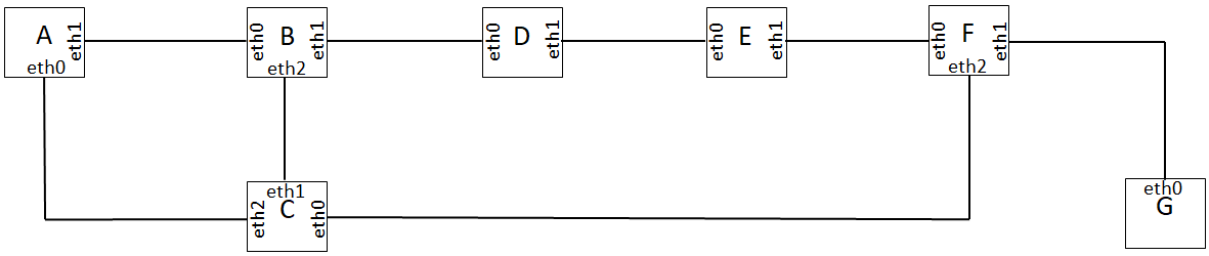
Exercice 2 (3 points)

Cet exercice traite des réseaux, de la programmation Python et de l'algorithmique.

1. Pour vendre ses produits, un grossiste en confiseries utilise un service web. Son réseau local correspond au réseau ci-dessous dans lequel les nœuds A, B, C, D, E, F et G sont des routeurs pour lesquels on souhaite déterminer les tables de routage.



Selon le protocole RIP, la distance est le nombre de routeurs traversés par un chemin pour aller d'un routeur à un autre et le chemin choisi entre deux routeurs est celui qui minimise la distance. On exécute ce protocole sur ce réseau.

Reproduire sur la copie et compléter la table suivante, qui indique pour chaque routeur la portion de la table de routage vers le routeur G.

Routeur	destination	passerelle	interface	distance
A	G	C	eth0	2
B	G			
C	G			
D	G			
E	G			
F	G			

Le grossiste propose à ses clients deux assortiments de 100 g chacun :

- le premier « `ToutFruit` » à base de bonbons de différents goûts ;
- le second « `ToutChoc` » à base de chocolats.

Voici un exemple de commande réalisée par un confiseur :

Commande n°343			
Confiserie	**Prix unitaire (€)**	**Quantité**	**Total (€)**
`ToutFruit`	7,00	1	7,00
`ToutChoc`	16,00	3	48,00
	Montant commande (€) :		55,00

2. Pour écouler ses stocks plus aisément, le grossiste propose à ses clients les réductions suivantes :
   - si le montant de la commande est supérieur ou égal à 100 € et strictement inférieur à 200 €, il applique une réduction de 10% ;
   - si le montant de la commande est supérieur ou égal à 200 €, il applique une réduction de 20%.

   Écrire en les complétant les lignes de code, à partir de la ligne 13, de la fonction `calcul_montant` afin de respecter la spécification donnée ci-dessous.

Numéro de lignes	Fonction `calcul_montant`
1	`def calcul_montant(prix_TF, quantite_TF, prix_TC, quantite_TC):`
2	`    """`
3	`    Renvoie le montant de la commande`
4	`    Entrées :`
5	`        prix_TF : prix de ToutFruit`
6	`        quantite_TF : quantite de ToutFruit`
7	`        prix_TC : prix de ToutChoc`
8	`        quantite_TC : quantite de ToutChoc`
9	`    Sortie :`
10	`        montant de la commande apres reduction eventuelle`
11	`    """`
12	`    montant = quantite_TF * prix_TF + quantite_TC * prix_TC`
13	`    if` ***`à compléter`*** `:`
14	`        montant = montant - montant*0.10`
15	`    elif` ***`à compléter`*** `:`
16	`        montant =` ***`à compléter`***
17	`    return` ***`à compléter`***

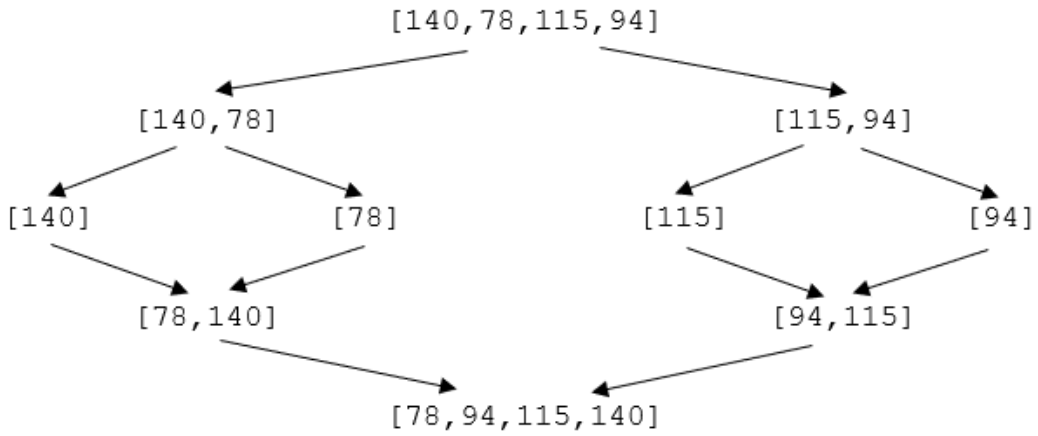
3. Le grossiste enregistre dans la liste `sommes` les montants des commandes, avant réduction, que des clients ont passées dans son magasin.
   Par exemple : `sommes = [140, 78, 115, 94, 46, 108, 55, 53]`

   Il va utiliser la méthode de tri-fusion pour trier la liste `sommes` dans l'ordre croissant.

   L'algorithme du tri-fusion est naturellement décrit de façon récursive (dans toute la suite, on supposera pour simplifier que le nombre d'éléments de la liste est une puissance de 2) :

   - si la liste n'a qu'un ou aucun élément, elle est déjà triée,
   - sinon,
     - on sépare la liste initiale en deux listes de même taille,
     - on applique récursivement l'algorithme sur ces deux listes,
     - on fusionne les deux listes triées obtenues en une seule liste triée.

On trouve ci-dessous un exemple de déroulement de cet algorithme pour la liste `[140,78,115,94]` de 4 éléments :



a. Appliquer sur votre copie, de la même façon que dans l'exemple précédent, le déroulement de l'algorithme pour la liste suivante :

`[46, 108, 55, 53]`

On rappelle que l'appel `L.append(x)` ajoute l'élément `x` à la fin de la liste `L` et que `len(L)` renvoie la taille de la liste `L`.

b. La fonction `fusion` ci-dessous renvoie une liste de valeurs triées à partir des deux listes `liste1` et `liste2` préalablement triées.

Numéro de lignes	Fonction `fusion`
1	`def fusion(liste1, liste2):`
2	`    liste_finale = []`
3	`    i1, i2 = 0, 0`
4	`    à compléter`
5	`        if liste1[i1] <= liste2[i2]:`
6	`            liste_finale.append(liste1[i1])`
7	`            i1 =i1 + 1`
8	`        else:`
9	`            liste_finale.append(liste2[i2])`
10	`            i2 = i2 + 1`
11	`    while i1 < len(liste1):`
12	`        liste_finale.append(liste1[i1])`
13	`        i1 = i1 + 1`
14	`    while i2 < len(liste2):`
15	`        liste_finale.append(liste2[i2])`
16	`        i2 = i2 + 1`
17	`    return liste_finale`

Parmi les quatre propositions suivantes, écrire sur la copie l'instruction à placer à la ligne 4 du code de la fonction `fusion` :

Proposition 1	`while i1 < len(liste1) and i2 < len(liste2):`
Proposition 2	`while i1 < len(liste1) or i2 < len(liste2):`
Proposition 3	`while i1 + i2 < len(liste1) + len(liste2):`
Proposition 4	`while liste1[i1 + 1] < liste2[i2]):`

c. Recopier et compléter sur la copie la fonction récursive `tri_fusion` suivante, qui prend en paramètre une liste non triée et la renvoie sous la forme d'une nouvelle liste triée.

On utilisera la fonction `fusion` de la question précédente.

Exemple : `tri_fusion([140,115,78,94])` doit renvoyer
`[78,94,115,140]`

Aide Python : si `L` est une liste, l'instruction `L[i:j]` renvoie la liste constituée des éléments de `L` indexés de `i` à `j-1`.
Par exemple, si `L=[5,4,8,7,3]`, `L[2:4]` vaut `[8,7]`.

Numéro de lignes	Fonction `tri_fusion`
1	`def tri_fusion(liste):`
2	`    if len(liste) <= 1:`
3	`        return liste`
4	`    à compléter`
...	

Exercice 3 (5 Points)

Cet exercice traite des arbres et de l'algorithmique.

Dans cet exercice, la taille d'un arbre est égale au nombre de ses nœuds et on convient que la hauteur d'un arbre ne contenant qu'un nœud vaut 1.

On utilisera la définition suivante : un arbre binaire de recherche est un arbre binaire, dans lequel

- on peut comparer les valeurs des nœuds : ce sont par exemple des nombres entiers, ou des lettres de l'alphabet ;
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre gauche de x, alors il faut que `y.valeur < x.valeur`
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre droit de `x`, alors il faut que `y.valeur` $\geq$ `x.valeur`