Informatique et Programmation

Cours 4

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-py

Plan

- courbe de Hilbert
- hanoi — recap
- fonctions récursives non numériques
- récursivité et raisonnement inductif
- merge sort
- quicksort
- types et adresses en Python

dès maintenant: télécharger Python 3 en `http://www.python.org`

Fractales

- la courbe de Hilbert (avec le paquetage `turtle.py`)

```
def hilbertG(n, d):
    if n > 0:
      left(90); hilbertD(n-1, d)
      forward(d)
      right(90); hilbertG(n-1, d)
      forward(d)
      hilbertG(n-1, d); right(90)
      forward(d)
      hilbertD(n-1, d); left(90)


def hilbertD(n, d):
    if n > 0:
      right(90); hilbertG(n-1, d)
      forward(d)
      left(90); hilbertD(n-1, d)
      forward(d)
      hilbertD(n-1, d); left(90)
      forward(d)
      hilbertG(n-1, d); right(90)
```

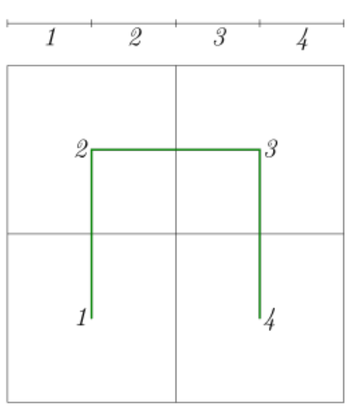


Fig. 1.

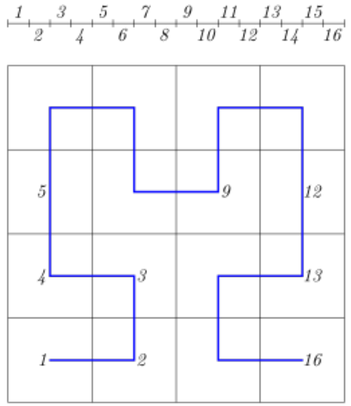


Fig. 2.

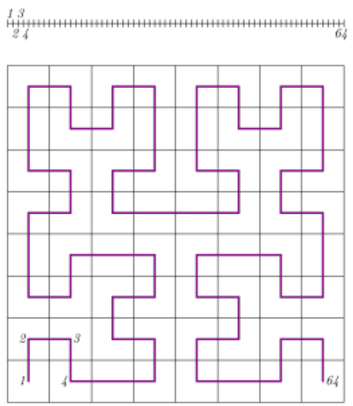
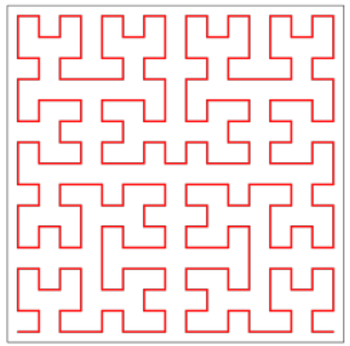
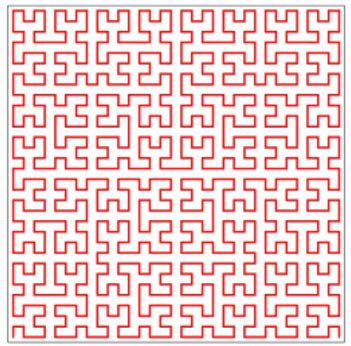
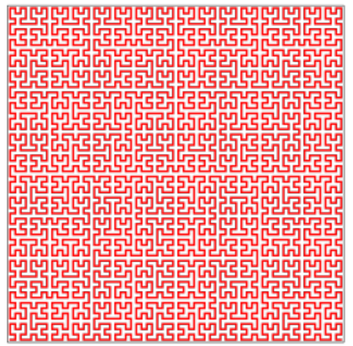


Fig. 3.







- les deux fonctions (mutuellement récursives) dessinent la courbe de Hilbert à gauche et à droite

Fractales

- la courbe de Hilbert (avec le paquetage `turtle.py`)

```
def hilbert(n, angle, d):
    if n > 0:
      left(angle); hilbert(n-1, -angle, d)
      forward(d)
      right(angle); hilbert(n-1, angle, d)
      forward(d)
      hilbert(n-1, angle, d); right(angle)
      forward(d)
      hilbert(n-1, -angle, d); left(angle)
```

```
def main (n, d) :
    home(); clear();
    tracer (0, 0); setheading (0)
    hilbert (n, 90, d)
    update ()
```

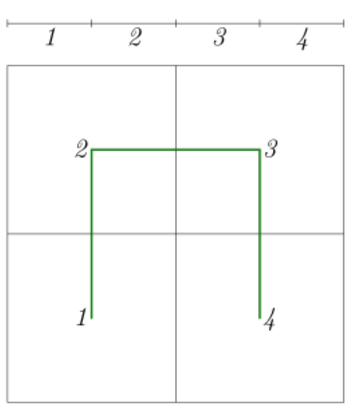


Fig. 1.

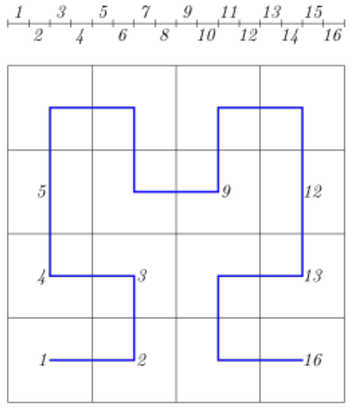


Fig. 2.

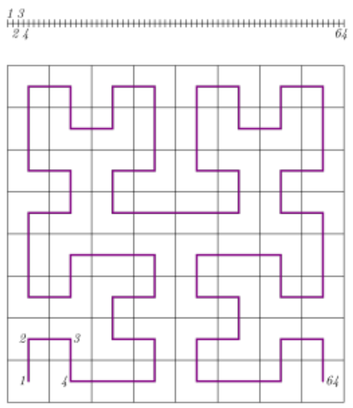
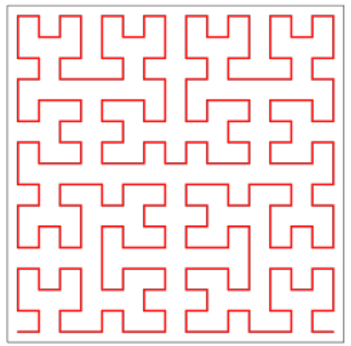
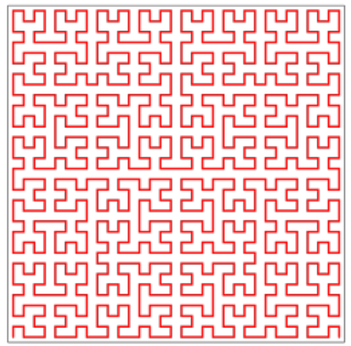
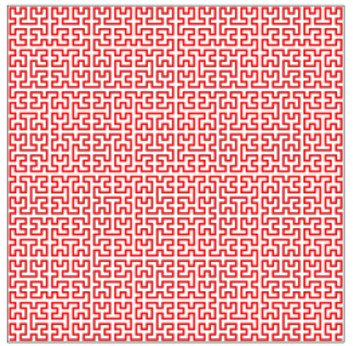


Fig. 3.





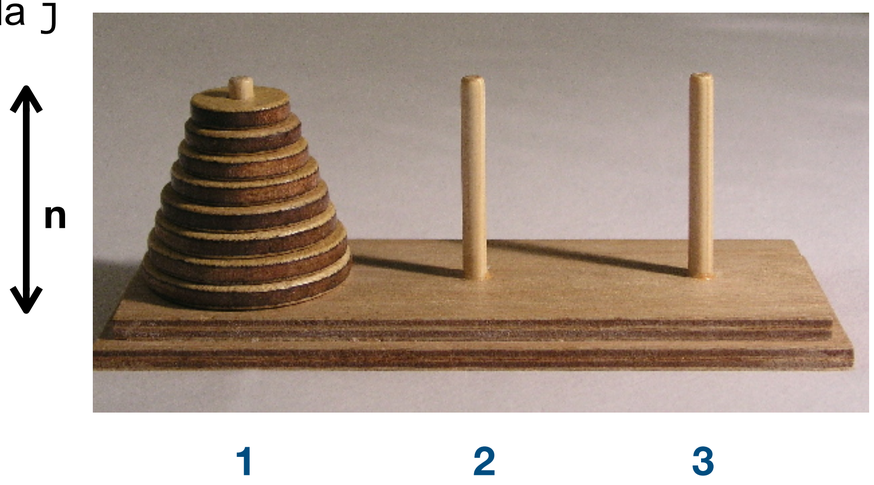


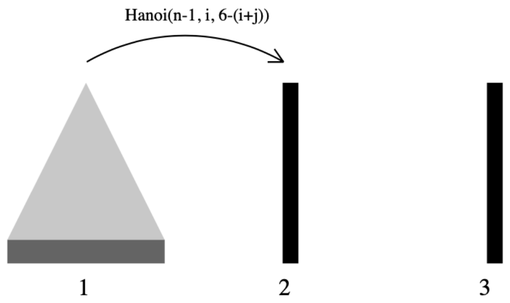
- la même solution avec une seule fonction et un paramètre supplémentaire `+/- 90` degrés

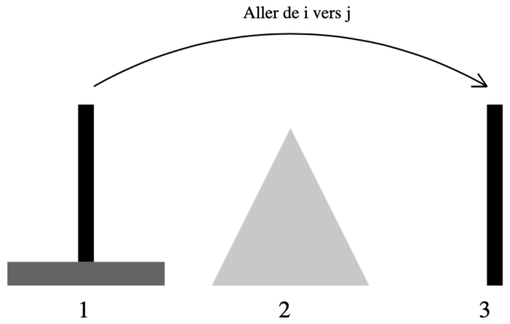
Récursivité et raisonnement inductif

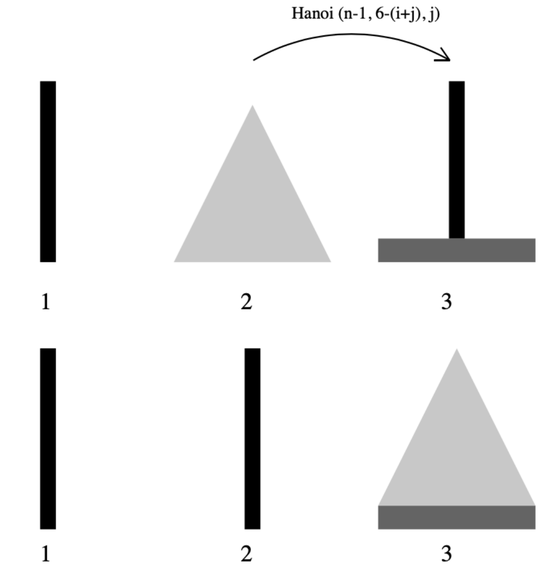
- le programme des tours de Hanoi: 3 piles numérotés 1, 2, 3
- on généralise le problème en mettant dans la variable `i` le numéro de la pile de départ et dans `j` le numéro de la pile d'arrivée
- on suppose le problème résolu avec n-1 rondelles pour aller de `i` à `j` quelles que soient les valeurs de `i` et `j` dans {1, 2, 3}
- on amène les n-1 rondelles du sommet de la pile i vers la pile `6-i-j`
- on amène la grosse rondelle de la pile `i` vers la pile `j`
- on amène les n-1 rondelles de la pile `6-i-`j vers la `j`

```
def hanoi (n, i, j) :
    if n > 0 :
       hanoi (n-1, i, 6 - i -j)
       print ("%d --> %d" %(i, j))
       hanoi (n-1, 6 - i - j, j)
```



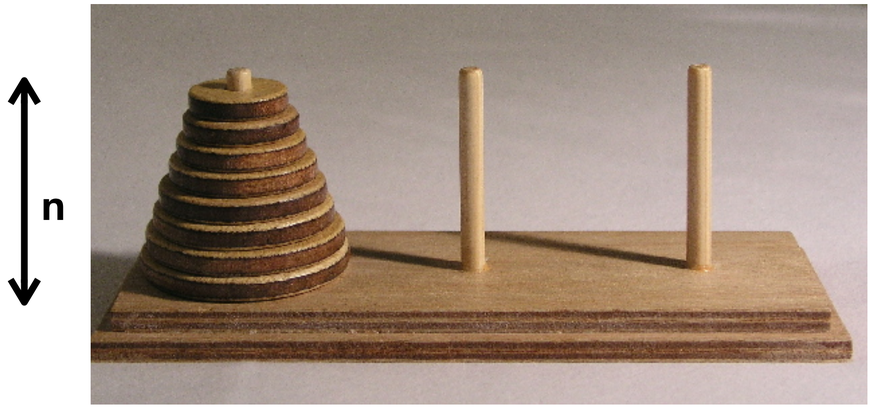






Récursivité et raisonnement inductif

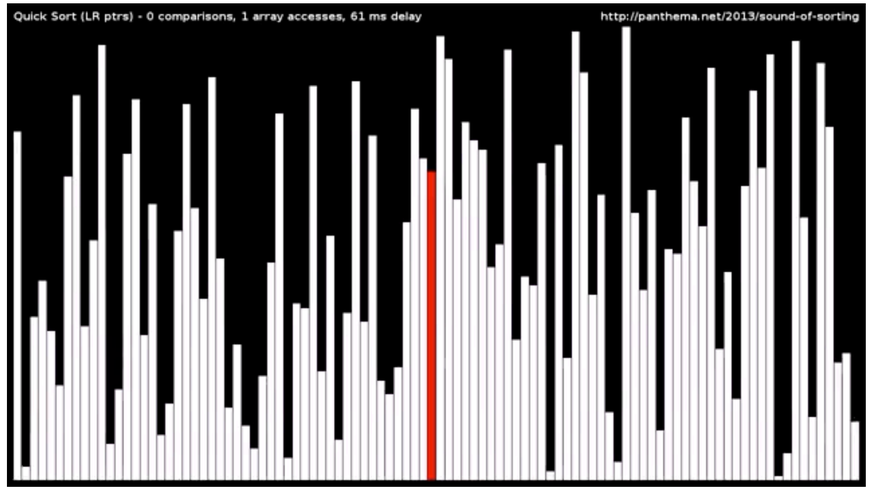
- le programme des tours de Hanoi: 3 piles numérotés A, B, C
- on généralise le problème en appelant x et y les piles de départ et d'arrivée (z est le nom de la troisième pile)
- on suppose le problème résolu avec n-1 rondelles pour aller de x à y en se servant de la pile z auxiliaire
- on amène les n-1 rondelles du sommet de la pile x vers la pile z
- on amène la grosse rondelle de la pile x vers la pile y
- on amène les n-1 rondelles de la pile z vers la y



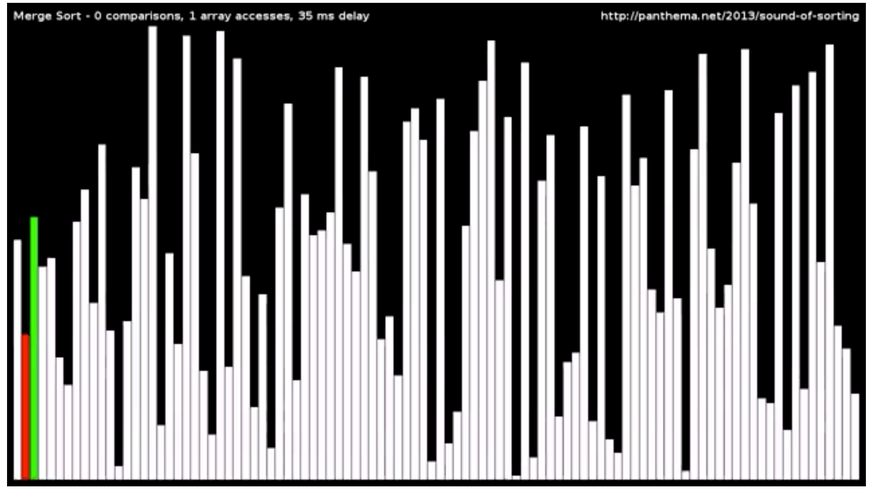

```
def hanoi (n, x, z, y) :
    if n > 0 :
       hanoi (n-1, x, y, z)
       print ("%s --> %s » %(x, y))
       hanoi (n-1, z, x, y)
```

Tri rapide — Tri fusion

- tri et récursivité: quicksort



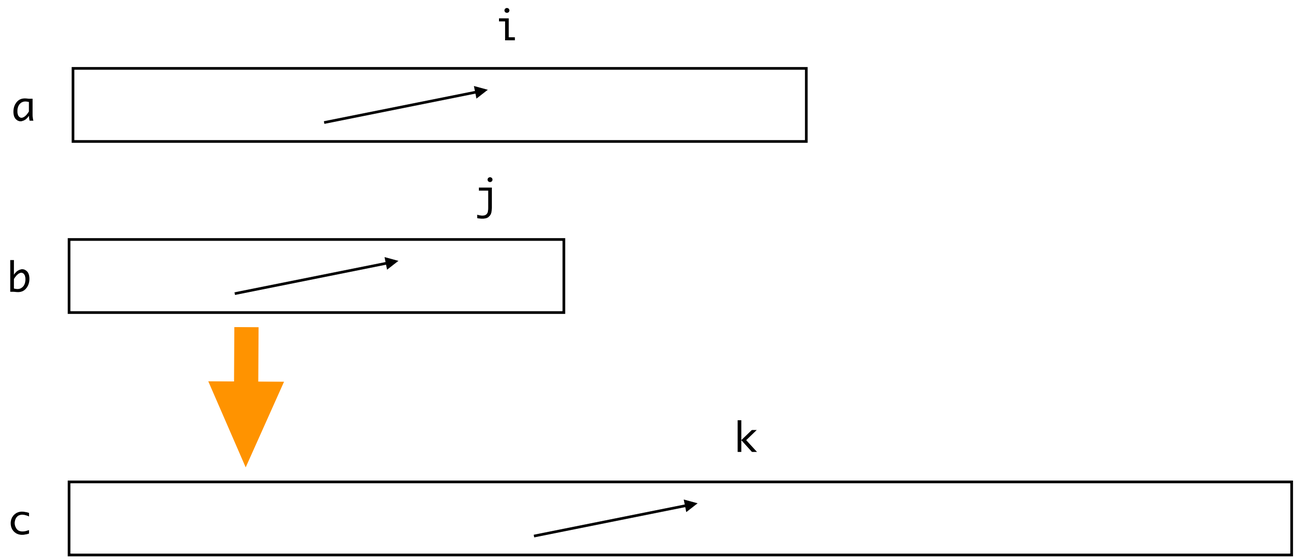
- tri et récursivité: mergesort



Tri fusion

- on veut fusionner 2 tableaux déjà triés en ordre croissant

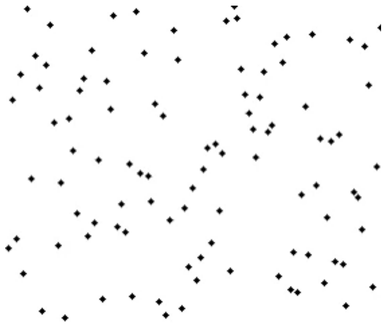
```
def fusion (a, b) :
    m = len (a); n = len (b)
    p = m + n
    c = p * [0]
    i = 0; j = 0; k = 0
    while i < m and j < n:
        if a[i] <= b[j] :
            c[k] = a[i]
            i = i + 1
            k = k + 1
        else :
            c[k] = b[j]
            j = j + 1
            k = k + 1
    while i < m :
        c[k] = a[i]
        i = i + 1
        k = k + 1
    while j < n :
        c[k] = b[j]
        i = j + 1
        k = k + 1
    return c
```

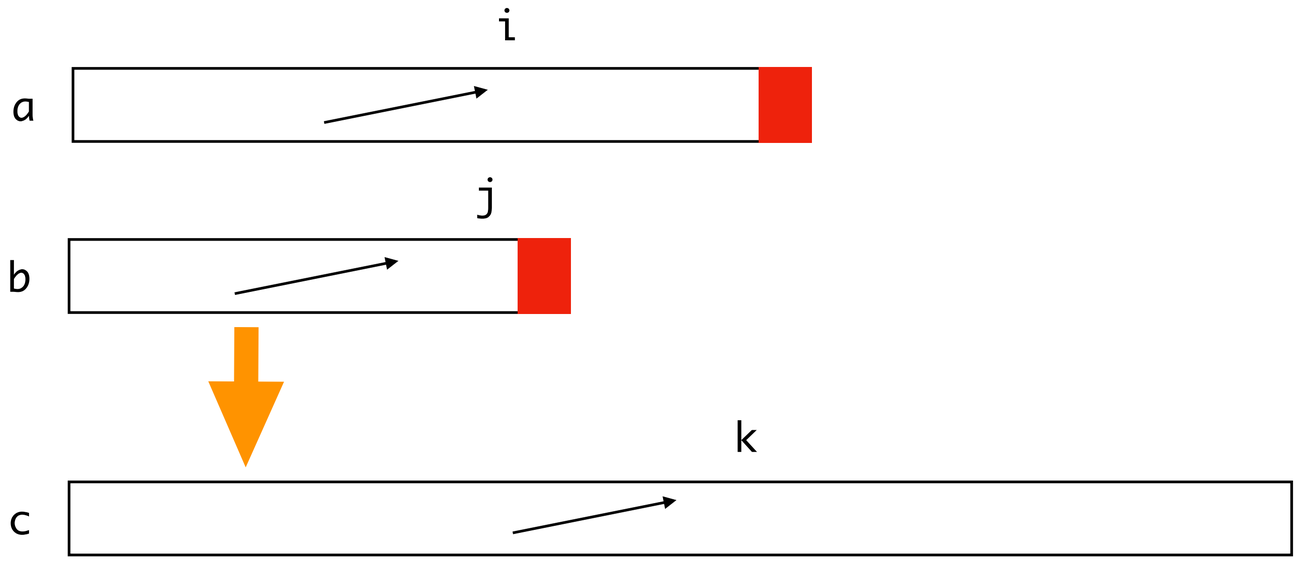


Tri fusion

- on veut fusionner 2 tableaux déjà triés en ordre croissant avec des sentinelles

```
def fusion1 (a, b) :
    m = len (a); n = len (b)
    a[m-1] = b[n-1] = MAX_INT
    p = m + n - 2
    c = p * [0]
    i = 0; j = 0
    for k in range (p) :
        if a[i] <= b[j] :
            c[k] = a[i]
            i = i + 1
        else :
            c[k] = b[j]
            j = j + 1
    return c
```



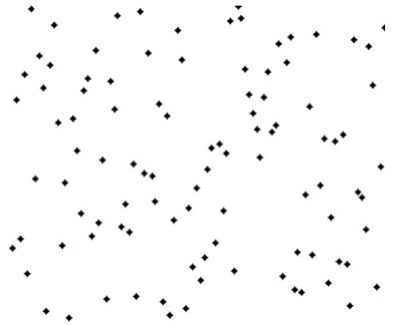


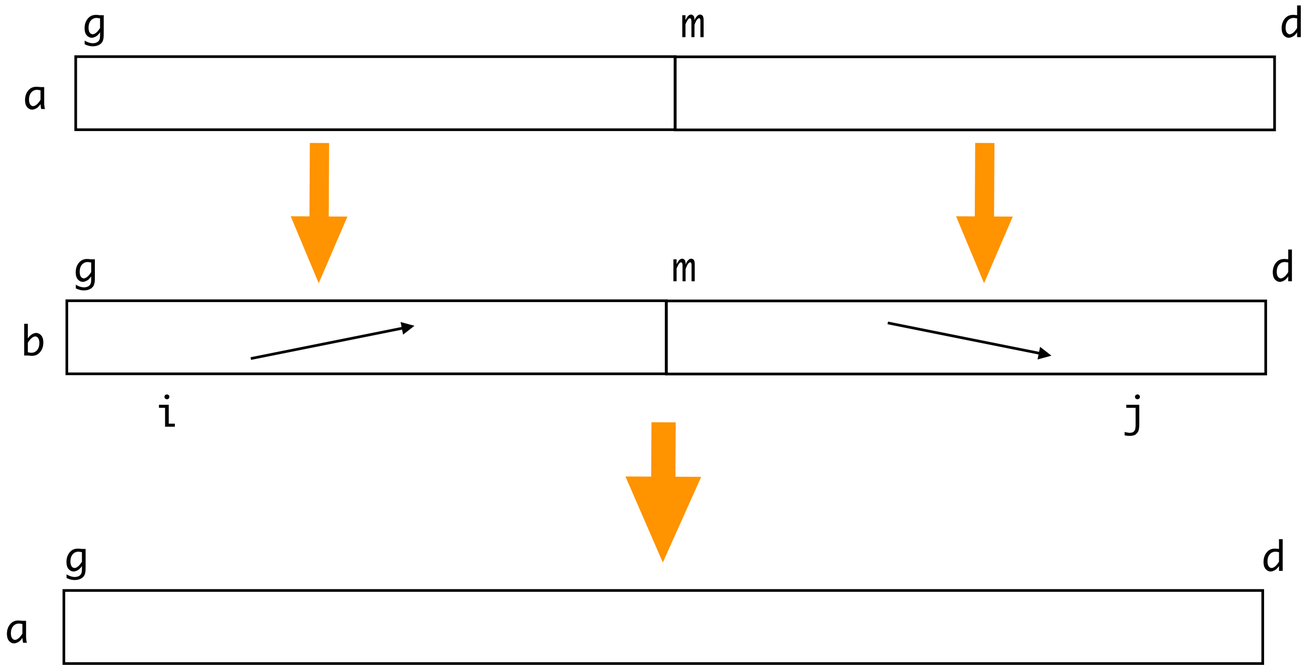
Tri fusion

- on trie les parties basse et haute — puis on fusionne les résultats

```
def tri_fusion (a, g, d, b) :
    if g < d - 1 :
        m = (g + d) // 2
        tri_fusion (a, g, m, b)
        tri_fusion (a, m, d, b)
        for i in range (g, m) :
            b[i] = a[i]
        for j in range (m, d) :
            b[m + d - 1 - j] = a[j]
        i = g; j = d - 1;
        for k in range (g, d) :
            if b[i] < b[j] :
                a[k] = b[i]
                i = i + 1
            else :
                a[k] = b[j]
                j = j - 1

def tri_Fusion (a) :
    n = len (a)
    b = n*[0]
    tri_fusion (a, 0, n, b)
```



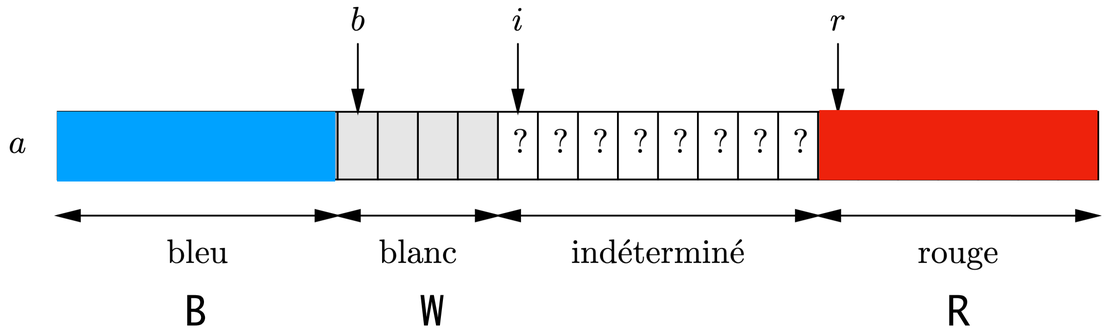


besoin d’un tableau annexe

Le problème du drapeau hollandais

- un tableau contient 3 couleurs aléatoirement rangées (bleu, blanc, rouge) — on doit remettre en ordre

```
def drapeau_hollandais (a) :
    b = 0; i = 0; r = len(a)
    while i < r :
        if a[i] == 'B' :
            t = a[b]; a[b] = a[i]; a[i] = t
            b = b + 1; i = i + 1
        elif a[i] == 'W' :
            i = i + 1
        else :  # a[i] == 'R'
            r = r - 1
            t = a[r]; a[r] = a[i]; a[i] = t
```



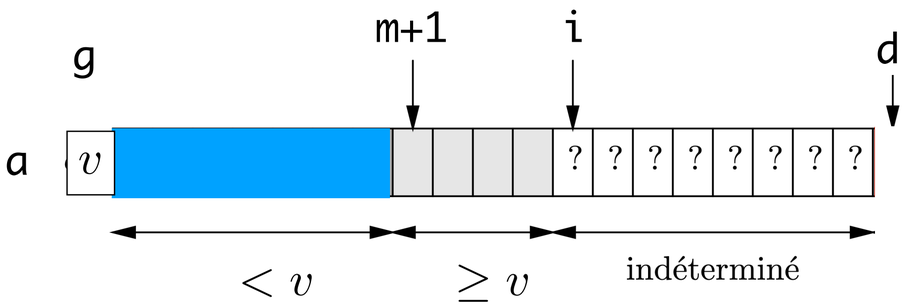

```
def aleatoire_hollandais (n) :
    a = n * [0]
    for i in range (n) :
       a[i] = random.choice (['B', 'W', 'R'])
    return a
```

Le tri rapide

- méthode de tri super simple (Tony Hoare 1960)

```
def tri_rapide (a, g, d) :
    if g < d - 1 :
      v = a[g]
      Partitionner le tableau autour de v et mettre v
      à sa position a[m]

      tri_rapide (a, g, m)
      tri_rapide (a, m + 1, d)
```

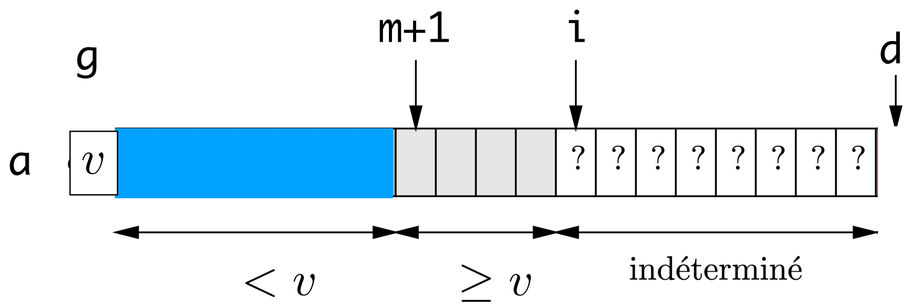


Le tri rapide

- méthode de tri super simple (Tony Hoare 1960)

```
def tri_rapide (a, g, d) :
    if g < d - 1 :
        v = a[g]
        m = g
        for i in range (g+1, d):
            if a[i] < v :
                m = m + 1
                t = a[m]; a[m] = a[i]; a[i] = t
        a[g] = a[m]; a[m] = v
        tri_rapide (a, g, m)
        tri_rapide (a, m+1, d)
```

```
def tri_Rapide (a) :
  tri_rapide (a, 0, len(a))
```



en moyenne QuickSort est bon

Exercice : modifier le programme avec 3 zones (plus petit que v, égal à v et plus grand que v)
le tri QuickSort devient alors stable (c'est-à-dire ne modifiant pas l'ordre d'arrivée des éléments égaux)

(indication: penser au drapeau hollandais)

Python est faiblement typé

- pour obtenir le type d’un objet Python

```
>>> type (a)
<class ‘list'>

>>> type ('abcde')
<class ‘str'>

>>> type (28)
<class ‘int'>

>>> type (4.7)
<class 'float'>


>>> 28 + 'abcde'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'


>>> def f (x) :
      if x == 1 :
        return 28
      else:
        return 'abcde'
```

Python ++

- entre accolades, on définit un ensemble

```
>>> {0, 1, 3, 1}
{0, 1, 3}

>>> {28, 'abcde', 4.6}
{28, 'abcde', 4.6}
```

- on ne peut indicer un ensemble, ni changer ses éléments

- on peut trouver l'adresse d'un objet Python

```
>>> id(a)
4310195808
>>> id(a[0])
4302126864
>>> id(a[1])
4302126896
>>> id(a[2])
4302126896
>>> id(a[3])
4302127024
```