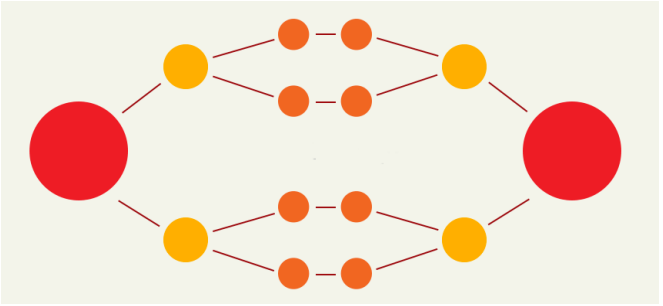
Diviser pour régner



- Diviser pour régner ("Divide and Conquer") est un principe de programmation qui consiste à décomposer un problème à résoudre en sous-problèmes, plus petits, les résoudre, éventuellement en les redécomposant encore avec le même principe, puis combiner les résultats des sous-problèmes pour en déduire le résultat du problème initial.
- Ce type de résolution invite à penser à la récursivité.
- Dans cette feuille seront détaillées deux applications de cette méthode :
  - La recherche dichotomique (déjà abordée en première sous une autre forme)
  - Un nouvel algorithme de tri : Le tri fusion

1. Recherche dichotomique

- Il s'agit de déterminer si une valeur `v` apparaît dans un tableau `T` supposé trié par ordre croissant. On renvoie l'indice auquel se trouve la valeur dans le tableau, `False` si elle n'y est pas.
- Le code ci-dessous génère une liste d'entiers aléatoires triée par ordre croissant.

In [1]:
```
#génération d'une liste triée de N nombres entiers aléatoires entre 0 et 65536
from random import *
N=2**16
T=[randint(0,2**16) for _ in range(N)]
T.sort()
```

Algorithme itératif(vu en première)

- Voici l'algorithme de recherche dichotomique d'un élément dans une liste d'entiers triée en ordre croissant qui a été vu en première :
- Si l'élément est présent, son index est renvoyé, sinon le booléen `False` est renvoyé.

(1) L *est une liste d'entiers triée,* n *est un entier*
(2) $gauche \leftarrow 0$ $(premier\ index\ de\ la\ liste)$
(3) $droite \leftarrow dernier\ index\ de\ la\ liste$
(4) $Tant\ que\ gauche \leq droite$
(5) $\quad m \leftarrow (g+d)//2$
(6) $\quad Si\ L[m] = n$
(7) $\quad\quad Renvoyer\ m$
(8) $\quad Sinon$
(9) $\quad\quad Si\ L[m] > n$
(10) $\quad\quad\quad droite \leftarrow m-1$
(11) $\quad\quad Sinon$
(12) $\quad\quad\quad gauche \leftarrow m+1$
(13) $Renvoyer\ False$

Exemple

Le programme ci-dessous affiche à chaque étape les indices `gauche` , `droite` l'indice médian `m` et sa valeur correspondante.

In [2]:
```
def dichotomie(n,T):
    print('On cherche ', n)
    gauche=0
    droite=len(T)-1
    while gauche<=droite:
        m=(gauche+droite)//2
        print('gauche :', gauche, ' droite :',droite, ' m :',m, 'valeur :',T[m] )
        if T[m]==n:
            print()
            print('Indice de ',n,' : ')
            return m
        else:
            if T[m]>n:
                droite=m-1
            else:
                gauche=m+1
    return False

n=choice(T)
dichotomie(n,T)
```

```
On cherche  41325
gauche : 0  droite : 65535  m : 32767 valeur : 32596
gauche : 32768  droite : 65535  m : 49151 valeur : 48969
gauche : 32768  droite : 49150  m : 40959 valeur : 40804
gauche : 40960  droite : 49150  m : 45055 valeur : 44882
gauche : 40960  droite : 45054  m : 43007 valeur : 42825
gauche : 40960  droite : 43006  m : 41983 valeur : 41806
gauche : 40960  droite : 41982  m : 41471 valeur : 41289
gauche : 41472  droite : 41982  m : 41727 valeur : 41556
gauche : 41472  droite : 41726  m : 41599 valeur : 41432
gauche : 41472  droite : 41598  m : 41535 valeur : 41366
gauche : 41472  droite : 41534  m : 41503 valeur : 41325

Indice de  41325  :
```

Out[2]:
```
41503
```

Remarques :

- On calcule à chaque étape l'indice médian de la partie du tableau dans laquelle on cherche l'élément.
- On compare la valeur à cet indice avec la valeur cherchée.
- Si on a pas trouvé la valeur , puisque le tableau est trié par ordre croissant, on cherche dans la partie gauche ou dans la partie droite en modifiant la valeur de `gauche` ou `droite`
- L'algorithme est très efficace, un petit nombre d'étapes suffit pour trouver la valeur.

Algorithme récursif

- Cette nouvelle version récursive contient quatre arguments : La valeur `v` recherchée, le tableau `T` trié dans lequel on cherche la valeur et l'intervalle(en terme d'index), délimité par les indexes `g` (gauche) et `d` (droite) dans lequel se situe la recherche.
- Les valeurs de `g` et `d` évoluent au fur et à mesure de la recherche

Voici le déroulement de cet algorithme :

- Si `g` est supérieur à `d` , c'est que la valeur `v` ne se trouve pas dans le tableau, on renvoie `False`
- On calcule l'indice médian `m` entre `g` et `d` comme dans la précédente version.
- Si la valeur du tableau à cet index est strictement inférieure à `v` , alors on relance récursivament une recherche dans la moitié droite délimité par les indices `g` et `m-1` .
- Si la valeur du tableau à cet index est strictement supérieure à `v` , alors on relance récursivament une recherche dans la moitié gauche délimité par les indices `m+1` et `d` .
- Sinon c'est que l'on a trouvé la valeur `v` , on renvoie son indice `m`

Exercice 1 :

Compléter la fonction `dichotomie_DQ(v,T,g,d)` comme décrite ci-dessus.

In [3]:
```
def dichotomie_DQ(v,T,g,d):
    pass
```

In [4]:
```
#On choisit une valeur dans le tableau trié
v=choice(T)
#On vérifie que la recherche renvoie bien l'index de v
assert(dichotomie_DQ(v,T,0,len(T)-1)==T.index(v))

#On vérifie que False est renvoyé lorsque la valeur v n'est pas dans le tableau
v=-1
assert(dichotomie_DQ(v,T,0,len(T)-1)==False)
```

Remarques :

- On est sûr que cet algorithme termine car l'écart entre `g` et `d` diminue strictement à chaque appel récursif.Lorsque $g-d<0$, c'est à dire lorsque $g>d$, on sort de la fonction.
- La stratégie "diviser pour régner" s'exprime ici par le fait que la taille de l'intervalle de recherche est divisée par deux à chaque étape. On se ramène à la résolution d'un problème plus petit.
- Pour un tableau de $2^{16}=65536$ éléments, il ne faut au maximum que 16 étapes pour trouver la valeur. Cet algorithme est de complexité logarithmique.
- Cet algorithme n'indique pas si une valeur se trouve plusieurs fois dans le tableau. Dès qu'il trouve un indice avec la valeur recherchée, il le renvoie.

Exercice 2

On considère le tabelau `T=[1,1,2,3,5,8,13,21]` .Ecrire la séquence des appels de la fonction précédente lors de l'éxécution de :

1. `dichotomie_DQ(T, 13 , 0, 7)`
2. `dichotomie_DQ(T, 12 , 0, 7)`

Réponse:

1.
2.

-
-

2.

-
-
-
-

2. Tri fusion

En première , nous avons abordé deux algorithmes de tri :

- Le tri par sélection
- Le tri par insertion

Dans les deux cas leur coût en temps est quadratique, c'est à dire de l'ordre de n^2 ou n est le nombre d'éléments à trier. Nous allons aborder un tri plus efficace, le tri fusion.

Présentation

- On partage l'ensemble des données à trier en deux parties de taille égale (à un élément près).
- On trie chacune de ces parties récursivement avec le tri fusion(c'est à dire éventuellement en partageant chacune des parties en deux).
- On fusionne les deux parties

Exemple :

Dans l'animation ci-contre, on cherche à trier la liste
$6, 5, 3, 1, 8, 7, 2, 4$.

- On partage la liste en deux sous listes : $6, 5, 3, 1$ et $8, 7, 2, 4$
- Pour trier ces deux sous listes, on les partage en deux $6, 5$ | $3, 1$ | $8, 7$ | $2, 4$
- Pour trier chacune des parties, on partage en deux : 6 | 5 | 3 | 1 | 8 | 7 | 2 | 4
- On peut plus partager mais toutes les parties sont triées, puisqu'elles n'ont qu'un seul élément.Il faut désormais les fusionner.
- On fusionne les parties deux à deux:
  - $5, 6$ | $3, 1$ | $7, 8$ | $2, 4$
  - $1, 3, 5, 6$ | $2, 4, 7, 8$
  - $1, 2, 3, 4, 5, 6, 7, 8$
- La liste est triée.

Exercice 3 :

En détaillant le contenu à chaque étape, trier par fusion la liste $38, 27, 43, 3, 9, 82, 10$

Réponse:

-
-
-
-
-
-

Remarque

- Il s'agit bien d'une application de la stratégie "diviser pour régner" car on ramène le problème du tri d'une liste aux sous-problèmes du tri de deux listes plus petites, jusqu'à parvenir à des listes d'au plus un élément, pour lesquelles il n'y a rien à faire.

Complexité

- En comparaison des tris vus en première (sélection et insertion), le tri fusion est beaucoup plus efficace.
- Ainsi pour trier 1 million d'éléments (10^6), il faut de l'ordre de $(10^6)^2 = 10^{12}$ opérations avec le tri sélection.
- Alors qu'avec le tri fusion on est plutôt de l'ordre de $10^6 \times log_2(10^6) \approx 10^6 \times 20 = 2 \times 10^7$ opérations.
- La division par deux du nombre d'éléments à chaque étape rend l'algorithme efficace.

Programmation

- Il est possible d'utiliser le tri fusion pour trier des tableaux (le type `list` de python) mais c'est assez délicat à mettre en oeuvre en pratique sans avoir à passer par des copies de tableaux, ce qui rend l'algorithme plus compliqué à écrire et plus coûteux.
- Nous allons donc implémenter le tri fusion sur des liste chaînées et pour cela réutiliser les classes ci-dessous.
- La fonction `lst_random(n, N)` permet de générer une liste de n entiers positifs inférieurs à N et de faire des tests.

In [3]:
```
class Cell:
    def __init__(self, valeur, suivant=None):
        self.valeur=valeur
        self.suivant=suivant

class Lc:
    def __init__(self, tete=None):
        self.tete=tete

    def est_vide(self):
        return self.tete is None

    def ajoute(self, nouvelle):
        if self.tete is None:
            self.tete=nouvelle

        else:
            cellule=self.tete
            while cellule.suivant is not None:
                cellule=cellule.suivant

            cellule.suivant=nouvelle

    def __str__(self):
        valeurs=str(self.tete.valeur)
        cellule=self.tete
        while cellule.suivant is not None:
            valeurs=valeurs+' | '+ str(cellule.suivant.valeur)
            cellule=cellule.suivant
        return valeurs
```

In [4]:
```
from random import *

def lst_random(n,N):
    '''génère une liste chaînée contenant
    n entiers positifs aléatoires inférieurs à N'''
    tete=Cell(randint(0,N), None)

    for i in range(n-1):
        nouvelle=Cell(randint(0,N), None)
        nouvelle.suivant=tete
        tete=nouvelle

    return Lc(tete)

print(lst_random(8,100))
```

```
59 | 7 | 30 | 30 | 26 | 75 | 44 | 68
```

Pour programmer le tri fusion, nous allons utiliser deux fonctions :

- La fonction `couper`
- La fonction `fusionner`

Couper

La fonction `couper(lst)` ci-dessous partage une liste en deux listes de taille égale à un élément près et les renvoie.

In [7]:
```
def couper(lst):
    '''Sépare une liste en deux listes
    ayant le même nombre d'éléments,
    à un près'''
    l1,l2=Lc(),Lc()
    while not lst.est_vide() :
        l1,l2=Lc(Cell(lst.tete.valeur,l2.tete)), l1
        lst=Lc(lst.tete.suivant)

    return l1,l2
```

Exercice 4:

On suppose que la liste `lst` à trier est `77 | 76 | 25 | 8 | 1 | 6 | 44 | 76 | 69` .

1. Ecrire un code qui permet de construire cette liste.

In [5]:
```
#1.
c1=Cell(77,None)


lst=Lc(c1)
print(lst)
```

```
77
```

1. Quelles valeurs contiendront les listes coupées `l1` et `l2` après l'appel de la fonction `couper(lst)` ?

Réponses :

- `l1` :
- `l2` :

1. Vérifier votre réponse ci-dessous.

In [1]:
```
#3.
```

Fusionner

La fonction `fusionner(l1,l2)` prend en paramètres deux listes triées et renvoie une liste triée contenant les éléments de ces deux listes.

Descriptif :

- On commence par le cas où l'une des deux listes est vide, il suffit alors de renvoyer l'autre.
- Sinon , cela veut dire que les deux listes sont non vides.
- On compare les premiers éléments de chaque liste entre eux.
- Si le plus petit provient de la première liste :
  - On le place en tête du résultat et le reste de la liste est construit en fusionnant récursivement le reste de la première liste avec la seconde.
- Sinon c'est l'inverse :
  - On place en tête le premier élément de la seconde liste et le reste de la liste est construit en fusionnant récursivement le reste de la seconde liste avec la première.

Exercice 5

La fonction `fusionner(l1,l2)` ci-dessous correspond au descriptif précédent. Ecrire un jeu de test qui permet de s'assurer que cette fonction fait bien ce que l'on attend d'elle.On s'assurera de proposer des exemples qui permettent de tester toutes les lignes du code

In [10]:
```
def fusionner(l1,l2):
    '''Fusionne deux listes triées
    en une seule'''
    if l1.est_vide():
        return l2
    if l2.est_vide():
        return l1
    if l1.tete.valeur <= l2.tete.valeur:
        return Lc(Cell(l1.tete.valeur,fusionner(Lc(l1.tete.suivant),l2).tete))
    else:
        return Lc(Cell(l2.tete.valeur,fusionner(l1,Lc(l2.tete.suivant)).tete))
```

In [6]:
```
#Réponses
```

Tri fusion

Exercice 6

A l'aide des fonctions `couper` et `fusionner` , compléter la fonction `tri_fusion(lst)` .

In [7]:
```
def tri_fusion(lst):
    pass
```

In [8]:
```
lst=lst_random(15,99)
print(lst)
print(tri_fusion(lst))
```

```
17 | 13 | 96 | 48 | 15 | 49 | 84 | 86 | 17 | 28 | 68 | 76 | 27 | 3 | 7
None
```

3. Exercices

Exercice 8 :

Vous disposez d'un tas de 512 pièces soit disant en or. Toutes sont fausses sauf une qui est un peu plus lourde que les autres. Déterminer un algorithme qui permet de trouver rapidement la vraie pièce en or à l'aide d'une balance à deux plateaux.

Réponse:

-
-
-
-

Exercice 9 :

Réécrire la fonction `fusionner(l1,l2)` à l'aide d'une boucle `while` . Aide : On pourra utiliser la méthode `ajoute(self,nouvelle)` disponible dans la classe `Lc()`

In [10]:
```
def fusionner(l1,l2):
    '''Fusionne deux listes triées
    en une seule'''
    pass
```

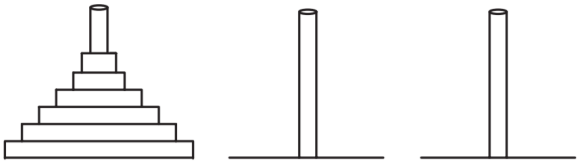
In [11]:
```
#Tests fusionner
l1=Lc(Cell(1,Cell(4,Cell(6,Cell(7,None)))))
l2=Lc(Cell(2,Cell(3,Cell(5,None))))
print(fusionner(l1,l2)) # 1 | 2 | 3 | 4 | 5 | 6 | 7

#l1 est vide
l1=Lc()
print(fusionner(l1,l2)) #2 | 3 | 5

#l2 est vide
l2=Lc()
l1=Lc(Cell(1,Cell(4,Cell(6,Cell(7,None)))))
print(fusionner(l1,l2)) #1 | 4 | 6 | 7
```

```
None
None
None
```

Exercice 10 : Les tours de Hanoi(voir l'exercice 13 du chapitre sur la récursivité)

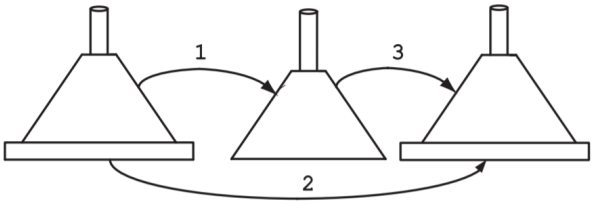


Ecrire une fonction `hanoi(n)` qui affiche la solution du problème des tours des Hanoi pour `n` disques , sous la forme de déplacements élémentaires désignant les trois tiges par les entiers 1,2 et 3 de la manière suivante :

- `1 vers 3`
- `1 vers 2`
- ...

Rappel : Pour transférer n disques

- On transfère $n-1$ disques sur le deuxième axe (1),
- Puis on transfère le plus gros disque sur le troisième axe (2) ,
- Et enfin les $n-1$ disques sur le troisième axe(3).



Aide :

- On pourra commencer par écrire la fonction récursive `deplace(a,b,c,k)` qui déplace k disques de la tige a vers la tige b en se servant de la tige c comme stockage intermédiaire.

In [12]:
```
def deplace(a,b,c,k):
    pass
def hanoi(n):
    pass
```

In [9]:
```
#Tests
hanoi(3)
```

```
1 vers 3
1 vers 2
3 vers 2
1 vers 3
2 vers 1
2 vers 3
1 vers 3
```