Stabilité d'un tri

Nous utilisons le tri à bulles pour illustrer une propriété importante des tris: la **stabilité**.

Cette propriété s'intéresse à ce qu'il se passe lorsque l'algorithme de tri rencontre **deux éléments qu'il considère comme de même valeur**.

In [1]:
```
TAB = [ ("Aubert","Beatrice"),
        ("Caron","Alain"),
        ("Bonnet","Christine"),
        ("Bonnet","Anne"),
        ("Aubert","Alexandre"),
        ("Caron","Catherine"),
        ("Bonnet","Benoit"),
        ("Aubert","Denis"),
        ("Aubert","Carole"),
        ("Caron","Brigitte") ]
```

On peut envisager de trier cette liste par

- nom de famille
- prénom
- nom/prénom
- prénom/nom
- ...

Pour le rendre plus **générique** - capable de trier selon divers critères - réécrivons notre tri à bulles avec une fonction de comparaison `plus_petit` passée en paramètre

In [3]:
```
def tri_a_bulles(T,plus_petit):

    N = len(T)
    for k in range(N,1,-1):
        for i in range(0,k-1):
            if plus_petit( T[i+1], T[i] ):
                T[i],T[i+1] = T[i+1],T[i]
```

Pour trier la liste par ordre alphabétique des prénoms, il suffit d'écrire une fonction de comparaison sur le deuxième élément du tuple

In [4]:
```
def compare_prenoms(a,b):
    return a[1] < b[1]
```

In [5]:
```
T1 = TAB.copy()
tri_a_bulles(T1,compare_prenoms)
afficher(T1)
```

```
       Nom |     Prénom
-----------+-----------
     Caron |      Alain
    Aubert |  Alexandre
    Bonnet |       Anne
    Aubert |   Beatrice
    Bonnet |     Benoit
     Caron |   Brigitte
    Aubert |     Carole
     Caron |  Catherine
    Bonnet |  Christine
    Aubert |      Denis
```

Pour trier selon l'ordre alphabétique des noms, il suffit d'utiliser la fonction de comparaison

In [6]:
```
def compare_noms(a,b):
    return a[0] < b[0]
```

In [7]:
```
T2 = TAB.copy()
tri_a_bulles(T2,compare_noms)
afficher(T2)
```

```
       Nom |     Prénom
-----------+-----------
    Aubert |   Beatrice
    Aubert |  Alexandre
    Aubert |      Denis
    Aubert |     Carole
    Bonnet |  Christine
    Bonnet |       Anne
    Bonnet |     Benoit
     Caron |      Alain
     Caron |  Catherine
     Caron |   Brigitte
```

Appliquons le tri par nom de famille à la liste T1, précédemment triée par prénom.

In [8]:
```
T3 = T1.copy()
tri_a_bulles(T3,compare_noms)
afficher(T3)
```

```
       Nom |     Prénom
-----------+-----------
    Aubert |  Alexandre
    Aubert |   Beatrice
    Aubert |     Carole
    Aubert |      Denis
    Bonnet |       Anne
    Bonnet |     Benoit
    Bonnet |  Christine
     Caron |      Alain
     Caron |   Brigitte
     Caron |  Catherine
```

Pour chaque nom de famille, les personnes sont triées par ordre alphabétique des prénoms

Un algorithme de tri est stable s'il ne modifie pas l'ordre initial des clés identiques.

Déstabiliser un tri stable

Dans l'algoritme du tri à bulle, remplaçons
`if plus_petit( T[i+1], T[i] )` par
`if not plus_petit( T[i], T[i+1] ):`

Cela revient à remplacer $T_{i+1} < T_i$ par $\neg(T_i < T_{i+1})$, i.e. $T_{i+1} \leq T_i$.

In [9]:
```
def tri_instable(T,plus_petit):

    N = len(T)
    for k in range(N,1,-1):
        for i in range(0,k-1):
            if not plus_petit( T[i], T[i+1] ):
                T[i],T[i+1] = T[i+1],T[i]
```

Ce changement modifie le traitement des éléments égaux.

- Avec `T[i] > T[i+1]`, les éléments égaux restent en place.
- Avec `T[i] >= T[i+1]`, les éléments égaux sont échangés.

Cela rend le tri instable. Trions de nouveau par nom de famille la liste T1 triée par prénom.

In [10]:
```
T4 = T1.copy()
tri_instable(T4,compare_noms)
afficher(T4)
```

```
       Nom |     Prénom
-----------+-----------
    Aubert |      Denis
    Aubert |   Beatrice
    Aubert |  Alexandre
    Aubert |     Carole
    Bonnet |  Christine
    Bonnet |     Benoit
    Bonnet |       Anne
     Caron |  Catherine
     Caron |   Brigitte
     Caron |      Alain
```

Il est également possible de déstabiliser la fonction `tri_a_bulles` sans la modifier mais en lui fournissant en paramètre une mauvaise fonction de comparaison.

In [11]:
```
def compare_noms_instable(a,b):
    return a[0] <= b[0]

T5 = T1.copy()
tri_a_bulles(T5,compare_noms_instable)
afficher(T5)
```

```
       Nom |     Prénom
-----------+-----------
    Aubert |      Denis
    Aubert |   Beatrice
    Aubert |  Alexandre
    Aubert |     Carole
    Bonnet |  Christine
    Bonnet |     Benoit
    Bonnet |       Anne
     Caron |  Catherine
     Caron |   Brigitte
     Caron |      Alain
```

Stabiliser un tri instable

De même, il est toujours possible de stabiliser un algorithme de tri instable en procédant en trois étapes.

- Ajouter un indice

In [12]:
```
def ajoute_indice(T):
    return [ (nom, prenom, indice) for indice, ( nom, prenom )
in enumerate(T)]

T6 = ajoute_indice(T1)
print(T6)
```

```
[('Caron', 'Alain', 0), ('Aubert', 'Alexandre', 1), ('Bonne
t', 'Anne', 2), ('Aubert', 'Beatrice', 3), ('Bonnet', 'Beno
it', 4), ('Caron', 'Brigitte', 5), ('Aubert', 'Carole', 6),
('Caron', 'Catherine', 7), ('Bonnet', 'Christine', 8), ('Au
bert', 'Denis', 9)]
```

- Trier en utilisant l'indice pour différencier les éléments égaux par ailleurs.

In [13]:
```
def compare_noms_indices(a,b):
    (nom_a, prenom_a, indice_a) = a;
    (nom_b, prenom_b, indice_b) = b;
    if(nom_a < nom_b):
        return True
    elif nom_a == nom_b:
        return indice_a < indice_b
    else:
        return False

tri_instable(T6,compare_noms_indices)

print(T6)
```

```
[('Aubert', 'Alexandre', 1), ('Aubert', 'Beatrice', 3), ('A
ubert', 'Carole', 6), ('Aubert', 'Denis', 9), ('Bonnet', 'A
nne', 2), ('Bonnet', 'Benoit', 4), ('Bonnet', 'Christine',
8), ('Caron', 'Alain', 0), ('Caron', 'Brigitte', 5), ('Caro
n', 'Catherine', 7)]
```

- Oter l'indice des éléments triés

In [14]:
```
def retire_indice(T):
    return [ (nom, prenom) for (nom, prenom, indice) in T ]

T6 = retire_indice(T6)

afficher(T6)
```

```
       Nom |     Prénom
-----------+-----------
    Aubert |  Alexandre
    Aubert |   Beatrice
    Aubert |     Carole
    Aubert |      Denis
    Bonnet |       Anne
    Bonnet |     Benoit
    Bonnet |  Christine
     Caron |      Alain
     Caron |   Brigitte
     Caron |  Catherine
```

Visualisation

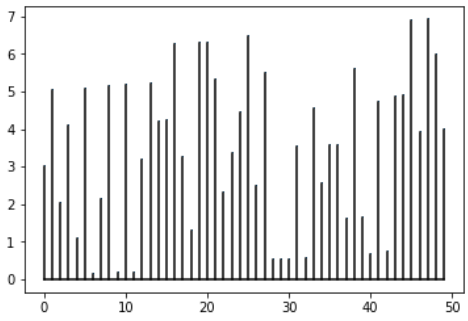
Pour analyser la stabilité des tris suivants, nous utiliserons une technique plus visuelle. Elle consiste à trier une liste de 50 nombres réels aléatoires entre 0 et 7 avec trois critères de tris distincts qui comparent

- les nombres originaux
- leurs parties entières
- leurs parties fractionnaires

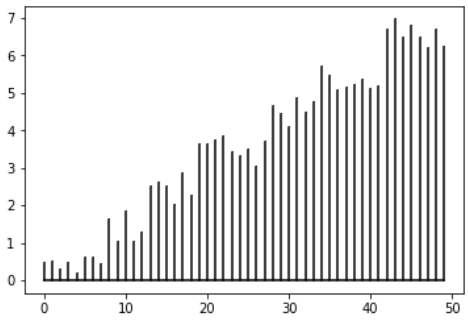
In [15]:
```
def comp(a,b):
    return a<b

def comp_int(a,b):
    return int(a) < int(b)

def comp_frac(a,b):
    return (a-int(a)) < (b - int(b))
```

Visualisons le tri de 50 nombres aléatoires

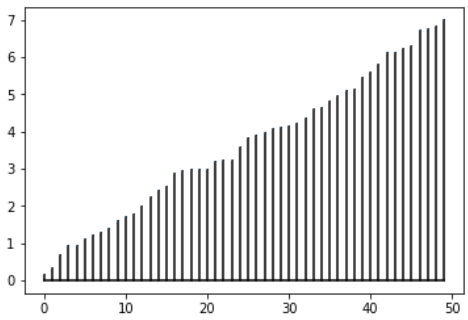
In [22]:
```
import numpy as np
import matplotlib.pyplot as plt

def comp_frac(a,b):
    return (a-int(a)) < (b - int(b))

T = np.random.uniform(0,7,50)
tri_a_bulles(T,comp_frac)
plt.stem(T,markerfmt=',',linefmt='black',basefmt='black')
plt.show()
```



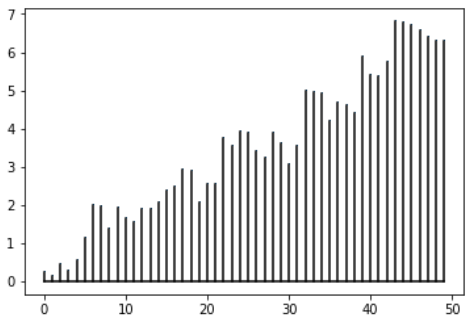
In [17]:
```
def comp_int(a,b):
    return int(a) < int(b)

T = np.random.uniform(0,7,50)
tri_a_bulles(T,comp_int)
plt.stem(T,markerfmt=',',linefmt='black',basefmt='black')
plt.show()
```



In [18]:
```
T = np.random.uniform(0,7,50)
tri_a_bulles(T,comp_frac)
tri_a_bulles(T,comp_int)
plt.stem(T,markerfmt=',',linefmt='black',basefmt='black')
plt.show()
```



In [23]:
```
T = np.random.uniform(0,7,50)
tri_instable(T,comp_frac)
tri_instable(T,comp_int)
plt.stem(T,markerfmt=',',linefmt='black',basefmt='black')
plt.show()
```



ASD1 Notebooks on GitHub.io
(https://ocuisenaire.github.io/ASD1-notebooks/)