Comparaison de tris développés en Python

par Guillaume Duriaud

Date de publication : 04/10/2006

Dernière mise à jour : 09/12/2008

Comparaison de divers tris (tri natif, tri à bulles, tri par sélection, tri radix, tri par insertion, tri fusion) d'entiers de 64 bits

I - Prérequis

Langage: Python

II - Description

Dans le fichier *TriPython.py* sont développés plusieurs algorithmes de tris avec une version montre l'algorithme complet et parfois une deuxième version qui utilise les avantages de Python (tout en se conformant au bon algorithme du tri).
La fonction ***Swap*** permet de permuter 2 éléments d'une liste. Mieux vaut parfois directement réécrire la fonction dans l'algorithme de tri pour éviter trop d'appels à la fonction qui nuirait aux performances.
Je ne vais pas expliquer les différents tris. Vous trouverez les explications des algorithmes dans de nombreux livres ou sites Internet. Ici, j'ai pour le moment implémenté les tris:

- Tri à bulles
- Tri par sélection
- Tri par insertion
- Tri fusion
- Tri pas base (Tri radix)

Pour le tri radix, je propose 2 solutions, la solution classique à base d'entiers et une autre solution qui consiste tout d'abord à transformer tous les nombres en hexadécimaux puis de faire le tri radix classique. L'avantage étant qu'un nombre en base 16 comportera moins de chiffres qu'un nombre en base 10. On effectuera donc moins de parcours complets de la liste à trier.

III - Source

TriPython.py

```
# -*- coding: cp1252 -*-
__version__ = (0, 2, 1)
__build__ = (0, 0)
__date__ = (2006, 10, 7)
__author__ = ('Guillaume', 'Duriaud')

def Swap(l,i,j):
    """ Created: 2005.11.08 - Updated:
        echange 2 valeurs d'une liste """
    t=l[i]
    l[i]=l[j]
    l[j]=t

def Merge(l1, l2):
    """ Created: 2006.10.07 - Updated:
        Tri 2 listes triées """
    l = []
    i = j =0
    n1=len(l1)
    n2=len(l2)
    while True:
        if i<n1 and j<n2:
            if l1[i]<l2[j]:
                l.append(l1[i])
                i+=1
            else:
                l.append(l2[j])
                j+=1
        elif i>=n1:
            l.extend(l2[j:])
            break
        else:
            l.extend(l1[i:])
            break
    return l


def Tri_Bulle(l):
    """ Created: 2005.11.08 - Updated:
        Tri bulle - O(n^2)"""
    for i in xrange(len(l)):
        for j in reversed(xrange(i,len(l))):
            if l[j]<l[j-1]:
                t=l[j]
                l[j]=l[j-1]
                l[j-1]=t


def Tri_Selection(l):
    """ Created: 2005.11.08 - Updated:
        Tri Selection - O(n^2) """
    for i in xrange(len(l)-1):
        mini=i
        for j in xrange(i+1,len(l)):
            if l[j]<l[mini]: mini=j
        Swap(l,i,mini)

def Tri_Selection_Optimise(l):
    """ Created: 2006.10.05 - Updated:
        Tri Selection - O(n^2) """
    for i in xrange(len(l)-1):
        mini=i
        lmini = l[i]
        for j in xrange(i+1,len(l)):
            if l[j]<lmini:
                mini=j
                lmini = l[j]
```

TriPython.py

```
        l[mini], l[i] = l[i], lmini

def Tri_Insertion(l):
    """ Created: 2005.11.08 - Updated:
        Tri Insertion - O(n^2) """
    for i in xrange(len(l)):
        m=l[i]
        for j in xrange(i+1):
            if m<l[j]:
                for k in reversed(xrange(j+1,i+1)):
                    l[k]=l[k-1]
                l[j]=m
                break

def Tri_Insertion_Optimise(l):
    """ Created: 2006.10.05 - Updated: """
    for i in xrange(len(l)):
        m=l[i]
        for j in xrange(i+1):
            if m<l[j]:
                l[j+1:i+1] = l[j:i]
                l[j]=m
                break

def Tri_Radix10(l):
    """ Created: 2005.11.08 - Updated:
        Tri Radix - O(n) """
    maxi=max(l)
    ll = []
    for i in xrange(len(str(maxi))):
        ll.append([])
    for j in xrange(len(str(maxi))):
        for k in xrange(11):
            ll[j].append([])
    for i in l:
        ll[len(str(i))-1][10].append(i)
    for i in xrange(len(str(maxi))):
        for j in reversed(xrange(i+1)):
            for k in ll[i][10]:
                ll[i][int(str(k)[j])].append(k)
            ll[i][10]=[]
            for m in xrange(10): ll[i][10].extend(ll[i][m])
            for m in xrange(10): ll[i][m]=[]
    lll = []
    for i in xrange(len(str(maxi))):
        lll.extend(ll[i][10])
    l[:] = lll[:]


def Tri_Radix16(l):
    """ Created: 2006.10.07 - Updated:
        Tri Radix en passant en hexadécimaux - O(n) """
    for i in range(len(l)): l[i] = hex(l[i]).replace('L','')
    maxi=max(l)
    nbmaxi = len(maxi)-2
    ll = []
    for i in xrange(nbmaxi): ll.append([])
    for j in xrange(nbmaxi):
        for k in xrange(17):
            ll[j].append([])
    for i in l:
        ll[len(str(i))-3][16].append(i)  ## Plantage parfois encore non compris
##        except Exception, err: print err, i, nbmaxi, maxi
    for i in xrange(nbmaxi):
        for j in reversed(xrange(2,i+3)):
            for k in ll[i][16]:
                ll[i][int('0x'+str(k)[j],16)].append(k)
            ll[i][16]=[]
            for m in xrange(16): ll[i][16].extend(ll[i][m])
            for m in xrange(16): ll[i][m]=[]
    lll = []
```

TriPython.py

```
    for i in xrange(nbmaxi): lll.extend(ll[i][16])
    for i in range(len(l)): l[i] = int(lll[i],16)


def Tri_Fusion(l):
    """ Created: 2006.10.07 - Updated:
        Tri Fusion - O(nlog(n)) """
    def Tri_Fusion_interne(l):
        if len(l)<2: return l
        return Merge( Tri_Fusion_interne(l[:len(l)//2]), Tri_Fusion_interne(l[len(l)//2:]))
    l[:] = Tri_Fusion_interne(l)
```

test.py

```
# -*- coding: cp1252 -*-
import time
import random
from TriPython import *

random.seed()

p2 = 2**64
n = input("Nombre d'elements a trier\n")
liste = []
for i in range(n): liste.append(random.randint(0, p2))

lsort = list(liste)
a=time.clock()
lsort.sort()
b=time.clock()
print "Tri natif: ", b-a

lbulle = list(liste)
a=time.clock()
Tri_Bulle(lbulle)
b=time.clock()
print "Tri Bulle: ", b-a

lselection = list(liste)
a=time.clock()
Tri_Selection(lselection)
b=time.clock()
print "Tri par Selection: ", b-a

lselection2 = list(liste)
a=time.clock()
Tri_Selection_Optimise(lselection2)
b=time.clock()
print "Tri par Selection optimisé: ",  b-a

linsertion = list(liste)
a=time.clock()
Tri_Insertion(linsertion)
b=time.clock()
print "Tri par Insertion: ", b-a

linsertion2 = list(liste)
a=time.clock()
Tri_Insertion_Optimise(linsertion2)
b=time.clock()
print "Tri par Insertion optimisé: ", b-a

lfusion = list(liste)
a=time.clock()
Tri_Fusion(lfusion)
b=time.clock()
print "Tri Fusion: ", b-a

lradix10 = list(liste)
```

test.py

```
a=time.clock()
Tri_Radix10(lradix10)
b=time.clock()
print "Tri Radix10: ", b-a

lradix16 = list(liste)
a=time.clock()
Tri_Radix16(lradix16)
b=time.clock()
print "Tri Radix16: ", b-a


## Vérification que les tris trient correctement
print lsort == lbulle == lselection == linsertion == linsertion2 == lradix10 == lselection2 ==
 lfusion == lradix16


raw_input("Appuyez sur une touche pour quitter")
```

IV - Résultats

On peut ainsi comparer les performances de ces tris. Je me suis contenté de comparer le temps d'exécution sur un P4 centrino 1,7 Ghz - 2 Go de RAM avec Python 2.6.1. Les nombres à trier sont des entiers compris entre 0 et 2^64-1. Tous les temps donnés sont en secondes.

Algorithme	1000	5000	10000	50000	100000	500000	1000000
Tri natif	$6,7.10^{-4}$	$4,1.10^{-3}$	$8,9.10^{-3}$	$5,4.10^{-2}$	$1,3.10^{-1}$	$8,4.10^{-1}$	1,9
Tri à bulles	$2.3.10^{-1}$	5,6	$2,2.10^{1}$	$5,7.10^{2}$	$2,8.10^{3}$		
Tri par sélection	$1,1.10^{-1}$	2,8	$1,1.10^{1}$	$2,7.10^{2}$	$1,3.10^{3}$		
Tri par sélection optimsé	$8,1.10^{-2}$	2,0	8,0	$2,0.10^{2}$	$1,0.10^{3}$		
Tri par insertion	$9,2.10^{-2}$	2,3	9,0	$2,3.10^{2}$	$1,2.10^{3}$		
Tri par insertion optimisé	$4,4.10^{-2}$	1,3	5,0	$1,4.10^{2}$	$9,8.10^{2}$		
Tri fusion	$1,0.10^{-2}$	$6,2.10^{-2}$	$1,3.10^{-1}$	$7,4.10^{-1}$	1,6	9,2	$2,0.10^{1}$
Tri par base10 (Tri radix10)	$5,8.10^{-2}$	$2,8.10^{-1}$	$5,6.10^{-1}$	2,9	6,2	$3,3.10^{1}$	$6,9.10^{1}$
Tri par base16 (Tri radix16)	$2,5.10^{-2}$	$1,2.10^{-1}$	$2,4.10^{-1}$	1,4	3,2	$1,7.10^{1}$	$3,6.10^{1}$

V - Téléchargement

Version	Date	Taille	Mode FTP	Mode HTTP de secours
0.2.1.0.0	2006.10.07	1.73 Ko	**TriPython.zip**	**TriPython.zip**