Exercice 1 (7 points)

Les quelques déclarations sont précisées en Ada pour fixer les idées, mais vous pouvez utiliser le langage de votre choix.

Dans une compétition de sports individuels, les candidats sont numérotés par ordre d'inscription, et doivent passer un certain nombre d'épreuves. Pour chacune des épreuves, ils reçoivent une note qui intervient dans le score final. Nous ne préciserons pas comment ce score est calculé. Il nous suffit de savoir que le score est un nombre flottant positif ou nul (il est nul si le candidat n'a encore participé à aucune épreuve, ou n'en a réussi aucune) et que, lors de chacune des épreuves, un nouveau score lui est attribué qui peut être meilleur ou moins bon que son score précédent.

On désire conserver les informations dans une structure de données qui permette:

- d'ajouter un candidat lors de son inscription,

- de modifier le score d'un candidat,

- de savoir à tout instant quels sont les trois meilleurs candidats, c'est-à-dire ceux dont le score est le plus élevé,

- de connaître pour tout candidat inscrit quelle est sa position dans le classement de tous les concurrents.

On prend une liste contiguë dont les éléments sont constitués du numéro du candidat et de son score. La déclaration Ada de cette liste, appelée `L_SCORE`, pourrait être la suivante:

```
type Candidat is
record
Numero : Positive;
Score : Float := 0.0;
end record;
package PLC is new Listes_Contigues (Candidat);
L_Score : PLC.Liste (1000);
-- la liste L_Score est une liste contiguë d'au plus
-- 1000 éléments de type Candidat
```

A(1 pt.)- Définissez l'ordre dans lequel vous choisissez de mettre les éléments dans la liste, en justifiant votre choix.

B(1 pt.)- Proposer la procédure qui attribue un numéro à un candidat lors de son inscription et l'ajoute à la liste. Évaluer la complexité. La spécification Ada pourrait être:

```
procedure Nouveau_Candidat (N : out Positive);
```

C(1 pt.)- Proposer la fonction qui retourne la place actuelle dans la compétition d'un candidat dont on connaît le numéro. Évaluer la complexité. La spécification Ada pourrait être:

```
function Position (N : Positive) return Positive;
```

D(1 pt.)- Proposer la procédure qui modifie le score d'un candidat. Évaluer la complexité. La spécification Ada pourrait être:

```
procedure Nouveau_Score (N : in Positive; S : in
Float);
```

E(1 pt.)- Proposer la fonction qui retourne le numéro du candidat qui a le score le plus élevé. Évaluer la complexité. La spécification Ada pourrait être:

```
function Meilleur return Positive;
```

F(2 pt.)- Proposer les fonctions qui retournent le numéro du candidat qui occupe, respectivement, la deuxième et la troisième place dans la compétition, c'est-à-dire celui qui a le score le plus élevé après le meilleur ou après les deux meilleurs. Évaluer la complexité dans chaque cas. La spécification Ada pourrait être:

```
function Deuxieme return Positive;
function Troisieme return Positive;
```

Exercice 2 (4 points)

Dans cet exercice, nous abordons le tri rapide.

A(1 pt.)- Rappeler en quelques lignes le principe et la complexité du tri rapide.

B(3 pts.)- Trier la liste ci-dessous cette méthode. Vous indiquerez pour chaque grande itération les opérations effectuées et la liste obtenue.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
35	5	70	15	40	65	10	25	20	30	55	50	45	60

La ligne du haut donne les indices des éléments, la ligne du dessous donne les valeurs contenues dans la liste.

Exercice 3 (9 points)

Cet exercice est constitué de trois parties indépendantes.

A- On s’intéresse aux arbres AVL.

A.1(0,5 pt.)- Rappeler la définition d’un arbre AVL.

A.2(0,5 pt.)- Rappeler en quoi consiste l'opération d'adjonction dans un arbre AVL.

A.3(2 pts.)- Construire l'arbre AVL par adjonction des valeurs dans l'ordre de la liste donnée ci-dessous. On s'attachera particulièrement à expliquer le raisonnement. Donner quelques arbres intermédiaires, sans oublier d'y porter les déséquilibres.

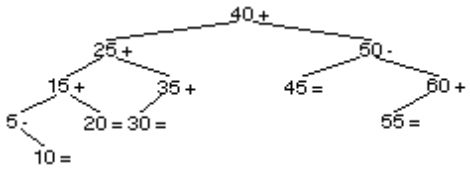
35	5	70	15	40	65	10	25	20	30	55	50	45	60

A.4(0,5 pt.)- Quelles vérifications peut-on faire sur l'arbre final pour contrôler une erreur éventuelle ?

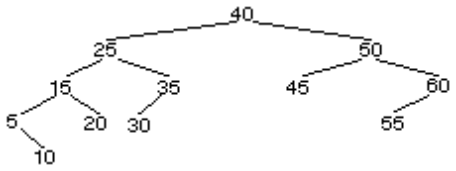
B- On s'intéresse à la suppression dans un arbre AVL.

B.1(0,5 pt.)- Rappeler en quoi consiste l'opération de suppression dans un arbre AVL.

B.2(2 pts;)- On se donne l'arbre AVL ci-dessous (il est différent de celui que vous avez obtenu dans la question A.3). Donner l'arbre obtenu par suppression de 45 dans cet arbre, puis celui obtenu par suppression de 25 dans le nouvel arbre, en justifiant, à chaque fois votre raisonnement.



C. On s'intéresse aux arbres binaires de recherche, et on se donne l'arbre suivant (c'est le même que celui de la question B.2, mais c'est sans importance).



C.1(0,5 pt.)- Donner la liste préfixe de l'arbre ci-dessus.

C.2(1 pt.)- Construire l'arbre binaire de recherche par adjonction des valeurs aux feuilles, dans l'ordre de la liste préfixe obtenue en C.1. On s'attachera particulièrement à expliquer le raisonnement ? Donner quelques arbres intermédiaires.

C.3(0,5 pt.)- Quelle vérification peut-on faire sur l'arbre final pour contrôler une erreur éventuelle ?

C.4(1 pt.)- L'arbre obtenu en C.2 devrait être le même que celui dont on est

parti. Pourquoi ?