A) Arbre : Incontestablement...

Soit l'arbre binaire B dont les parcours *préfixe* et *infixe* affichent les séquences suivantes:

Préfixe : `T W C I N E O D S T E U R`
Infixe : `I C N W O E D T T E S R U`

1) Représenter graphiquement l'arbre B correspondant à ces deux parcours.

2) Donner le parcours *suffixe* de l'arbre B.

B) Arbre : Poids...

Définition: Soit un arbre binaire B quelconque étiqueté par des valeurs entières, on appelle **poids de B** la somme de toutes les étiquettes de l'arbre B.

1) Donner le principe d'un algorithme déterminant le poids de B.

2) En utilisant les opérations définies par le *type algébrique abstrait* rappelé en annexe (dernière page), écrire la fonction récursive `calcule_poids(B)` correspondant à ce principe où B est de type a*rbrebinaire*

C) Liste : Progression arithmétique...

Écrire, après avoir donné son principe, la fonction Pascal, C ou Caml `arithmetique(L)` qui vérifie si les valeurs entières d'une liste `L`, contenant au moins deux éléments, suivent une progression arithmétique.

Si la liste possède au moins deux éléments et suit bien une progression arithmétique de raison non nulle, la fonction devra retourner la *raison* de la suite. Dans le cas contraire, la fonction devra retourner 0.

D) Liste : Insertion et tri...

1) Écrire, après avoir donné son principe, la procédure Pascal, C ou Caml `insertion(e,L)` qui insère un élément `e` à sa place dans une liste `L` triée en ordre croissant au sens large (la redondance de valeur est possible).

   Exemple :
   La liste L *avant insertion :* `1 4 8 12 15 …`
   La liste L *après insertion de la valeur 10 :* `1 4 8 10 12 15 …`

2) En utilisant ***obligatoirement*** la procédure `insertion(e,L)` du 1) , *que vous l'ayez écrite ou non*, écrire la procédure Pascal, C ou Caml `tri_insert(Lorig,Lres)` qui construit une nouvelle liste `Lres` triée en ordre croissant (au sens large) à partir d'une liste `Lorig` quelconque.

ANNEXE

Définition d'un type abstrait algébrique

ARBRE BINAIRE

Types

arbrebinaire

Utilise

noeud, élément

Opérations

arbrevide	:	→ arbrebinaire
<_, _, _>	:	nœud x arbrebinaire x arbrebinaire → arbrebinaire
racine	:	arbrebinaire → noeud
g	:	arbrebinaire → arbrebinaire
d	:	arbrebinaire → arbrebinaire
contenu	:	noeud → élément

Préconditions

racine(B) est défini ssi B≠arbre_vide
g(B) est défini ssi B≠arbre_vide
d(B) est défini ssi B≠arbre_vide

Axiomes

racine(<o, B1, B2>)= o
g(<o, B1, B2>)= B1
d(<o, B1, B2>)= B2

Avec

noeud	o
arbrebinaire	B, B1, B2