Mini apunte de Python para el Taller de Sorting

Algoritmos y Estructuras de Datos II

26 de octubre de 2016

1. Sintaxis

Recuerden, **TODOS** los lugares donde en C++ ponen llaves, acá tienen que indentar el código apretando tab.

1.1. If

```
if a > 0:
    print "a > 0"
elif a < 0:
    print "a < 0"
elif not(a > 0 or a < 0):
    print "a == 0"
else:
    print "Como llegaste?"
```

1.2. While

```
i = 0
while i < 100:
    i+=1 # no vale hacer i-- o i++
```

1.3. For

Cuando ponen `range(a,b)`, van a estar recorriendo desde a, hasta b **sin incluir b**.

```
# for(int i=5;i<10;i++)
for i in range(5, 10):
    print random.randint(0, i)
```

1.4. Subíndices

Se indexa basado en 0 y se tienen las siguientes operaciones:

```
vec = [41, 12, 11, 7, 3 ,6]
elem_prim = vec[0]
ele_ult = vec[-1]
ele_ult2 = vec[len(vec)-1]
sub_vec = vec[1:3] # el ultimo es no inclusive
prefix = vec[:2]
sufix = vec[3:]
```

1.5. Una función completa

Vamos a escribir la función máximo.
La precondición es que el arreglo es no vacío.

```
def maximo(l):
    res = l[0]
    for i in range(1, len(l)):
        if res < l[i]:
            res = l[i]
    return res
```

2. Tips

2.1. Crear un arreglo temporal

La función `crear_temporal` construye un arreglo temporal del tamaño del arreglo original que *copia* los elementos. **No usen otra forma para crear arreglo porque quedan descalificados.**

```
arreglo_temporal = a.crear_temporal()
```

2.2. Swap

Para swapear el contenido de dos variables:

```
a, b = b, a
```

En el caso de arreglo

```
arreglo[i], arreglo[j] = arreglo[j], arreglo[i]
```

3. Como correr los algoritmos

Tienen que ejecutar este comando

```
python test.py <nombre-algoritmo>
```

Así como esta arriba les va a preguntar que quieren hacer. Tienen las siguientes opciones:

- `-l`:
  les prueba su algoritmo con una lista aleatoria de 10 elementos
- `-L`:
  les prueba su algoritmo con una lista aleatoria de 5000 elementos
- `-c`:
  les prueba su algoritmo con una lista pasada por parámetro de la forma: `[3, 12, 414, 12]`
- `-e`:
  corre tu algoritmo varias veces y te estima cual es la constante

Ejemplos:

```
\: python test.py mergesort -l

nop, algo fallo
la lista original era: [0, 6, 8, 7, 9, 4, 5, 1, 3]
tu algoritmo la dejo asi: [0, 6, 8, 7, 9, 4, 5, 1]
```

```
\: python test.py mergesort -e

Ejecutando el algoritmo...
Tu algoritmo es O(n*log(n)) con constante 6.74
```

4. Complejidad

• La operación `a.crear_temporal()` tiene costo O(n) donde n es el tamaño del arreglo pasado por parámetro.
• La operación `arr[i:j]` tiene costo O($j - i$)