Exercice 1:

Le problème suivant est appelé FizzBuzz. Vous devez compléter la fonction *fizzBuzz()* qui prend en entrée un nombre n.

- Imprimer fizz si n est un multiple de 3
- Imprimer buzz s'il s'agit d'un multiple de 5
- Imprimer fizzbuzz s'il s'agit à la fois d'un multiple de 3 et 5
- Imprimer ledit chiffre autrement.

Il suffit de compléter la fonction `fizzBuzz()` .

In [5]:

```python
def fizzBuzz(n):
    # TODO imprimer la chaine de caractère appropriée avec la fonction print().
    #  Assigner ensuite la valeur à la variable resultat
    if n%3 == 0:
        if n%5 == 0:
            resultat = "fizzbuzz"
        else:
            resultat = "fizz"
    elif n%5 == 0:
        resultat = "buzz"
    else:
        resultat = n

    return resultat

n = int(input("indiquez le nombre: "))
print(fizzBuzz(n))
```

```
indiquez le nombre: 8
8
```

Exercice 2:

Dans cet exercice, vous devez résoudre une équation quadratique de la forme: $ax^2 + bx + c$

Le programme commence en demandant à l'utilisateur de saisir la valeur des variables `a` , `b` et `c` . Il suffit de compléter la fonction `resoudreEquation()` .

In [10]:

```python
import math


def resoudreEquation(a, b, c):
    # TODO: Calculer le discriminant et assigner la valeur dans la variable "delta"
    delta = b * b - 4 * a * c

    # TODO: Déterminer la condition (bool) qui correspond à aucune solution de l'équa

    if delta < 0:
        # ces ligne de code seront executé si il y'a aucune racine
        # TODO: afficher sur l'écran "Aucune racine"
        print("Aucune racine")
        # ne pas modifier
        return None

    # TODO: Déterminer la condition (bool) qui correspond à une unique solution de l'é
        # ces ligne de code seront executé si il y'a une seule racine
        x1 = 0
        # TODO: afficher sur l'écran "Une seule racine"
        print("Une seule racine")
        # TODO: assigner a la variable x1 la valeur de la racine
        x1 = -b/(2*a)
        # ne pas modifier
        return x1

    # TODO: Déterminer la condition (bool) qui correspond à deux solutions de l'équat
    if delta > 0:
        # TODO: afficher sur l'écran "Deux racines"
        print("Deux racine")
        # ne pas modifier

        # TODO: calculer la prmiere racine, assigner la a "x1"
        x1 = (-b - math.sqrt(delta)) / (2 * a)

        # TODO: calculer la deuxieme racine, assigner la a "x2"
        x2 = (-b + math.sqrt(delta)) / (2 * a)

        # ne pas modifier cette ligne
        return x1, x2



a = int(input("Entrez a, non nul: "))
b = int(input("Entrez b: "))
c = int(input("Entrez c: "))

print(resoudreEquation(a, b, c))
```

```
Entrez a, non nul: 3
Entrez b: 2
Entrez c: 4
Aucune racine
None
```

Exercice 3:

Dans cet exercice vous devez convertir un nombre de secondes en nombres d'années, semaines, jours, heures, minute et secondes. Par exemple, si l'utilisateur entre '633323104' secondes, votre programme devra renvoyé 20 années, 4 semaines, 2 jours, 3 heures, 5 minutes et 4 secondes. Vous pouvez créer d'autres variables pour vous aider.

PS: On considère qu'une année est composée exactement de 365 jours !

In [12]:

```python
def decomposer(secondes):

    # Ne pas modifier.
    annees = semaines = jours = heures = minutes = 0

    # TODO: Assigner à la variable "annees" le nombre d'années
    annees = secondes // (3600 * 24 * 365)
    secondes %= 3600 * 24 * 365

    # TODO: Assigner à la variable "semaines" le nombre de semaines restantes
    semaines = secondes // (3600 * 24 * 7)
    secondes %=  3600 * 24 * 7

    # TODO: Assigner à la variable "jours" le nombre de jours restants
    jours= secondes // (3600 * 24)
    secondes %= 3600 * 24

    # TODO: Assigner à la variable "heures" le nombre d'heures restantes
    heures = secondes // 3600
    secondes %= 3600

    # TODO: Assigner à la variable "minute" le nombre de minutes restantes
    minutes = secondes // 60

    # TODO: Assigner à la variable "secondes" le nombre de secondes restantes
    secondes %= 60

    # TODO: Afficher le nombres d'années, semaines, jours, heures, minutes et seconde:
    print (annees ,semaines ,jours ,heures ,minutes ,secondes)

    return (annees ,semaines ,jours ,heures ,minutes ,secondes)


secondes = int(input("Entrer les secondes: "))
print(decomposer(secondes))
```

```
Entrer les secondes: 633323104
20 4 2 3 5 4
(20, 4, 2, 3, 5, 4)
```

Exercice 4:

Completer la fonction *pointDeRencontre()* qui calcule la position du point de rencontre entre deux véhicules se déplaçant l'un vers l'autre à une vitesse respective de v1 et v2 et se trouvant à une distance d. Considérez que le véhicule 1 se trouve initialement au point 0 et que les vitesses sont constantes.

Si par exemple le véhicule 1 se déplace à une vitesse de 2 unités de distance par unité de temps, que le véhicule 2 se déplace à une vitesse de 1 unité de distance par une unité de temps et que les deux véhicules se trouvent à 12 unités de distance, ils se rencontreront au point situé à 8 unités de distance de la position initiale du véhicule 1.

Il suffit de compléter la fonction `pointDeRencontre()` . .

In [ ]:

```python
def pointDeRencontre(v1, v2, distance):
    # TODO faites les calculs intermediaires, vous pouvez initialiser des variables lo

    # TODO calculer la position de rencontre, assignez la valeur à la variable "posit:
    positionRencontre =

    return positionRencontre

v1 = int(input("Entrez v1: "))
v2 = int(input("Entrez v2: "))
distance = int(input("Entrez la distance: "))
print(pointDeRencontre(v1, v2, distance))
```

Exercice 5:

Dans cet exercice vous devez convertir une date (jour, mois, année) entré par l'utilisateur en jour de la semaine. Par exemple, si l'utilisateur entre 4 pour le jour, 2 pour le mois et 2021 pour l'année,la fonction va retourner jeudi. Pour cela, nous allons utiliser la formule de la congruence de Zeller.

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J\right) \bmod 7,$$

Afin de vous faciliter la tâche, la formule a été simplifié en plusieurs étapes.

$p = \left\lfloor \frac{14 - mois}{12} \right\rfloor$

$q = annee - p$

$r = q + \left\lfloor \frac{q}{4} \right\rfloor - \left\lfloor \frac{q}{100} \right\rfloor + \left\lfloor \frac{q}{400} \right\rfloor$

$s = mois + 12 * p - 2$

$t = (\left\lfloor \frac{jour + r + 31 * s}{12} \right\rfloor$\mod7)

Une fois ces étapes finies, le résultat t est convertir en jour. Finalement, il faut l'afficher.

In [13]:

```python
def convertirJour(j) :
    return {
        0 : "Samedi",
        1 : "Dimanche",
        2 : "Lundi",
        3 : "Mardi",
        4 : "Mercredi",
        5 : "Jeudi",
        6 : "Vendredi",
    }[j]

def trouverJourSemaine(annee, mois, jour):
    # TODO: Calculer p
    p = (14 - mois) // 12

    # TODO: Calculer q
    q = annee - p

    # TODO: Calculer r
    r = q + q // 4 - q // 1000 + q // 400

    # TODO: Calculer s
    s = mois + 12 * p - 2

    # TODO: Calculer t
    t = ((jour + r + 31 * s) // 12) % 7

    # Convertir le résultat de la formule de Zeller
    jour_semaine = convertirJour(t)

    # TODO: Afficher le jour de la semaine

    return jour_semaine

annee = int(input("Entrer l'annee: "))
mois = int(input("Entrer le mois: "))
jour = int(input("Entrer le jour: "))
trouverJourSemaine(annee, mois, jour)
```

```
Entrer l'annee: 2022
Entrer le mois: 1
Entrer le jour: 10
```

Out[13]:

```
'Lundi'
```

Exercice 6:

Dans cet exercice, vous devez calculer le nombre de chiffres qui compose un nombre indiqué par l'utilisateur. Cepandant, vous ne pouvez utiliser que les fonctions de la libraire Math.

In [33]:

```python
import math


def calculerNombreChiffres(nombre):

    # TODO: Déterminer le nombre de chiffres de "nombre" et mettre la valeur dans "no
    nombreDeChiffres = math.floor(math.log10(nombre + 1)) + 1

    # TODO: Afficher la valeur de "nombreDeChiffres"
    return nombreDeChiffres



nombre = int(input('veuillez indiquer un nombre strictement postif: '))
calculerNombreChiffres(nombre)
```

```
veuillez indiquer un nombre strictement postif: 1000
```

Out[33]:

```
4
```

Exercice 7:

Dans cet exercice, vous devez écrire un programme qui combine deux dictionnaires dans un 3e dictionnaire en gardant la valeur maximale des clés communes.
Exemple :

```python
dic_1 = {'a': 5, 'b': 2, 'c':9}
dic_2 = {'a': 1, 'b': 8, 'd':17}
dic_3 = {'a': 5, 'b': 8, 'd': 17, 'c': 9}
```

In [62]:

```python
def combineDic(dic_1, dic_2):
    # TODO Compléter la fonction afin de combiner dic_1 et dic_2
    #      en gardant la valeur max en cas de clef commune
    dic_3 = dict.fromkeys(list(dic_1.keys()) + list(dic_2.keys()))

    for i in dic_3.keys():
        dic_3[i]  = (dic_1[i] if i in dic_1.keys() else 0)
        dic_3[i] += (dic_2[i] if i in dic_2.keys() else 0)

    return dic_3

dic_1 = {'a': 5, 'b': 2, 'c': 9}
dic_2 = {'a': 1, 'b': 8, 'd': 17}

dic_3 = combineDic(dic_1,dic_2)
print(dic_3)
```

```
{'a': 6, 'b': 10, 'c': 9, 'd': 17}
```

Exercice 8:

Lors de la manipulation de liste il est commun de vouloir trier les valeurs qui la compose. Pour ce faire nous allons implémenter un algorithme de tri appelé tri à bulle. Il s'agit d'un algorithme peu efficace, mais facile à implémenter.
Le pseudo-code de l'algorithme est le suivant :

```
triBulles(Tableau T)
   pour i allant de (taille de T)-1 à 1
       pour j allant de 0 à i-1
           si T[j+1] < T[j]
               on inverse les valeurs
```

Une animation qui représente le fonctionnement de l'algorithme :

Nous ferons un tri par ordre croissant ce qui donnera :

```python
val = [5,8,1,9,6,2,4,3,7,5]
sorted_val = [1,2,3,4,5,5,6,7,8,9]
```

ATTENTION Un *pseudo-code* est une façon de décrire un algorithme qui ne prend pas en compte les spécificités du language. Mais dans le cas présent les tableaux sont quand même considérés comme commençant à 0.

In [68]:

```python
def triBulles(tab):
    # TODO écrire l'algorithme du tri à bulle comme décrit dans l'énoncé
    n = len(tab)
    for i in range(n-1, 0,-1):
        for j in range(i):
            if tab[j+1] < tab[j]:
                tab[j], tab[j+1] = tab[j+1], tab[j]
    return tab

val = [5,8,1,9,6,2,4,3,7,5]
sorted_val = triBulles(val)
print(val)
```

```
[1, 2, 3, 4, 5, 5, 6, 7, 8, 9]
```

Exercice 9:

Dans cet exercice, vous devez écrire un programme qui permet de calculer une valeur approchée de π par la méthode de Monte-Carlo basée sur les probabilités.

L'idée est la suivante : si on insère un cercle de rayon 1 (soit un cerle d'aire égale à π) dans un carré de côté 2 (et donc d'aire égale à 4), la probabilité qu'un point placé aléatoirement dans le carré soit également dans le cercle est donc de $\frac{\pi}{4}$ (le rapport des aires).

Afin d'estimer la valeur de π, voici la procédure que vous devez implémenter :

- générer itérativement deux nombres réels aléatoires x et y, tous deux compris entre -1 et 1.
- vérifier l'appartenance du point au cercle : si $x^2 + y^2 < 1$, le point est à l'intérieur du cercle.
- estimer la valeur de π en calculant le ratio entre les points à l'intérieur du cercle et les points à l'intérieur du carré, itérativement, jusqu'à ce que l'écart relatif entre votre estimation et la valeur précise de π soit de l'ordre de 0.001. Le résultat de votre estimation sera donc de 3,141xxxx.

Votre fonction doit retourner un tuple qui contiendra votre estimation de π et le nombre d'itérations que le programme a effectué pour arriver à ce résultat. Vous devez utiliser la fonctions random() pour la génération de nombres aléatoires.

In [74]:

```python
from random import random
def compute_pi(p):
    approximation_pi = 0
    nb_iterations = 0
    precision = 10 ** -p
    nb_dans_cercle = 0
    # TODO calculer la valeur de pi d'après la formule donnée dans l'énoncé
    while math.fabs(approximation_pi - math.pi) > precision:
        nb_iterations += 1
        x = random() * 2 - 1
        y = random() * 2 - 1
        if x**2 + y**2 < 1:
            nb_dans_cercle += 1
        approximation_pi = nb_dans_cercle / nb_iterations * 4.0
    return approximation_pi, nb_iterations

pi = 3.141592653589793
p = 5
computed_pi , nb_iter = compute_pi(p)
print("La valeur réel de pi est : {}".format(pi))
print("La valeur approché de pi à 10e-{} est : {}".format(p, computed_pi))
print("Résultat obtenu après {} itérations".format(nb_iter))
```

```
La valeur réel de pi est : 3.141592653589793
La valeur approché de pi à 10e-5 est : 3.141598489616111
Résultat obtenu après 3178 itérations
```

Exercice 10:

Sois la suite de Fibonacci, définie comme suit: $\begin{cases} S_0 = 0 \\ S_1 = 1 \\ S_n = S_{n-2} + S_{n-1} \end{cases}$ Dans cet exercice, vous devez écrire un programme qui demande à l'usager de saisir un nombre entier **P** et qui affiche le plus petit entier **n** tel que $S_n > P$

In [79]:

```python
def fibonacci(P):
    # TO-DO: Calculer le neme element de la suite de Fibonacci
    S, S1, S2 = 0, 0, 1
    n = 2;
    while S <= P:
        n += 1;
        S = S1 + S2;
        S1, S2 = S2, S

    return n

P = int(input("Entrez P: "))
valeur = fibonacci(P)
print(valeur)
```

```
Entrez P: 10
8
```

Exercice 11:

Dans cet exercice, vous devez écrire un programme qui calcule le **PGCD** de deux nombres saisi par l'utilisateur à l'aide de la méthode des différences successives.

Principe de cette méthode: si un nombre est un diviseur de **2** nombres **a** et **b (a > b)**, alors il est aussi un diviseur de leur différence **(a – b)**;

Exemple: calculons le **PGCD** de **36** et **60** à l'aide de la méthode des différences. Commençons par soustraire **36** de **60**:

60 - 36 = 24

donc le **PGCD** de **60** et **36** est un diviseur de **24**.

Cherchons alors le **PGCD** de **36** et **24**. Comme la différence obtenue est non nulle, on continue en utilisant le résultat obtenu et le plus petit des **2** termes de la soustraction:

36 - 24 = 12

La différence est non nulle donc on continue:

24 - 12 = 12

La différence est non nulle donc on continue:

12 - 12 = 0

La différence est nulle, on prend le dernier résultat non nul qui est: **12**. On conclut donc que **PGCD(36;60)= 12**.

In [80]:

```python
def PGCD(nb_1, nb_2):
    txt = "le plus grand diviseur entre " + str(nb_1) + " et " +  str(nb_2) + " est: 
    while nb_2:
        nb_1, nb_2 = nb_2, nb_1 % nb_2
    print(txt.format(nb_1))
    return nb_1

nb_1 = int(input("veuillez saisir le premier nombre: "))
nb_2 = int(input("veuillez saisir le second nombre: "))
nombre = PGCD(nb_1, nb_2)
```

```
veuillez saisir le premier nombre: 36
veuillez saisir le second nombre: 60
le plus grand diviseur entre 36 et 60 est: 12
```

Exercice 12:

Dans cet exercice, vous devez écrire un programme qui demande à l'utilisateur de saisir deux matrices **A** et **B** et calcul le produit des deux matrices. Si le nombre de colonnes de la matrice A est diffèrent du nombre de lignes de la matrice B, vous devez retourne None.

Exemple 1:

```python
A = [[2, 3, 6], [5, -1, 12]]
B = [[1, 0, 4, 7], [2, -3, 6, 2], [5, 8, 9, 6]]
C = multiplication_matrice(A,B)
  = [[38, 39, 80, 56], [63, 99, 122, 105]]
```

Exemple 2:

```python
A = [[2, 3, 6], [5, -1, 12]]
B = [[1, 0, 4, 7], [2, -3, 6, 2], [5, 8, 9, 6]]
C = [[38, 39, 80, 56], [63, 99, 122, 105]]
D = multiplication_matrice(C,A)
  = None
E = multiplication_matrice(B,C)
  = None
```

In [ ]:

```python
def multiplication_matrice(A, B):


    # TO-DO: determiner les nombre de Colonne de la matrice A et le nombre de ligne de


    # TO-DO: Vérifier si la multiplication est possible, si elle n'est pas possible re


    # TO-DO: Effectuer la multiplication matricielle


    return ...


A = [[2, 3, 6], [5, -1, 12]]
B = [[1, 0, 4, 7], [2, -3, 6, 2], [5, 8, 9, 6]]
C = multiplication_matrice(A, B)
print(C)
```

Exercice 13:

In [2]:

```python
def premier(nombre):
    estpremier = False
    if nombre > 1:
        for i in range(nombre -1, 0, -1):
            if nombre % i == 0:
                break
        if i == 1:
            estpremier = True
    return estpremier

nombre = int(input("veuillez saisir un nombre: "))
estpremier = premier(nombre)
```

```
veuillez saisir un nombre: 1
```

Exercice 14:

Dans cet exercice, vous devez écrire un programme qui demande à l'usager de saisir une matrice **M**, et qui retourne les éléments **M(i,j)** premier.

In [93]:

```python
def premierMatrice(M):
    # TO-DO: determiner les M(i,j) premier
    ligne, colonne = len(M), len(M[0])
    list_nb_premier = []
    for i in range(ligne):
        for j in range(colonne):
            nombre = M[i][j]
            if nombre > 1:
                for k in range(nombre -1, 0, -1):
                    if nombre % k == 0:
                        break
                if k == 1:
                    list_nb_premier.append(nombre)

    return list_nb_premier

M = [[2, 3, 6], [5, 12, -1]]
M_premier = premierMatrice(M)
print(M_premier)
```

```
[2, 3, 5]
```

Exercice 15:

In [ ]:

```python
def Somme_Div_Premier(nombre):
    somme = 0;
    for i in range(2, nombre + 1):
        if nombre % i == 0:
            for j in range(i -1, 0, -1):
                if i % j == 0:
                    break
            if j == 1:
                somme += i
    print("La somme des diviseur premier de {} est: {}".format(nombre, somme))
    return somme

nombre = int(input("veuillez saisir un nombre: "))
somme = Somme_Div_Premier(nombre)
```

Exercice 16:

In [ ]:

```python
def Factorielle(nombre):
    fac = 1
    for i in range(nombre, 1, -1):
        fac *= i
    print("le factorielle de {} est: {}".format(nombre,fact))
    return fac
nombre = int(input("veuillez saisir un nombre: "))
fact = Factorielle(nombre)
```

Dans cet exercice, le mot "expression" désigne une chaîne de caractères ne contenant que des parenthèses ouvrantes et fermantes. Par exemple "(()())", "(()()" et "(()))(". Une expression est bien parenthésée si le nombre de parenthèses ouvrantes est égal au nombre de parenthèses fermantes, et si quelque soit la position dans l'expression, le nombre de parenthèses ouvrantes qui précèdent cette position est toujours supérieur ou égal au nombre de parenthèses fermantes qui précèdent.

• "(()())" est une expression bien parenthésée.

• "(()()" est mal parenthésée car il y a 3 parenthèses ouvrantes et 2 parenthèses fermantes seulement.

• "(()))()" est mal parenthésée car le cinquième caractère est la troisième parenthèse fermante, alors qu'il n'y a que deux parenthèses ouvrantes qui précèdent.

Dans cet exercice l'utilisateur va entrer une expression et, si l'expression est mal parenthésée, la fonction retourne "Incorrect", sinon elle retourne la même expresion, mais en insérant des '.' à chaque fois qu'une parenthèse ouvrante est suivie d'une parenthèse fermante. Voici un exemple pour illustrer ce qui est attendu:

In [ ]:

```python
def ValideExpression(expression):
    #TODO: retourner la valeur adéquate





expression = input("veuillez entrer l'expression : ")
print(ValideExpression(expression))
```

Dans cet exercice, vous devez écrire un programme qui détermine le nombre de rebonds effectué par la balle avant que la hauteur du rebond soit inferieure à 0.01 mètre. Les données à lire du clavier sont : la hauteur initiale, le coefficient de rebond.

Les variables sont les suivantes : h_{i-1} est la hauteur avant le rebond numéro i, et h_i est la hauteur après le rebond numéro i.

v_{i-1} est la vitesse de la balle avant le rebond numéro i, et v_i est la vitesse après le rebond i.

Les relations entre les variables sont les suivantes :

$v_{i-1} = \sqrt{2 * g * h_{i-1}}$, avec $g = 9.81$"> est la constante de gravité.

$v_i = v_{i-1} * c$, avec c le coefficient de rebond.

$h_i = (v_i)^2 / 2 * g$

In [ ]:

```python
import math

constanteGravitationnelle = 9.81
def NbRebond(hauteurInitiale, coefficientDeRebond):
    #TODO: faites vos calculs et mettez le resultat dans la variable 'nombreDeRebonds



    return nombreDeRebonds

hauteurInitiale = float(input("Quelle est la hauteur initiale: "))
coefficientDeRebond = float(input("Quel est le coefficient de rebond(entre 0 et 1 excl
print(NbRebond(hauteurInitiale, coefficientDeRebond))
```

Dans cet exercice, vous allez participer à la conception d'un système de filtrage des spams **RENEGE** (use**R** awar**E** bayesia**N** filt**E**rin**G** syst**E**m). Ce système permet d'étiqueter les courriels en déduisant si ce sont des **spams** (non désirés) ou des **hams** (messages ok). Il s'agit essentiellement d'un classificateur bayésien associé à des règles heuristiques.

Vous ne devrez pas implémenter l'entièreté du système, mais simplement créer un vocabulaire qui sera utilisé pour entrainer les algorithmes d'intelligence artificielle de RENEGE. Dans le fichier `email.json` , vous avez une liste de 1000 courriels, avec les attributs suivants :

- **From**: le destinataire du courriel.
- **Date**: la date du courriel.
- **Body**: la liste des mots qui sont contenus dans le courriel.
- **Spam**: si la valeur est 'true', le courriel est un Spam, sinon c'est un Ham.

Le fichier `email.json` à été sauvegardé dans la variable 'emails' sous forme d'une liste de dictionnaires avec la fonction

```python
with open('email.json') as json_data:
            emails = json.load(json_data)
```

Voici à quoi ressemble la structure de la variable **emails:**

figure 6.1

Vous devez calculer, pour les mots du **Body** de chaque courriel, la probabilité que le mot soit dans un **spam** ou dans un **ham**. Par exemple, la probabilité qu'un mot soit dans un **spam** se calcule ainsi:

À l'inverse, la probabilité qu'un mot soit dans un **ham** se calcule ainsi:

Finalement, ces probabilités seront insérés dans un dictionnaire où l'on distiguera le résultat pour les spams de ceux des hams ainsi :
figure 6.2

Vous n'avez pas besoin d'arrondir le résultat. Votre résultat final sera injécté dans fichier `results.json` , avec la fonction :

```python
with open('results.json', 'w') as fp:
            json.dump(result, fp, indent=4)
```

où la variable `result` contient le dictionnaire en question.

Voici à quoi ressemble la fonction à compléter:

```python
def createVocabulary():
            #les mails contient la structure de donnée présente dans mail.json
            with open("mails.json") as json_data:
                email = json.load(json_data)

           #TODO: Affécté à la variable 'result' le résultat final

            result = {}
            with open('results.json', 'w') as fp:
                json.dump(result, fp, indent=4)
```

In [ ]:

```python
import json

def createVocabulary():
    #les mails contient la structure de donnée présente dans mail.json
    with open("mails.json") as json_data:
        emails = json.load(json_data)

    #TODO: Affécté à la variable 'result' le résultat final
    result = {}
    with open('results.json', 'w') as fp:
        json.dump(result, fp, indent=4)

createVocabulary()
```

Les problèmes de tri font partie des problèmes les plus célèbres en informatique. Il existe un nombre pratiquement infini d'algorithmes permettant de trier des éléments en ordre croissant(ou autre), toutefois tous ne sont pas égaux. En effet, bien que chaque algorithme (fonctionnel) arrive à un résultat correct, certains sont beaucoup plus performants que d'autres. La performance n'est pas critique lorsque l'on trie des centaines, voire des milliers d'éléments... Mais quand on tombe dans les millions et plus, ça devient important !

C'est là que l'algorithme de tri par fusion (merge sort) se démarque du tri par sélection vu au dernier TP. Bien que ce dernier fonctionne... Il est très lent à comparer du tri par fusion. Cet algorithme n'est toutefois pas le plus rapide existant, mais il est une bonne introduction au monde de l'algorithmie. De plus, celui-ci utilise le principe de récursivité, qui est une notion importante à comprendre, ainsi que le principe de *diviser pour régner* qui est un *patron* important en algorithmie.

merge-sort

> Schéma du tri par fusion

La fonction *.sort()* de Python, par exemple, utilise une variation de l'algorithme *merge sort*, soit le Timsort.

Visualisation de l'algorithme

Cette vidéo permet de bien visualiser le fonctionnement de l'algorithme de tri par fusion.

Fonctionnement

En utilisant le principe de diviser pour régner, cet algorithme parvient à rapidement trier une séquence de nombre en ordre croissant. Les étapes, sous forme de commentaires, sont déjà placées pour vous dans le code.

Voici une *tentative* d'explication de l'algorithme. Si ce n'est pas clair, une simple recherche sur Google avec les termes: *Tri fusion étape par étape* ou bien *How to merge sort* devrait répondre à toutes vos questions.

1. Diviser la séquence de nombres en appelant récursivement la fonction de tri fusion. Cette fonction sépare la lsite reçue en 2, et fusionne les 2 parties une fois *triées*. Cette fonction atteint sa condition d'arrêt seulement quand la liste reçue est de taille 1
2. Le tri se passe dans la fonction *fusionner*. En effet, celle-ci reçoit 2 listes (de taille 1 ou plus) et place dans une liste résultat les éléments des 2 listes reçues, en ordre croissant.
3. Le tri se termine lorsque l'on revient au premier appel récursif, soit lorsque la liste résultat a la même longueur que la liste initiale.

Notes

- Le fichier *listeDeNombres.txt* vous est fourni, et contient des lignes contenant des séquences de nombre à trier. Ces nombres sont séparés par des *espaces*, et les lignes par des *sauts de ligne*. La dernière ligne du fichier est une ligne vide.
- Les résultats doivent être enregistrés dans le fichier *resultats.txt*, sous la même forme qu'ils sont présentés dans le fichier *listeDeNombres.txt* (mais triés !), soit une séquence de nombres (par ligne) séparés par des espaces, avec un saut de ligne à la fin.
- Le *main* et la fonction de test *testerResultat(sequences_a_trier, sequences_triees)* n'ont pas besoin d'être modifié, seulement les fonctions *lire_fichier()*, *sauvegarder_sequences_triees(chemin, nom, liste_triees)*, *fusionner(gauche, droite)*, et *tri_fusion(sequence_de_nombre)* doivent être modifiés.
- Des instructions textuelles vous sont données pour vous indiquer quels sont les changements à effectuer.

In [ ]: