43-PROGR-ordenacion-listas

October 12, 2017

In [1]:
```
randint(-1000,1000)
```

Out[1]:
```
92
```

"Mergesort"

In [2]:
```
def intercalar(L,L1,L2):
    if len(L1) == 0 or len(L2) == 0:
        return L+L1+L2
    elif L1[0] <= L2[0]:
       L.append(L1[0])
       L1.pop(0)
        #print L,L1,L2
       intercalar(L,L1,L2)
    else:
        L.append(L2[0])
        L2.pop(0)
        #print L,L1,L2
        intercalar(L,L1,L2)
    return L+L1+L2
```

In [3]:
```
intercalar([],[10,11,13],[-6,9,10,12])
```

Out[3]:
```
[-6, 9, 10, 10, 11, 12, 13]
```

In [4]:
```
def mergesort(L):
    n = len(L)
    if n == 0 or n == 1:
        return L
    #elif n == 2:
    #    if L[0] <= L[1]:
     #       return L
    #    else:
     #       return [L[1],L[0]]
    else:
         m = n//2
         return intercalar([],mergesort(L[:m]),mergesort(L[m:]))
```

In [5]:
```
mergesort([2,7,5,1,-1,3])
```

```
Out[5]: [-1, 1, 2, 3, 5, 7]
```

```
In [6]: L = [randint(-1000,1000) for _ in srange(800)]
        L1 = list(L)
        L2 = list(L)
        L3 = mergesort(L1)
        L2.sort()
        L3 == L2
        print L3[:10],L2[:10],L[:10]
```

```
Out[6]: [-997, -991, -980, -978, -972, -969, -960, -958, -958, -956] [-997, -991, -980, -978,
        -972, -969, -960, -958, -958, -956] [-429, -768, -230, 535, -953, 493, -923, 699, 373,
```

"Insertion sort"

En primer lugar definimos una función para intercambiar un entero con el anterior si éste es mayor:

```
In [7]: def intercambiar(L,i,j):
            if  L[i] > L[j] and i<j:
                L[i],L[j] = L[j],L[i]
            return L
```

```
In [8]: intercambiar([1,3,2,1],2,3)
```

```
Out[8]: [1, 3, 1, 2]
```

Para que la lista quede ordenada hay que comparar, como máximo, cada entero, en algún momento del programa, con cada uno de los otros $n-1$, lo que sugiere un bucle doble, con cada uno de n vueltas :

```
In [9]: def insertionsort(L):
            n = len(L)
            for i in srange(n):
                for j in srange(n):
                     intercambiar(L,i,j)
            return L
```

```
In [10]: insertionsort([3,1,-7,5,2,0,-1,-2,-3])
```

```
Out[10]: [-7, -3, -2, -1, 0, 1, 2, 3, 5]
```

```
In [11]: L = [randint(-1000,1000) for _ in srange(800)]
         L1 = list(L)
         L2 = list(L)
         L3 = insertionsort(L1)
         L2.sort()
         print L3 == L2
         print L3[:10],L2[:10],L[:10]
```

```
Out[11]: True
         [-1000, -1000, -999, -999, -998, -997, -992, -992, -990, -989] [-1000, -1000, -999, -
         -998, -997, -992, -992, -990, -989] [-783, 806, -990, -980, -899, -815, -504, 489, -2
         197]
```

Comparación de tiempos

```
In [12]: L = [randint(-1000,1000) for _ in xrange(800)]
         L1 = list(L)
         L2 = list(L)

In [13]: time LL1 = mergesort(L1)

Out[13]: Time: CPU 0.01 s, Wall: 0.01 s

In [14]: time LL2 = insertionsort(L2)

Out[14]: Time: CPU 0.23 s, Wall: 0.23 s

In [15]: LL1 == LL2

Out[15]: True
```

La comparación de tiempos no se puede hacer más amplia porque el *mergesort* se encuentra enseguida con el límite en la profundidad de la recursión. De todas formas parece claro, incluso a priori, que es más eficiente que el otro.