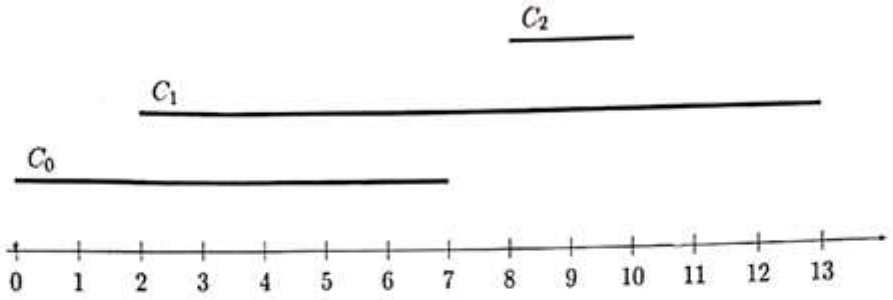
Dans ce problème, on s'intéresse à l'allocation des salles d'un lycée à partir des horaires des cours.

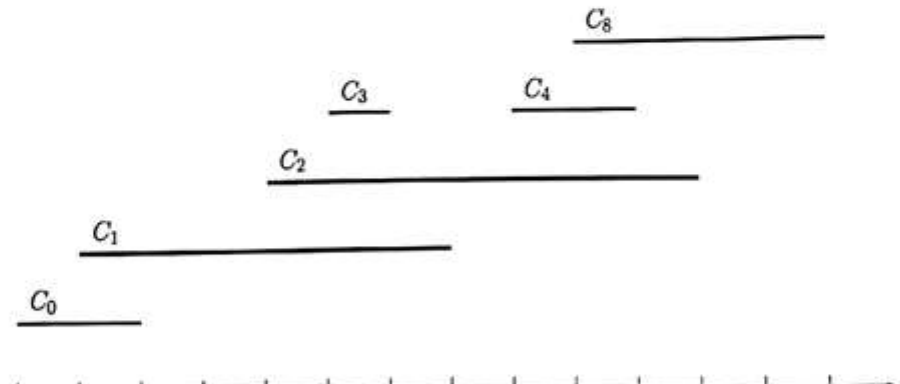
La donnée du problème est une liste de cours. Un cours est simplement représenté par un intervalle `[deb,fin[`, où `deb` est l'instant de début du cours et `fin` est l'instant de fin du cours. Les instants peuvent être décomptés en heures ou en minutes au fil de la journée selon les besoins ; dans ce sujet, les instants sont des nombres entiers et l'unité de temps n'est pas précisée.

À chaque cours, on doit allouer une salle. Deux cours peuvent se voir allouer la même salle, uniquement si leurs intervalles sont disjoints. En particuliers, deux cours représentés par les intervalles $[4, 6[$ et $[6, 8[$ peuvent se voir allouer la même salle. L'objectif est d'utiliser un minimum de salles.

On présente ci-dessous, deux instances de ce problème.



EXEMPLE 1



EXEMPLE 2

Dans l'exemple 1, le cours C_2 est représenté par l'intervalle $[8, 10[$.

1 Pour l'allocation des salles, on parcourt simplement les cours par instants de début croissants et on alloue systématiquement la salle disponible avec le numéro le plus petit.

a. Décrire, sans justification, les allocations ainsi obtenues pour chacun des exemples précédents.

b. Déterminer le nombre minimal de salles nécessaires pour l'allocation dans les deux exemples précédents.

On admet pour la suite que le nombre optimal de salles nécessaire pour une liste de cours est le nombre maximal d'intervalles s'intersectant mutuellement en un instant donné.

2 On s'intéresse dans cette question à une première modélisation du problème. Le cours représenté par la liste Python `[d,f]` se déroule sur l'intervalle mathématique $[d, f[$.

a. Déterminer les listes d'intervalles représentant les exemples 1 et 2.

b. Compléter la définition suivante :

```
def insere(l,elt):
```

où `l` est une liste d'entiers triée dans l'ordre croissant et `elt` est un entier, et qui renvoie une liste triée obtenue par insertion à sa place de `elt` dans `l`. On notera que la liste `l` peut être vide.

c. Créer une fonction similaire `insereBis` qui opère sur des listes de listes plutôt que sur des listes d'entiers, le critère de tri étant l'ordre des premiers éléments de chaque sous-liste. Plus précisément, la fonction `insereBis` a pour en-tête `def insereBis(LL,li):` et a pour effet d'insérer une liste `li` à la place adéquate dans la liste de listes `LL`.

3 Pour automatiser l'allocation des salles, on va utiliser une autre modélisation. L'intervalle `[deb, fin[` représentant le cours numéro i va être représenté par deux événements, modélisés par des listes de longueur 3 : `[deb,i,0]` et `[fin,i,1]`. Une liste d'intervalles pourra ainsi être représentée par une liste d'événements, c'est-à-dire une liste de listes de longueur 3 de la forme `[instant,num,0 ou 1]`.

a. Compléter la définition

```
def traduit(liste_intervalles):
```

qui prend en argument une liste d'intervalles représentés par des listes de longueur 2 et qui renvoie une liste d'événements (listes de longueur 3) correspondante. Notons que pour n intervalles, on obtient $2n$ événements.

b. Pour l'efficacité des algorithmes de résolution, on va travailler sur une liste d'événements triée par instants croissants. On appellera *agenda* toute liste d'événements triés par instants croissants. Quels sont les agendas correspondant aux exemples 1 et 2?

c. Compléter la définition

```
def agenda(liste_evt):
```

qui prend en argument une liste d'événements (listes de longueur 3) obtenue par un appel à la fonction traduit et qui renvoie l'agenda correspondant. On pourra utiliser la fonctions `insereBis`. Déterminer la complexité de la fonction agenda pour une liste de $2n$ événements.

4 On a demandé à des élèves d'écrire une fonction qui vérifie qu'une liste d'événements donnée contient bien autant de fin que de début, et dans le bon ordre, mais sans tester l'appariemment cours par cours, c'est-à-dire sans considérer les numéros d'intervalles.

a. Parmi les quatre solutions proposées, déterminer (sans justifier) la ou les réponses correctes et préciser leur complexité en fonction de la longueur de la liste en paramètre.

```
def valideA(agenda):
    c=0
    for e in agenda:
        if e[2] == 0: c += 1
        else: c -= 1
        if c < 0: return False
        else : return True

def valideB(agenda):
    n,c,i,b = len(agenda),0,0,True
    while b and (i < n):
        if agenda[i][2] == 0: c += 1
        else: c -= 1
        i += 1
        b = (c >= 0)
    return c == 0

def valideC(agenda):
    for i in range(len(agenda)):
        if agenda[i][2] == 0: b = False
            for j in range(i+1,len(agenda)):
                if agenda[j][2] == 1: b = True
            if not b : return(b)
    return(True)

def valideD(agenda):
    c=0
    for e in agenda:
        c += 1 - 2*e[2]
        if c < 0: return False
    return c == 0
```

b. Adapter une des fonctions précédentes pour écrire une fonction `intersection_max` qui calcule le nombre maximal d'intervalles qui s'intersectent mutuellement. Justifier la correction de l'approche.

c. En utilisant les fonctions précédentes, écrire une fonction `nbr_optimal` qui à partir d'une liste d'intervalles (modélisation initiale), calcule le nombre de salles nécessaire. Quelle est la complexité de la fonction `nbr_optimal` en fonction du nombre d'intervalles ?

5 On utilise à chaque instant une liste de booléens pour indiquer si une salle est disponible ou non.

a. Créer une fonction `plus_petit_vrai(liste)` qui étant donné une liste de booléens, calcule le plus petit entier *i* tel que la case d'indice *i* vaille `True`.

La fonction renverra -1 si un tel indice n'existe pas. Corriger sa fonction et en préciser la complexité en fonction de la longueur de la liste passée en paramètre.

b. En utilisant les fonctions précédentes, compléter la fonction suivante qui étant donnée une liste d'intervalles (listes de longueur 2), calcule une liste d'allocations des salles, toujours en allouant la salle disponible avec le plus petit numéro. Dans cette liste, la case d'indice le numéro du cours contient le numéro de la salle allouée.

```
def allocation(liste_intervalles):
    nb_cours = ...
    liste = ...
    nb_salles = ...
    salles_dispos = [True]*nb_salles
        alloc = [-1]*nb_cours
    for l in liste:
        if l[2] == 0 :
            alloc[l[1]] = ...
            salles_dispos[...] = False
        else :
            salles_dispos[...] = ...
    return(alloc)
```

Voici le programme complet que nous avons testé sur le site Python en ligne : https://repl.it/languages/Python3

```
#Cette fonction traduit une liste de plages horaires de cours (liste [heure début,heure fin] en une liste de longueur double d'évènements [heure début, numéro de cours, 0], [heure fin, numéro de cours, 1]
def traduit(l):
  n=len(l)
  a=list()
  for i in range(n):
    a.append([l[i][0],i,0])
    a.append([l[i][1],i,1])
  return a
#Cette fonction ordonne la liste précédente par ordre croissant d'instants (heure début ou heure fin) pour créer un agenda (= liste d'évènements ordonnés selon l'heure où ils se produisent)
def agenda (L) :
    permutation = True
    n=len(L)
```

```
    while permutation==True : #on répète la procédure tant qu'on a trouvé une permutation (c-à-d l'élément e de la liste est plus grand que l'élément qui le suit)

        permutation = False # on considère qu'il n'existe aucune permutation (c-à-d la liste est triée)

        for  i  in range( n-1) : # on parcourt la liste pour chercher s'il y une permutation
            if (L[i][0] > L[i+1][0]) :
                permutation = True # permutation existe on change la variable permutation en True
                L[i],L[i+1]= L[i+1],L[i] # on met L[i] et L[i+1] en ordre

    return L
#Cette fonction renvoie le plus petit indice dans une liste de Booléen pour lequel le terme correspondant a pour valeur True
def plus_petit_vrai(liste):
  n=len(liste)
  b=True
  i=-1
  while (i<n) and b:
    i += 1
    if liste[i]: b=False
  return(i)
#Cette fonction renvoie le nombre maximum d'intersections d'intervalle dans un agenda
def intersection_max(agenda):
  n,c,cmax,i,b=len(agenda),0,0,0,True
  while b and (i<n):
    if agenda[i][2]==0:
      c += 1
      if (c>=cmax): cmax=c
    else: c-=1
    i += 1
    b=(c>=0)
  return(cmax)
#Cette fonction renvoie le nombre maximum d'intersections d'intervalle dans une liste de plages horaires et donc le nombre optimal de salles nécessaires
def nbr_optimal(liste_intervalles):
  liste=agenda(traduit(liste_intervalles))
  n= intersection_max(liste)
  return(n)
#Cette fonction renvoie une liste contenant les numéros de salles (numérotées 0,1,2 etc) allouées aux différentes plages horaires (cours) numérotées 0,1,2 etc..
def allocation(liste_intervalles):
  nb_cours=len(liste_intervalles)
```

```
  liste=agenda(traduit(liste_intervalles))
  nb_salles=nbr_optimal(liste_intervalles)
  salles_dispos=[True]*nb_salles
  alloc=[-1]*nb_cours
  for l in liste:
    if l[2]==0:
      alloc[l[1]]=plus_petit_vrai(salles_dispos)
      salles_dispos[alloc[l[1]]] = False
    else :
      salles_dispos[alloc[l[1]]] = True
  return(alloc)
#exemple 1
cours=[[0,7],[2,13],[8,10]]
print(allocation(cours))

#exemple 2
cours=[[0,2],[1,7],[4,11],[5,6],[8,10],[9,13]]
print(allocation(cours))
```