ÉCOLE POLYTECHNIQUE **FILIÈRE MP**
OPTION INFORMATIQUE

CONCOURS D'ADMISSION 2004

COMPOSITION D'INFORMATIQUE

(Durée : 4 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

⋆ ⋆ ⋆

Médians et Convexité

On attachera une grande importance à la concision, à la clarté, et à la précision de la rédaction. Les deux problèmes sont indépendants.

Premier problème : Sélection

Un *médian* d'un ensemble $X = \{e_1, e_2, \dots e_n\}$ de n nombres entiers distincts est un nombre e dans X tel que les nombres d'éléments strictement plus petits et strictement plus grands que e dans X diffèrent d'au plus de 1 ($n > 0$). Si n est impair, le médian est unique ; si n est pair, il y a deux médians possibles.

Le problème de la *sélection* consiste à trouver l'élément de rang k dans X, c'est-à-dire l'élément e se trouvant en k-ième position quand X est trié en ordre croissant ($1 \leq k \leq n$).

Nous supposerons l'ensemble X représenté par la liste de ses éléments, c'est-à-dire par le type `ensemble` défini par :

(* *Caml* *)

```
type ensemble == int list;;
```

{ *Pascal* }

```
type
  ensemble = ^cellule;
  cellule = record contenu:integer;
    suivant:ensemble; end;
```

En Pascal, la liste vide est `nil` et l'on pourra utiliser la fonction suivante pour construire les listes :

```
function cons(x:integer; s:ensemble) : ensemble;
var r:ensemble;
begin new(r); r^.contenu := x; r^.suivant := s; cons := r end;
```

Cette fonction est applicable pour construire les listes du type `ensemble`.

Question 1. Écrire la fonction `card` qui prend un ensemble en argument et retourne son nombre d'éléments.

(* *Caml* *)	{ *Pascal* }
`card : ensemble -> int`	`function card(`X`:ensemble) : integer;`

Quelle est la complexité en temps de cette fonction par rapport à n?

Soit p un entier quelconque, on sera amené à partitionner l'ensemble X en éléments plus grands, égaux ou plus petits que p.

Question 2. Écrire la fonction `nPetits` qui prend en arguments un entier p et l'ensemble X ; et qui retourne le nombre d'éléments strictement inférieurs à p dans X.

(* *Caml* *)	{ *Pascal* }
`nPetits : int -> ensemble -> int`	`function nPetits` `(`p`:integer;` X`:ensemble) : integer;`

Quelle est la complexité en temps de cette fonction par rapport à n?

Pour sélectionner le k-ième élément de X, on prend un nombre p bien choisi dans X, appelé *pivot* ($p \in X$), et on effectue une partition de X par rapport à p.

Question 3. Écrire la fonction `partitionP` qui prend en arguments un entier p et l'ensemble X ; et qui retourne l'ensemble de tous les éléments de X strictement plus petits que p.

(* *Caml* *)	{ *Pascal* }
`partitionP : int -> ensemble -> ensemble`	`function partitionP` `(`p`:integer;` X`:ensemble) : ensemble;`

Modifier cette fonction pour obtenir la fonction `partitionG` qui retourne l'ensemble de tous les éléments de X strictement plus grands que p. Quelles sont les complexités en temps de ces fonctions par rapport à n?

Question 4. Écrire la fonction récursive `elementDeRang` qui prend en arguments un nombre k entier naturel et l'ensemble X ; et qui retourne l'élément de rang k dans X, en choisissant comme pivot le premier élément de la liste représentant X.

(* *Caml* *)	{ *Pascal* }
`elementDeRang : int -> ensemble -> int`	`function elementDeRang` `(`k`:integer;` X`:ensemble) : integer;`

Le nombre d'opérations pris par la fonction précédente varie en fonction du choix du pivot.

Question 5. Donner un ordre de grandeur, par rapport à n, du nombre maximum $M(n)$ d'opérations pris par la fonction précédente.

En supposant équiprobables tous les rangs possibles pour le pivot choisi dans X, on peut démontrer que le temps moyen pris par `elementDeRang` est borné supérieurement par $T(n)$ où T est une fonction croissante, vérifiant $T(0) = 0$ et la formule de récurrence suivante :

$$T(n) \leq \ell\, n + \frac{1}{n} \sum_{i=1}^{n} \max\{T(i-1), T(n-i)\}$$

(ℓ est une constante indépendante de n et k).

Question 6. En déduire une constante c, qu'on déterminera en fonction de ℓ, telle que, pour tout n entier naturel, on a $T(n) \leq cn$.

On s'intéresse à présent à optimiser le coût dans le cas le pire $M(n)$. Considérons d'abord les sous-ensembles $X_i = \{e_{5i+1}, e_{5i+2}, e_{5i+3}, e_{5i+4}, e_{5i+5}\}$ de 5 éléments consécutifs dans X pour $0 \leq 5i < n$. Soit Y l'ensemble des médians des X_i.

Question 7. Écrire une fonction `medians` qui prend en arguments l'ensemble X ; et qui retourne l'ensemble Y.

(* *Caml* *)	{ *Pascal* }
`medians : ensemble -> ensemble`	`function medians (`X`:ensemble) : ensemble;`

Quelle est la complexité en temps pris par cette fonction par rapport à n?

Pour améliorer la vitesse de la fonction de sélection, on prend à présent comme pivot le médian de l'ensemble Y.

Question 8. Écrire la fonction `elementDeRangBis` qui prend en argument un entier naturel k et l'ensemble X ; et qui retourne l'élément de rang k dans X, en prenant comme pivot le médian de l'ensemble Y.

(* *Caml* *)	{ *Pascal* }
`elementDeRangBis : int -> ensemble -> int`	`function elementDeRangBis (`k`:integer; `X`:ensemble) : integer;`

Question 9. Montrer que son temps maximum $M'(n)$ vérifie la formule :

$$M'(n) \leq \ell' n + M'(\left\lfloor \frac{n}{5} \right\rfloor) + M'(7 \left\lfloor \frac{n}{10} \right\rfloor + 4)$$

où $\lfloor x \rfloor$ est la partie entière de x, et ℓ' est une constante que l'on ne cherchera pas à expliciter. En déduire que, pour n suffisamment grand, on a $M'(n) \leq c'n$ où c' est une constante à déterminer en fonction de ℓ'.

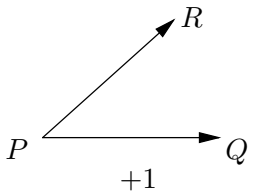
Question 10. Expliquer pourquoi l'ordre de grandeur du nombre maximal d'opérations pris par la fonction de sélection serait différent si on avait regroupé les éléments de X par groupes de 3, plutôt que par groupes de 5.

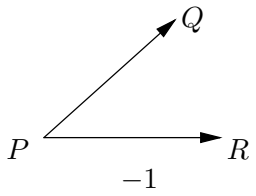
Second problème : Localisation dans un polygone convexe

Dans un repère cartésien, les points sont représentés par des enregistrements de type `point` défini par

(* *Caml* *)	{ *Pascal* }
`type point = {x:int; y:int};;`	`type point = record x:integer; y:integer end;`

Question 11. Écrire la fonction `orientation` qui prend comme arguments trois points P, Q, R ; et qui retourne $+1$, 0 ou -1 selon que l'angle α formé par les demi-droites $[PQ)$ et $[PR)$ vérifie soit $0 < \alpha < \pi$, soit $\alpha \in \{0, \pi\}$, soit $\pi < \alpha < 2\pi$.





(* *Caml* *)

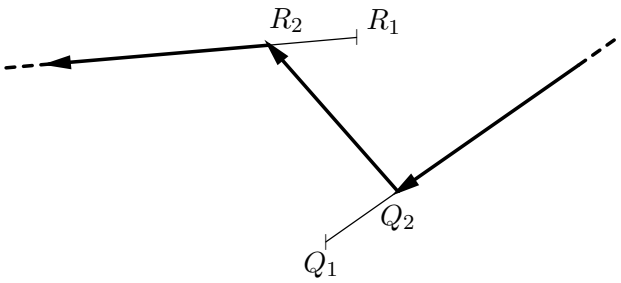
```
orientation :
  point -> point -> point -> int
```

{ *Pascal* }

```
function orientation
  (P:point; Q:point, R:point) : integer;
```

La région $\rho(Q_1Q_2R_1R_2)$, encore appelée région *à droite du zigzag* $Q_1Q_2R_1R_2$, est définie par trois éléments :

- la demi-droite partant de l'infini et finissant en Q_2 de vecteur directeur $\overrightarrow{Q_2Q_1}$,
- le segment de droite Q_2R_2,
- la demi-droite partant de R_2 et de vecteur directeur $\overrightarrow{R_1R_2}$.



La région $\rho(Q_1Q_2R_1R_2)$ se situe donc à droite de ces trois éléments pour un observateur les parcourant dans le sens indiqué.

Question 12. Écrire la fonction `aDroiteDe` qui prend comme arguments les points Q_1, Q_2, R_1, R_2, T ; et qui teste si le point T est à droite du zigzag $Q_1Q_2R_1R_2$.

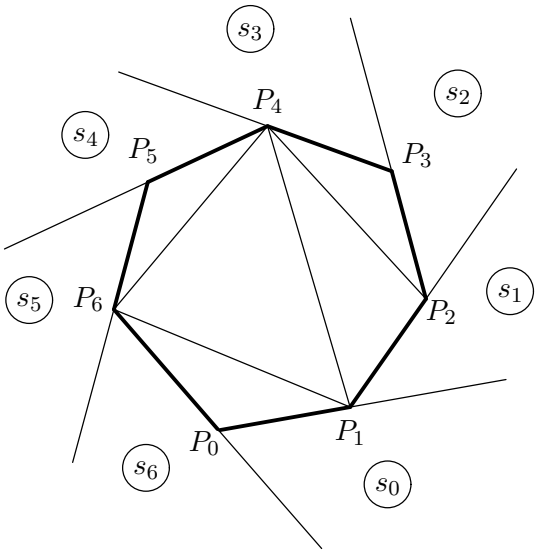
(* *Caml* *)

```
aDroiteDe :
  point -> point ->
  point -> point -> point -> bool
```

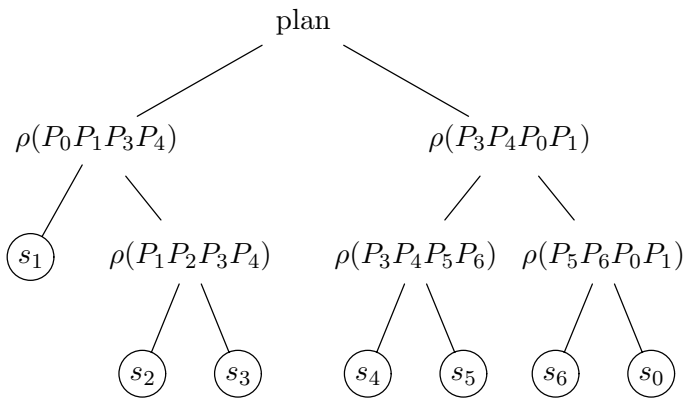
{ *Pascal* }

```
function aDroiteDe
  (Q1:point; Q2:point;
   R1:point; R2:point;
   T:point): boolean;
```

À présent, on se donne un polygone convexe $P_0P_1P_2 \ldots P_{n-1}$ de n côtés ($n \geq 3$), et on cherche à déterminer si un point P quelconque est à l'intérieur de ce polygone, en décomposant le plan par rapport au polygone de la façon suivante :



L'extérieur du polygone est composée des secteurs s_i délimités par les demi-droites $[P_{i-1}P_i)$ et $[P_iP_{i+1})$ pour tout i vérifiant $0 \leq i < n$ (les calculs d'indice se feront toujours dans l'arithmétique modulo n). L'intérieur du polygone est décomposé en triangles en considérant la construction arborescente suivante :



La racine est un nœud spécial représentant tout le plan. Tout nœud interne $\rho(P_{i-1}P_iP_{j-1}P_j)$ représente la région à droite du zigzag $P_{i-1}P_iP_{j-1}P_j$. Remarque : $\rho(P_{i-1}P_iP_{j-1}P_j)$ est la réunion des secteurs s_i, s_{i+1}, $\ldots$ s_{j-1} et de l'intérieur du polygone $P_iP_{i+1}\ldots P_{j-1}P_j$. Les feuilles sont les secteurs s_i déterminés par les trois points P_{i-1}, P_i, P_{i+1} comme expliqué précédemment.

Dans cette construction arborescente, chaque nœud (sauf la racine) est découpé en trois parties disjointes, deux de ces parties correspondent aux régions associées aux deux fils, et la troisième est le triangle $P_iP_kP_j$ intérieur au polygone (où P_k est le nouveau point apparaissant dans la définition des fils, en 4ème position dans le fils gauche et en 2ème dans le fils droit).

Le polygone est représenté par la liste $\langle P_0, P_1, \ldots P_{n-1}\rangle$ de ses sommets (dans l'ordre). L'arbre précédent de décomposition du plan est représenté par des éléments du type `arbre` défini par :

```
(* Caml *)
type listePoints ==
   point list;;
type arbre =
   Racine of
     arbre*arbre
 | Noeud of
     point*point
    *point*point
    *arbre*arbre
 | Feuille of
     point*point
    *point;;
```

```
{ Pascal }
type
  listePoints = ^cellule; cellule = record
    contenu:point; suivant:listePoints;
    end;
  tNoeud = (Racine, NoeudInterne, Feuille);
  arbre = ^noeud; noeud = record case indicateur of
    Racine: (gauche:arbre; droite:arbre);
    NoeudInterne: (q1:point; q2:point;
        r1:point; r2:point;
        gauche:arbre; droite:arbre);
    Feuille: (q1:point; q2:point; q3:point);
    end;
```

L'arbre est équilibré si, pour chacun de ses nœuds, la différence de hauteur entre ses deux fils a une valeur absolue inférieure ou égale à 1.

Question 13. Écrire la fonction `construire` qui prend en argument la liste $\langle P_0, P_1, \ldots P_{n-1} \rangle$; et qui retourne un arbre équilibré représentant sa décomposition arborescente. Donner la complexité en temps pris par cette fonction en fonction de n.

```
(* Caml *)
construire : listePoints -> arbre
```

```
{ Pascal }
function construire (polygone:listePoints):arbre;
```

Question 14. Avec quels arguments peut-on appeler la fonction `aDroiteDe` pour tester si le point T est dans le secteur s_i?

Question 15. Écrire la fonction `aLInterieurDe` qui prend en argument un arbre équilibré a et un point T ; et qui teste si T est dans le polygone représenté par a. Donner la complexité en temps de cette fonction par rapport à n.

```
(* Caml *)
aLInterieurDe :
  arbre -> point -> bool
```

```
{ Pascal }
function aLInterieurDe
  (a:arbre; T:point) : boolean;
```

Si T est un point à l'extérieur du polygone, les *tangentes* à partir de T par rapport au polygone convexe $P_0P_1 \ldots P_{n-1}$ sont les deux droites issues de T qui touchent le polygone sans couper son intérieur. La *tangente de gauche* se trouve à gauche pour un observateur en T faisant face au polygone ; l'autre tangente est la *tangente de droite*.

Question 16. Écrire la fonction `tangenteG` qui prend en arguments un arbre équilibré a et un point T ; et qui retourne un sommet P_i du polygone $P_0P_1 \ldots P_{n-1}$, représenté par l'arbre a, point de contact de la tangente de gauche issue d'un point T à l'extérieur du polygone. Donner la complexité en temps de cette fonction par rapport à n.

```
(* Caml *)
tangenteG : arbre -> point -> point
```

```
{ Pascal }
function tangenteG (a:arbre; T:point) : point;
```

Question 17. Donner une idée de modification des fonctions ou des structures de données précédentes pour obtenir également un point de contact de la tangente de droite.

* *
*