Les deux parties sont indépendantes et peuvent être traitées dans un ordre quelconque.

Partie I - Algorithmes pour la sélection

Pour $x \in \mathbb{R}$, on note $\lfloor x \rfloor$ la partie entière de x, c'est-à-dire le plus grand entier inférieur ou égal à x, et $\lceil x \rceil$ le plus petit entier supérieur ou égal à x.

On s'intéresse au problème suivant, appelé *problème de la sélection* : on a un ensemble E de n entiers positifs distincts, un entier i inférieur ou égal à n, et on recherche le i-ème élément de E classé dans l'ordre croissant, c'est-à-dire l'élément x de E tel qu'il y ait $i-1$ éléments de E strictement inférieurs à x et $n-i$ strictement supérieurs à x.

Le *médian* d'un ensemble de n nombres est le $\lceil n/2 \rceil$-ème lorsqu'ils sont rangés en ordre croissant.

On insiste sur le fait que dans toute la partie I, on considère des tableaux ou listes d'entiers ***distincts deux à deux***.

I.A - Recherche des deux plus grands éléments

Les éléments de E sont donnés non classés dans un tableau T.

I.A.1) Soit $p \in \mathbb{N}^*$. On organise un *tournoi* entre $n = 2^p$ entiers $e_1, \ldots, e_n$ au sens suivant : on constitue un arbre binaire parfait de hauteur p (i.e. : chaque noeud de profondeur $< p$ possède exactement deux fils), dont les 2^p feuilles sont étiquetées par les e_i. Ensuite, pour h allant de $p-1$ à 0, les étiquettes des noeuds de profondeur h sont égales au maximum des étiquettes de leurs deux fils.

a) Donner le nombre de comparaisons nécessaires pour réaliser un tel tournoi.

b) Montrer qu'à l'aide de ce procédé, on peut trouver le maximum et le second plus grand élément des e_i en moins de $n + \log_2 n$ comparaisons.

En fait, on peut montrer (mais ce n'est pas demandé) qu'un tel algorithme est optimal.

c) Exemple : appliquer la méthode précédente à l'entrée :

$T = [3, 6, 8, 20, 2001, 1515, 17, 1]$.

d) Comment généraliser cette méthode lorsque n n'est pas de la forme 2^p ? Appliquer à l'entrée

$T = [1, 1024, 1515, 5, 12, 4]$.

I.A.2) On suppose qu'il existe un algorithme $\mathscr{A}$ prenant en entrée n entiers, et retournant le plus grand de ces entiers. On suppose que cet algorithme exécute des comparaisons entre des éléments du tableau, et que le résultat retourné ne dépend que de l'ensemble des résultats des comparaisons effectuées. On suppose également qu'il existe une entrée $e = (e_1, \ldots e_n)$ telle que $\mathscr{A}$ exécute strictement moins de $n-1$ comparaisons, lorsqu'il est exécuté sur l'entrée e.

a) Si S est un ensemble fini non vide de cardinal $p \geq 2$ dont les éléments sont appelés *sommets* $S = \{s_1, \ldots, s_p\}$, on appelle *arête* de S toute paire de sommets, c'est-à-dire tout ensemble $\{s_i, s_j\}$ de deux sommets distincts. Par définition, un *graphe (non orienté)* est un couple (S, A) où A est un ensemble d'arêtes de S.

Un tel graphe est dit *connexe* lorsque pour toute paire de sommets $\{s_i, s_j\}$, il existe un entier $n \in \mathbb{N}^*$ et une suite de sommets $c_0, \ldots, c_n$ tels que $c_0 = s_i$, $c_n = s_j$ et $\{c_k, c_{k+1}\} \in A$ pour tout $k \in [\![0, n-1]\!]$.

Montrer que si (S, A) est connexe, alors $|A| \geq |S| - 1$ (où $|X|$ désigne le cardinal de l'ensemble X).

b) Construire une entrée $e' = (e'_1, \ldots, e'_n)$ telle que l'algorithme $\mathscr{A}$ ne retourne pas le plus grand élément de e'. On pourra considérer le graphe $([\![1, n]\!], C)$, où C est l'ensemble des paires $\{i, j\} \in [\![1, n]\!]^2$ tels que dans l'exécution de $\mathscr{A}$ sur e, e_i a été comparé au moins une fois à e_j.

c) Conclure.

I.B - Recherche systématique dans un tableau

Les éléments de E sont à nouveau donnés non classés dans un tableau T.

I.B.1) On utilise l'algorithme suivant : en supposant $i \leq n/2$, on recherche le minimum du tableau, puis le deuxième plus petit élément, puis le troisième et ainsi de suite jusqu'au i-ème.

Calculer le nombre de comparaisons effectuées ; quel est son maximum lorsque i varie ?

I.B.2) Indiquer un algorithme qui permettrait de résoudre le problème de la sélection pour un i quelconque en $O(n \ln n)$ comparaisons.

I.C - Tri rapide dans une liste chaînée

I.C.1) Écrire une fonction (ou une procédure) `partition` qui, étant donné une liste chaînée d'entiers `liste`, et un entier `pivot` (appartenant ou non à la liste) :

- constitue deux listes `avant` et `apres` constituées des éléments de `liste` et telles que : $\forall (x, y) \in$ `avant` $\times$ `apres`, $x \leq$ `pivot` $< y$.
- retourne les deux listes `avant` et `apres`, et le nombre d'éléments `nb` de la liste `avant`.

Les candidats rédigeant en Caml devront écrire une fonction de type :

```
partition : 'a liste —>'a —> 'a list * 'a list * int = <fun>
```

Ceux rédigeant en Pascal utiliseront la déclaration de type

```
type
    ptrliste=^element;
    element=record
        entier:integer;
        suivant:ptrliste
      end;
```

et ils écriront une procédure déclarée de la façon suivante :

```
procedure partition(liste:ptrliste; pivot:integer;
                    var avant,apres:ptrliste; var taille:integer);
```

I.C.2)

a) Écrire une fonction recherche récursive qui, pour un entier i et une liste d'entiers *liste*, retourne le i-ème élément de la *liste*.

On utilisera la fonction/procédure partition précédente, en prenant comme pivot le premier élément de la liste. Les candidats ayant choisi Caml écriront une fonction de signature :

```
recherche : int—>'a list—>'a = <fun>
```

Ceux ayant choisi Pascal écriront une fonction déclarée :

```
fonction recherche (liste:ptrliste; i:integer):integer;
```

b) Justifier la terminaison et la correction de la fonction précédente.

c) Donner des indications sur son temps de calcul. On exhibera des « cas limites ».

d) Si la distribution des entiers de E est uniforme dans un intervalle, comment choisir l'entier *pivot* pour réduire au mieux le temps de calcul ?

I.D - Recherche de l'élément médian

L' ensemble des éléments de E est donné sous la forme d'un tableau T d'entiers de longueur n indexé de 0 à $n-1$. Dans cette partie du problème, on supposera pour simplifier que n est une puissance de 2. On pose $n = 2^p$ et $m = n/2 = 2^{p-1}$.

On suppose que les deux sous-tableaux $T[0...m-1]$ et $T[m...n-1]$ sont triés par ordre croissant, donc que :

$$T_0 < T_1 < \ldots\ldots < T_{m-1} \text{ et } T_m < T_{m+1} < \ldots < T_{n-1}.$$

I.D.1)

a) En utilisant la fonction (supposée donnée) de fusion de deux tableaux triés, donner un algorithme déterminant le médian μ de T.

b) Évaluer son temps de calcul (on prendra en compte seulement le nombre de comparaisons de deux éléments).

I.D.2) On veut améliorer la recherche précédente en effectuant une recherche dichotomique simultanée sur les deux moitiés du tableau T.

a) Expliquer (éventuellement à l'aide de schémas) une stratégie à adopter pour déterminer le médian de T.

b) Donner le code d'une telle fonction. On introduira une fonction récursive de la forme recherche_dicho ($T, g1, d1, g2, t2$).

On rappelle que la taille du tableau initial est une puissance de 2.

c) Prouver la validité du programme précédent.

d) Donner un ordre de grandeur de son temps de calcul (on prendra en compte seulement le nombre de comparaisons de deux éléments).

I.E - Utilisation d'arbres binaires de recherche

Afin de pouvoir insérer puis rechercher rapidement n'importe quel élément de l'ensemble E, on organise les données selon une structure d'arbres binaires de recherche. On utilisera les types suivants :

En Caml :

```
type arbre = vide | Noeud of Nd
   and Nd = {valeur : int; mutable taille : int;
             mutable gauche : arbre; mutable droit : arbre};;
```

En Pascal :

```
type
    ptrNoeud=^Noeud;
    Noeud = record
              valeur, taille : integer;
              gauche,  droit : ptrNoeud;
            end;
```

Chaque noeud N de la structure d'arbres binaires contient la valeur d'un entier de E, la taille (comptée en nombre de noeuds non vides) de l'arbre de racine N, et des pointeurs vers les deux fils.

I.E.1) Pour tester le fonctionnement du programme, on utilise des nombres entiers engendrés par un processus pseudo-aléatoire, qui sont rangés, l'un à la suite de l'autre, dans l'arbre binaire initialement vide.

a) On considère la suite de nombres pseudo-aléatoires 3; 4; 1; 2. Donner un schéma de la structure de l'arbre binaire obtenu après insertion de ces 4 éléments dans un arbre initialement vide. Pour chaque noeud, on précisera la valeur de l'étiquette, et la taille.

b) Même question avec la séquence suivante :

7; 3; 5; 11; 13; 12; 14; 9; 4; 6; 1; 8; 10; 0; 2

I.E.2) Écrire une fonction (ou une procédure) insere, récursive, réalisant l'insertion d'un nouvel entier x dans l'arbre binaire a. Cette fonction/procédure devra retourner l'arbre dans lequel on a inséré l'entier x. Dans la mesure du possible, on modifiera les champs de l'arbre donné en paramètre, plutôt que d'en recréer un.

En Caml, la signature de insere sera :

```
insere : int -> arbre -> arbre = <fun>
```

En Pascal, la déclaration sera :

```
procedure insere (x:integer; var a : PtrNoeud)
```

I.E.3)

a) En supposant que l'arbre est bien équilibré (en un sens que l'on précisera), indiquer un ordre de grandeur du temps d'exécution de la fonction `insere` en fonction de la taille de l'arbre dans lequel on insère un nouvel élément.

b) Que se passe-t-il si l'arbre n'est pas équilibré ? Citer un cas où cela peut se produire.

I.E.4) Écrire une fonction recherche, récursive, déterminant le k-ème élément de l'ensemble des étiquettes d'un arbre binaire de recherche.

Si on appelle g la taille du sous-arbre gauche, on pourra considérer divers cas suivant la valeur de g par rapport à k.

En Caml, on aura :

```
recherche : int -> arbre -> int = <fun>
```

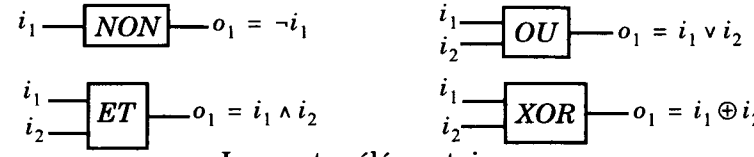
Et en Pascal :

```
function recherche (k:integer; a:PtrNoeud):integer;
```

Partie II - Portes logiques universelles

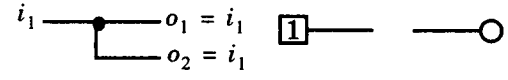
Rappels - notations - définitions

- *$\mathscr{B}$ désigne l'ensemble des booléens : $\mathscr{B} = \{0,1\} = \{Faux, Vrai\}$ avec l'analogie $0 = Faux$ et $1 = Vrai$.*
- *Les fonctions logiques sont les applications de $\mathscr{B}^k$ dans $\mathscr{B}^n$, avec $k, n \in \mathbb{N}^*$.*
- *Les formules logiques sont construites à partir de variables, des connecteurs $\vee$ (ou), $\wedge$ (et), $\neg(non)$ et $\oplus(ou\ exclusif)$, ainsi éventuellement que des constantes $Vrai$ et $Faux$.*
- *On rappelle qu'à toute formule ϕ construite sur les variables $(v_1, ..., v_n)$, on associe de façon naturelle une fonction $F_\phi : \mathscr{B}^n \to \mathscr{B}$.*
  Réciproquement, si f est une application de $\mathscr{B}^k$ dans $\mathscr{B}$, on sait qu'il existe une formule ϕ construite avec les variables $(v_1, ..., v_k)$ telle que $f = F_\phi$. Il n'y a pas unicité de ϕ, mais on peut par exemple imposer à ϕ de n'utiliser que les connecteurs $\wedge$, $\vee$ et $\neg$. On dit que l'ensemble $\{\wedge, \vee$ et $\neg\}$ est complet.
- *Présentons sommairement les circuits logiques : il s'agit d'assemblages orientés de portes logiques élémentaires disposant d'entrées et de sorties.*
  — *Une porte (n,p) est constituée d'entrées $(i_1, ..., i_n)$ et de sorties $(o_1, ..., o_p)$; chaque o_i prend une valeur fonction des i_j. Traditionnellement, les entrées sont représentées à gauche et les sorties à droite.*
  — *On dispose des portes élémentaires suivantes : la porte NON est de type $(1,1)$, avec $o_1 = \neg i_1$. Les portes OU, ET et XOR sont de type $(2,1)$, avec $o_1 = i_1 \vee i_2$ (resp. $o_1 = i_1 \wedge i_2$ et $o_1 = i_1 \oplus i_2$).*



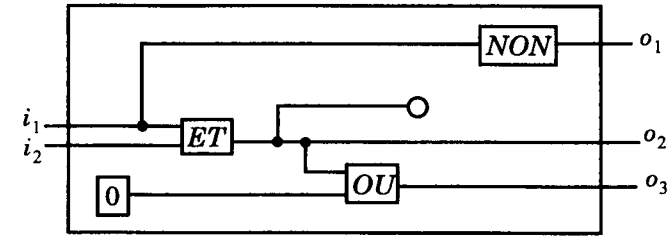
Les portes élémentaires

  — *Chaque entrée de porte peut constituer une entrée du circuit global, ou bien être reliée à la sortie d'une autre porte, ou encore à une source, qui constitue une porte particulière sans entrée, avec une sortie constante égale à 0 ou 1.*
  — *Chaque sortie de porte peut constituer une sortie du circuit global, ou bien être reliée à l'entrée d'une autre porte, ou encore à un puits, qui est une porte particulière sans sortie, avec une entrée : en somme, il s'agit d'une sortie « dont on ne tiendra pas compte ».*
  — *L'assemblage est acyclique au sens où en suivant l'orientation des portes, il n'existe pas de boucle dans le circuit.*
  — *Les duplicateurs sont des portes particulières à une entrée i_1 et deux sorties en o_1 et o_2 égales à i_1. Ils sont représentés par un cercle plein. Les sources sont représentées par un carré et les puits par des cercles.*



Un duplicateur, une source de 1, et un puits.

  — *Un circuit C avec n entrées $(i_1, ..., i_n)$ et k sorties $(o_1, ..., o_k)$ calcule de façon naturelle une fonction $F_C : \mathscr{B}^n \to \mathscr{B}^k$. Dans l'exemple qui suit, C_0 calcule la fonction $(i_1, i_2) \mapsto (\neg i_1, i_1 \wedge i_2, (i_1 \wedge i_2) \vee 0)$*



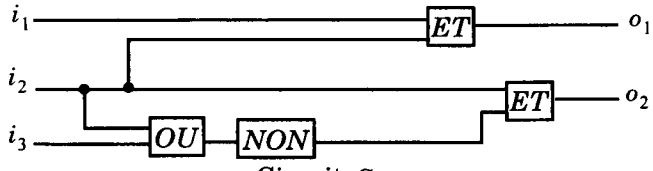
Circuit C_0

  — *Deux circuits seront dits équivalents lorsqu'ils calculent la même fonction.*

— *Un ensemble E de portes sera dit universel lorsque tout circuit est équivalent à un autre circuit constitué de portes de E, de duplicateurs, de sources et de puits. Lorsque E est réduit à un singleton $\{P\}$, on dit que P est une porte universelle.*

— *Enfin, une porte P de type (n,p) sera dite réversible lorsqu'il existe une porte Q de type (p,n) telle que l'assemblage constitué de la porte P suivie de la porte Q calcule la fonction identité de $\mathscr{B}^n$. Par exemple, la porte NON est réversible (prendre $Q = P$).*

II.A - Donner deux formules logiques calculant les valeurs respectives de o_1, o_2 en fonction de i_1, i_2 et i_3 pour le circuit C_1 suivant :



Circuit C_1

II.B - Ensembles de portes universelles

II.B.1) Construire un circuit logique C_2 permettant de calculer la fonction

$$\varphi : \mathscr{B}^3 \to \mathscr{B}^3,\ (i_1, i_2, i_3) \mapsto (i_1, i_1 \oplus i_3, i_2 \wedge i_3).$$

On utilisera uniquement des duplicateurs, sources, portes NON, OU et ET.

II.B.2)

a) Montrer que l'ensemble de portes {NON, ET} est universel.

b) Exemple : construire un circuit équivalent à C_2 à l'aide des seules portes NON et ET, et de duplicateurs et sources.

II.B.3)

a) Montrer que l'ensemble de portes {XOR, OU } est universel.

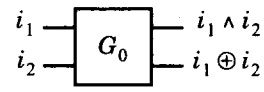
b) Exemple : construire un circuit équivalent à C_0 à l'aide des seules portes XOR et OU, et de duplicateurs et sources.

II.B.4)

a) Montrer que la porte G_0 suivante est universelle :

b) Construire à partir de G_0 (et de duplicateurs, sources et puits) un circuit C_3 calculant la fonction de $\mathscr{B}^3$ dans $\mathscr{B}^2$:

$$(i_1, i_2, i_3) \mapsto (i_1 \vee i_2, (\neg i_1) \wedge (i_2 \vee i_3)).$$



Porte G_0

c) Montrer que la porte G_0 n'est pas réversible.

II.C - Portes réversibles

II.C.1) Montrer que si une porte logique (n, p) est réversible, alors $n \leq p$.

II.C.2) Quelles sont les portes $(1, 1)$ réversibles ?

II.C.3) Donner deux exemples simples mais non triviaux (i.e. dont les fonctions associées ne sont pas l'identité) de portes $(2, 2)$ réversibles.

II.C.4) Combien existe-t-il de portes $(2, 2)$ réversibles ? Et de portes (n, p) réversibles ? (il s'agit bien sûr de portes non équivalentes...).

II.D - Portes réversibles universelles

II.D.1) On se propose de montrer qu'il n'existe pas de porte $(2, 2)$ réversible et universelle.

Soit G une porte réversible $(2, 2)$.

On note g la fonction de $\mathscr{B}^2$ dans $\mathscr{B}^2$ calculée par G.

a) Montrer que g est affine au sens suivant :

$$\forall i, i' \in \mathscr{B}^2 \times \mathscr{B}^2,\ g(i + i') = g(i) + g(i') + g(0),$$

où $+$ désigne la somme dans $\mathscr{B}^2$ (assimilé au groupe additif $V = (\mathbb{Z}/2\mathbb{Z})^2$ et 0 désigne $(0, 0)$). On pourra commencer par le cas où $i = i'$, puis le cas où i ou i' vaut 0.

On rappelle que si $x \in V$, on a $x + x = 0$, et que si x, y, z sont les trois éléments non nuls de V, alors $x = y + z$.

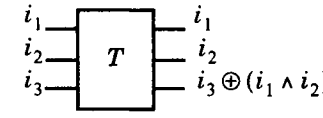
b) Montrer que tout circuit (n, p) construit à partir de G, de duplicateurs, sources et puits calcule une fonction affine de $\mathscr{B}^n$ dans $\mathscr{B}^p$ (pour la définition d'une fonction affine se reporter au II.D.1.a) en remplaçant $\mathscr{B}^2$ par $\mathscr{B}^n$).

c) Conclure.

II.D.2) La porte de Toffoli est la porte logique $(3, 3)$ suivante :

a) Montrer que la porte de Toffoli est réversible et universelle.

b) Vérifier que la fonction calculée par T n'est pas affine.



Porte de Toffoli

••• FIN •••