4_tris

December 1, 2025

[1]:
```
import time,random as rd
import numpy as np
```

1 2. Présentation des fonctions de tri natives en Python

[2]:
```
n = 20
L = [rd.randint(1,n) for i in range(n)]
print(L)
# L.sort()
LL=sorted(L)
print(L)
print(LL)
```

```
[9, 18, 17, 11, 15, 13, 7, 15, 16, 7, 4, 19, 6, 19, 15, 7, 15, 12, 2, 19]
[9, 18, 17, 11, 15, 13, 7, 15, 16, 7, 4, 19, 6, 19, 15, 7, 15, 12, 2, 19]
[2, 4, 6, 7, 7, 7, 9, 11, 12, 13, 15, 15, 15, 15, 16, 17, 18, 19, 19, 19]
```

[3]:
```
n = 20
L = [rd.randint(1,n) for i in range(n)]
print(L)
LL = sorted(L)
print(L)
print(LL)
```

```
[14, 20, 13, 11, 5, 12, 15, 11, 6, 3, 11, 13, 4, 1, 16, 3, 11, 9, 16, 15]
[14, 20, 13, 11, 5, 12, 15, 11, 6, 3, 11, 13, 4, 1, 16, 3, 11, 9, 16, 15]
[1, 3, 3, 4, 5, 6, 9, 11, 11, 11, 11, 12, 13, 13, 14, 15, 15, 16, 16, 20]
```

Fonction permettant de tester les tris

[4]:
```
def test_tri(fonc,aux=False):
    '''
    teste si la fonction fonc effectue un tri correct
    attention, la stabilité n'est pas testée
    '''
    l=[]
    for _ in range(10**3):
        a=rd.randint(1,10**4)
        if a not in l:
```

```
            l.append(a)
    l1=sorted(l)
    fonc(l)
    if not l==l1:
        al=np.array(l)
        al1=np.array(l1)
        print(al-al1,al)
```

2 3. Algorithmes de tri au programme

2.1 3.1. Tri sélection

Cette fonction retourne l'indice i associé au minimum de l entre les indices i_d et i_f.

[5]:
```
def trouve_min(l:list,i_d:int,i_f:int) -> int:
    i_prov=i_d
    for i in range(i_d,i_f):
        if l[i]<l[i_prov]:
            i_prov=i
    return i_prov
```

On applique ensuite le tri en parcourant la liste, déterminant le minimum et le plaçant en première position.

[6]:
```
def tri_selection(l:list):
    n=len(l)
    for i in range(0,n-1):
        j=trouve_min(l,i,n)
        l[i],l[j]=l[j],l[i]
```

[7]:
```
l=[3,3.0,2,5,12,16,1,14]
tri_selection(l)
print(l)
```

```
[1, 2, 3.0, 3, 5, 12, 14, 16]
```

[8]:
```
test_tri(tri_selection)
```

2.2 3.2. Tri insertion

[9]:
```
def tri_insertion(l:list):
    for indice,valeur in enumerate(l):
        j=indice
        while j>0 and valeur<l[j-1]:
            l[j]=l[j-1]
            j=j-1
        l[j]=valeur
```

[10]:
```
l=[3,3.0,2,5,12,16,1,14]
tri_insertion(l)
print(l)
```

```
[1, 2, 3, 3.0, 5, 12, 14, 16]
```

[11]:
```
test_tri(tri_insertion)
```

2.3 3.3. Tri fusion

Version avec liste auxiliaire

[12]:
```
def tri_fusion_aux(l:list):
    if len(l)==1:
        return l
    l1,l2=l[:len(l)//2],l[len(l)//2:]
    l1,l2,l_sort=tri_fusion_aux(l1),tri_fusion_aux(l2),[]
    n1,n2,i1,i2=len(l1),len(l2),0,0
    while i1<n1 or i2<n2:
        if i2==n2 or (i1!=n1 and l1[i1]<l2[i2]):
            l_sort.append(l1[i1])
            i1+=1
        else:
            l_sort.append(l2[i2])
            i2+=1
    return l_sort
```

[13]:
```
l=[3,3.0,2,5,12,16,1,14,0]
print(tri_fusion_aux(l))
```

```
[0, 1, 2, 3.0, 3, 5, 12, 14, 16]
```

2.4 3.4. Tri rapide

2.4.1 3.4.1. Version avec liste auxiliaire

[14]:
```
def tri_rapide_aux(l:list)->list:
    if l==[]:
        return l
    pivot=l[0]
    linf,lsup=[],[]
    for el in l[1:]:
        if el<pivot:
            linf.append(el)
        elif el>=pivot:
            lsup.append(el)
    return tri_rapide_aux(linf)+[pivot]+tri_rapide_aux(lsup)
```

[15]:
```
l=[3,3.0,2,5,12,16,1,14]
print(tri_rapide_aux(l))
```

```
[1, 2, 3, 3.0, 5, 12, 14, 16]
```

2.4.2 3.4.2. Version en place

[16]:
```
def tri_rapide_rec(l:list,i:int,j:int):
    if i<j:
        p,q=i,i
        for k in range(i+1,j+1):
            if l[k]<=l[p]:
                q+=1
                l[q],l[k]=l[k],l[q]
        l[p],l[q]=l[q],l[p]
        tri_rapide_rec(l,i,q-1)
        tri_rapide_rec(l,q+1,j)

def tri_rapide_place(l:list):
    l=tri_rapide_rec(l,0,len(l)-1)
```

[17]:
```
l=[3,3.0,2,5,12,16,1,14,0]
tri_rapide_place(l)
print(l)
```

```
[0, 1, 2, 3.0, 3, 5, 12, 14, 16]
```

[18]:
```
test_tri(tri_rapide_place)
```

[ ]:

[ ]: