le deuxième appel, ils sont à leur place dans $t[j-k+1..j]$. Enfin, le troisième appel trie $t[i..j-k]$.

**2°** $C_1 = 0$, $C_2 = 1$ et $C_n = 3C_{n-\lfloor n/3 \rfloor}$ pour $n \geqslant 2$.

**3°** À l'aide de la fonction Maple suivante (dopée par l'option `remember`) :

```
c := proc(n) option remember; 3*c(n-iquo(n,3)) end;
c(1) := 0; c(2) := 1;
```

on a déterminé les résultats :

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $C_n$ | 0 | 1 | 3 | 9 | 27 | 27 | 81 | 81 | 81 | 243 |

**4°** En fait, Maple permet de constater que les seules valeurs de $n \leqslant 1000$ telles que $C_n < n^2$ sont 1, 2, 3, 4 et 6. Soit $p \geqslant 3$ tel que $C_n \geqslant n^2$ pour tout $n \in [\![2p, 3p-1]\!]$. Alors :

$$\begin{aligned}
C_{3p} &= 3C_{2p} \geqslant 12p^2 \geqslant (3p)^2 \\
C_{3p+1} &= 3C_{2p+1} \geqslant 3(2p+1)^2 = 12p^2 + 12p + 3 \\
&\geqslant 9p^2 + 6p + 1 = (3p+1)^2 \\
C_{3p+2} &= 3C_{2p+2} \geqslant 3(2p+2)^2 = 12p^2 + 24p + 12 \\
&\geqslant 9p^2 + 12p + 4 = (3p+2)^2
\end{aligned}$$

Donc $C_n \geqslant n^2$ pour $n \in [\![2p, 3p+2]\!]$, et à plus forte raison pour $n \in [\![2p+2, 3p+2]\!] = [\![2(p+1), 3(p+1)-1]\!]$. Comme $C_n \geqslant n^2$ pour $n \in [\![8, 11]\!] = [\![2 \times 4, 3 \times 4 - 1]\!]$, on peut affirmer que $C_n \geqslant n^2$ pour tout $n \geqslant 8$, et du coup pour tout $n \geqslant 7$.

**5°** Oui ! (j'attends vos rédactions subtiles !).

**6°** On établit aisément par récurrence que $C_{n+1}$ est égal à $C_n$ ou à $3C_n$ (ce qui nous assure que la suite est croissante). Il existe une infinité d'indices $n$ tels que $C_{n+1} = 3C_n$ (sinon la suite serait stationnaire, contredisant sa minoration par $n^2$). Ainsi, de la suite de terme général $\frac{C_{n+1}}{C_n}$, on peut extraire une suite qui converge vers 3, interdisant à $C_n$ d'être équivalent à $n^\alpha$. Notons qu'il existe aussi une infinité d'indices $n$ tels que $C_{n+1} = C_n$ (ne serait-ce que $n = 3p + 2$).

**7°** Idée : si $C_n$ était du même ordre de grandeur que $n^\alpha$, en faisant $n = 3p$ on trouverait $(3p)^\alpha$ du même ordre que $3(2p)^\alpha$, imposant $3^\alpha = 3 \cdot 2^\alpha$, soit $\alpha = \frac{\ln 3}{\ln 3/2} \approx 2.71$ ; nous allons établir en toute rigueur que $\ln(C_n)$ est équivalent à $\alpha \ln n$ lorsque $n$ tend vers l'infini.
Un *palier* de la suite est un intervalle $[\![p, q-1]\!]$ sur lequel elle est constante, et maximal vis-à-vis de cette propriété. Notons $x_n$ l'indice de début du $n$ième palier : $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, $x_4 = 4$, $x_5 = 5$, $x_6 = 7$, $x_7 = 10$ et $x_8 = 14$. Soient $p = x_n$ et $q = x_{n+1}$ ; on a $C_p = C_{p+1} = \cdots = C_{q-1} = \frac{C_q}{3}$, et $C_{p-1} = \frac{C_p}{3}$. Distinguons deux cas selon la parité de $p$ : si $p = 2p'$, alors $C_{3p'} = 3C_{2p'} = 3C_p = C_q$ donc $q \leqslant 3p'$ ; $C_{3p'-1} = C_{3(p'-1)+2} = 3C_{2(p'-1)+2} = 3C_{2p'} = 3C_p$, donc $q \leqslant 3p' - 1$ ; $C_{3p'-2} = C_{3(p'-1)+1} = 3C_{2(p'-1)+1} = 3C_{2p'-1} = 3C_{p-1} = C_p$, donc $q = 3p' - 1$. Une discussion analogue donne $q = 3p' + 1$ si $p = 2p' + 1$.
Dans les deux cas, on obtient $q = p + \lfloor (p-1)/2 \rfloor = \lfloor (3p-1)/2 \rfloor$ ; ainsi $x_{n+1} = \lfloor (3x_n - 1)/2 \rfloor$ pour $n \geqslant 3$. On en déduit l'encadrement

$$\frac{3(x_n - 1)}{2} < x_{n+1} \leqslant \frac{3x_n - 1}{2}$$

On en déduit la majoration $x_{n+1} - 1 \leqslant \frac{3}{2}(x_n - 1)$ d'où

$$x_n - 1 \leqslant \left(\frac{3}{2}\right)^{n-3} (x_3 - 1) = \frac{3^{n-3}}{2^{n-4}} \quad \text{soit} \quad x_n \leqslant 1 + \frac{3^{n-3}}{2^{n-4}}$$

De même, on a la minoration $x_{n+1} - 2 > \frac{3}{2}(x_n - 2)$, dont on déduit

$$x_n - 2 > \left(\frac{3}{2}\right)^{n-3}(x_3 - 2) = \left(\frac{3}{2}\right)^{n-3} \quad \text{soit} \quad x_n > 2 + \frac{3^{n-3}}{2^{n-3}}$$

On écrit l'encadrement :

$$\left(\frac{3}{2}\right)^n \left(\frac{8}{27} + \frac{2^{n-1}}{3^n}\right) < x_n \leqslant \left(\frac{3}{2}\right)^n \left(\frac{16}{27} + \frac{2^n}{3^n}\right)$$

dont on déduit, par passage au logarithme, que $\ln C_n$ est équivalent à $n \ln \frac{3}{2}$. Or, pour $n \geqslant 2$, on a $C_p = 3^{n-2}$ pour tout $x_n \leqslant p < x_{n+1}$, soit $\ln x_n \leqslant \ln p < \ln x_{n+1}$. On en déduit que $\ln p$ est lui aussi équivalent à $n \ln \frac{3}{2}$. Mais $\ln C_p = (n-2)\ln 3$ est équivalent à $n \ln 3$. Finalement, $\ln C_p$ est équivalent à $\frac{\ln 3}{\ln 3/2} \ln p = \alpha \ln p$. Le résultat de l'énoncé s'en déduit sans peine.

## 30. Tri par distribution

Le *tri par distribution* sert à trier une suite $(a_1, \ldots, a_n)$ d'entiers, lorsque l'on sait que $0 \leqslant a_i \leqslant m-1$ pour un entier $m$ "petit". Le principe est le suivant : on détermine le nombre d'éléments inférieurs à chaque valeur $j$ par :

$$c_j = \#\{i \mid a_i \leqslant j\} \qquad 0 \leqslant j \leqslant m-1$$

puis, dans une deuxième passe, on s'en sert pour calculer la place finale de $a_i$ par :

$$s_i = c_{a_i} - \#\{k > i \mid a_k = a_i\}$$

La suite triée est $(b_1, \ldots, b_n)$ avec $b_k = a_i$ si et seulement si $k = s_i$.

**1°** Écrire une fonction `distribution a n m` qui réalise ce tri. On peut utiliser un vecteur temporaire c pour calculer les $c_j$ et un tableau $b$ pour ranger la suite triée.

**2°** Donner le temps de calcul de votre algorithme en fonction de $n$ et de $m$, ainsi que l'espace mémoire nécessaire.

Indications

```
let m = 6;;
(* les entiers \‘{a} trier sont < 6 *)

let distribution a  =
 let n = vect_length a in
 let b = make_vect n 0 in
 let c = make_vect m 0 in
 for i=0 to (n-1) do c.(a.(i)) <- c.(a.(i)) + 1 done;
 for j=1 to m-1 do c.(j) <- c.(j-1) + c.(j) done;
 for i=(n-1) downto 0 do
   b.(c.(a.(i))-1) <- a.(i);
   c.(a.(i)) <- c.(a.(i)) -1
 done;
b;;
```

## 31. Quelques compléments sur la fonction d'Ackermann (voir le polycop de cours pour sa définition).

Par commodité d'écriture nous noterons $A(n,p)$ pour `ackermann n p`.

**1°** Donner les expressions explicites de $A(1,p)$, $A(2,p)$ et $A(3,p)$.

**2°** Que vaut $A(4,4)$ ? Comparer cet entier avec $10^{80}$ (qui est une estimation du nombre d'atomes de l'univers !).

**3°** Montrer que $\forall\ (n,p) \in \mathbb{N}^2,\ A(n,p) > p$.

**4°** Montrer que si $q > p$, alors $\forall\ n \in \mathbb{N},\ A(n,q) > A(n,p)$.

**5°** Montrer que $A(n+1,p) > A(n,p)$ pour tout couple $(n,p) \in \mathbb{N}^2$.

**6°** On définit la fonction $\alpha$ de $\mathbb{N}$ dans $\mathbb{N}$ comme l'*inverse fonctionnel* de $A$ : $\alpha(n)$ est le plus petit entier $k$ tel que $A(k,k) \geqslant n$. Montrer que la fonction $\alpha$ est bien définie sur $\mathbb{N}$.

Que peut-on dire de la croissance de $\alpha$ ?

Solution proposée

La fonction d'Ackermann.

**1°** Calcul de $A(1,p)$.

On a $A(1,0) = A(0,1) = 2$ et $A(1,p+1) = A(0,A(1,p)) = A(1,p)+1$ donc par récurrence $A(1,p) = p+2$.

Calcul de $A(2,p)$.

On a $A(2,0) = A(1,1) = 3$ et $A(2,p+1) = A(1,A(2,p)) = A(2,p)+2$ donc par récurrence $A(2,p) = 2p+3$.

Calcul de $A(3,p)$.

On a $A(3,0) = A(2,1) = 5$ et $A(3,p+1) = A(2,A(3,p)) = 2A(3,p)+3$ donc par récurrence $A(3,p) = 2^{p+3} - 3$.

**2°** On a $A(4,4) = 2^{A(4,3)+3} - 3 > 2^{A(4,3)}$ d'où, comme $A(4,0) = 13 > 2$, on a :

$$A(4,4) \gg 2^{\left(2^{\left(2^{\left(2^2\right)}\right)}\right)} = 2^{65536} \gg 10^{80} \quad !$$

Exactement, $A(4,4) = 2^{\left(2^{\left(2^{\left(2^{13}\right)}\right)}\right)} - 3$.

**3°** On va montrer le résultat par induction dans $(\mathbb{N}^2, \preccurlyeq)$ suivant le théorème de correction. Soit $\mathfrak{p}_A$ le prédicat défini pour tout $(n,p) \in \mathbb{N}^2$ par : $\mathfrak{p}_A(n,p) =$ "$A(n,p) > p$".

- Si $n = 0$, alors $A(0,p) = p+1 > p$, soit $\mathfrak{p}_A(0,p)$.
- Par définition, $A(n,0) = A(n-1,1)$. Par hypothèse d'induction, on a $A(n-1,1) > 1$ donc on a bien $A(n,0) = A(n-1,1) > 1 > 0$.
- Par définition, $A(n,p) = A(n-1,A(n,p))$. Par hypothèse d'induction, $A(n,p-1) > p-1$ et $A(n-1,A(n,p-1)) > A(n,p-1)$ donc $A(n,p) = A(n-1,A(n,p-1)) > A(n,p-1) > p-1$ d'où $A(n,p) > p$ (on travaille dans $\mathbb{N}$ !).

On a donc bien montré $\mathfrak{p}_A$ pour tout $(n,p) \in \mathbb{N}^2$.

**4°** Il suffit de montrer que $\forall\ (n,p) \in \mathbb{N}^2,\ A(n,p+1) > A(n,p)$. C'est clair si $n = 0$. Soit $n \in \mathbb{N}^*$, on a, pour tout $p$, d'après la question précédente, $A(n,p+1) = A(n-1,A(n,p)) > A(n,p)$ soit le résultat.

**5°** On va montrer le résultat à nouveau par induction. Soit $\mathfrak{p}_A$ le prédicat défini pour tout $(n,p) \in \mathbb{N}^2$ par : $\mathfrak{p}_A(n,p) =$ "$A(n+1,p) > A(n,p)$".

- Si $n = 0$, alors $A(1,p) = p+2 > A(0,p) = p+1$.
- Par définition, $A(n+1,0) = A(n,1)$. Par hypothèse d'induction, $A(n,1) > A(n-1,1)$ d'où $A(n+1,0) = A(n,1) > A(n,0) = A(n-1,1)$.
- Par définition, $A(n,p) = A(n-1,A(n,p))$. Par hypothèse d'induction, $A(n+1,p-1) > A(n,p-1)$ d'où, d'après la question précédente, $A(n-1,A(n+1,p-1)) > A(n-1,A(n,p-1))$. Et, par induction on peut supposer $A(n,A(n+1,p-1)) > A(n-1,A(n+1,p-1))$ et on en déduit bien

$$A(n+1,p) = A(n,A(n+1,p-1)) > A(n-1,A(n,p-1)) = A(n,p)$$

On a donc bien montré $\mathfrak{p}_A(n,p)$ pour tout $(n,p) \in \mathbb{N}^2$.

**6°** Pour vérifier la définition de $\alpha$, il suffit de montrer que la suite $(A(n,n))_{n\in\mathbb{N}}$ tend vers $+\infty$. On a $A(0,0)=1$, $A(1,1)=3$, $A(2,2)=7$, $A(3,3)=61$ et $A(4,4)=...$ Pour $n\geqslant 3$, en utilisant les questions précédentes, $A(n+1,n+1)=A(n,A(n+1,n))>A(n+1,n)>A(n,n)$, d'où la stricte croissance de la suite et le résultat s'en déduit.
La fonction $\alpha$ est évidemment une fonction à la croissance très lente. Elle intervient notamment dans le calcul de la complexité du problème *Set-Union-Find* (union et recherche dans les ensembles). On connaît en effet un algorithme d'union d'ensembles [3] "quasi-linéaire" (de complexité $\mathcal{O}(n\,\alpha(n))$).
On pourra se reporter au problème de Mathématiques appliquées et Algorithmique posé en 1995 à l'ENS de Lyon (attention, pour simplifier les calculs, l'auteur a choisi une définition de Ackermann légèrement différente de la définition usuelle).

[3] On emploie aussi le vocabulaire de "fusion de classes d'équivalence".