Auxiliar 9

Listas Indexadas, For y While.

Profesor: Patricio Inostroza
Auxiliares: Valeria V. Franciscangeli, Pablo Skewes, Gustavo Rivera

Problema 1 - For y While: versatilidad para resolver problemas.

a) Se tiene una lista de elementos de la estructura Pokemon (la misma de los auxiliares anteriores). Cree una función que itere sobre esta lista y devuelva otra que tenga sólo aquellos que son de tipo fuego, agua o hierba, iterando de 3 formas distintas: con un *for* usando de los elementos de la lista, con un *for* usando los índices de la lista y finalmente con un *while*.

```
    # Pokemon: Nombre(str) Npokedex(int) Tipo(str) HP(int)
    estructura.crear('Pokemon','Nombre Npokedex Tipo HP')
```

b) Sabemos que la siguiente función sirve para imprimir el número de dígitos de un número:

```
# digitos: int -> int
# cuenta los digitos de un número
def digitos_rec(num):
    if num//10==0:
      return 1
     else:
      return 1 + digitos2(num//10)
```

Cree una función que haga lo mismo pero sin recursividad.

Problema 2 - Algoritmos de Ordenamiento

Un algoritmo de ordenamiento recibe una lista y la ordena. Por ejemplo:

```
>>> L = [2,5,4,8,1,9,3,6,0,7]
>>> print(L)
[2, 5, 4, 8, 1, 9, 3, 6, 0, 7]
>>> ordenar(L)
>>> print(L)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

En este problema se busca programar dos algoritmos de ordenamiento y compararlos.

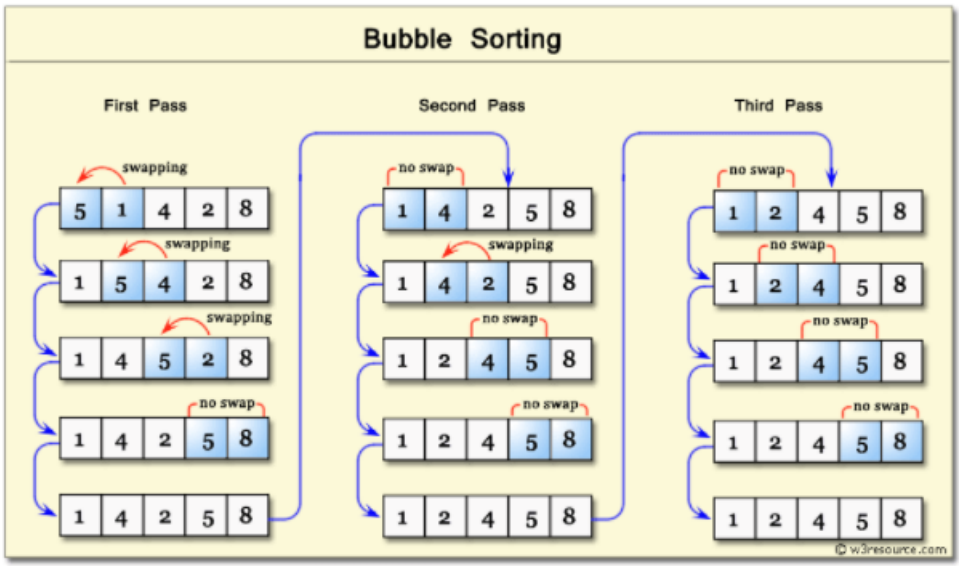
a) Primero, cree una función que reciba un entero y devuelva una lista de ese tamaño llenada con números aleatoriamente.

Hint: cree una lista con range() y luego puede usar el método "shuffle" del módulo *random*. ej:

```
>>> L = [0,1,2] ;
>>> random.shuffle(L)
>>> print(L)
[1,0,2]
```

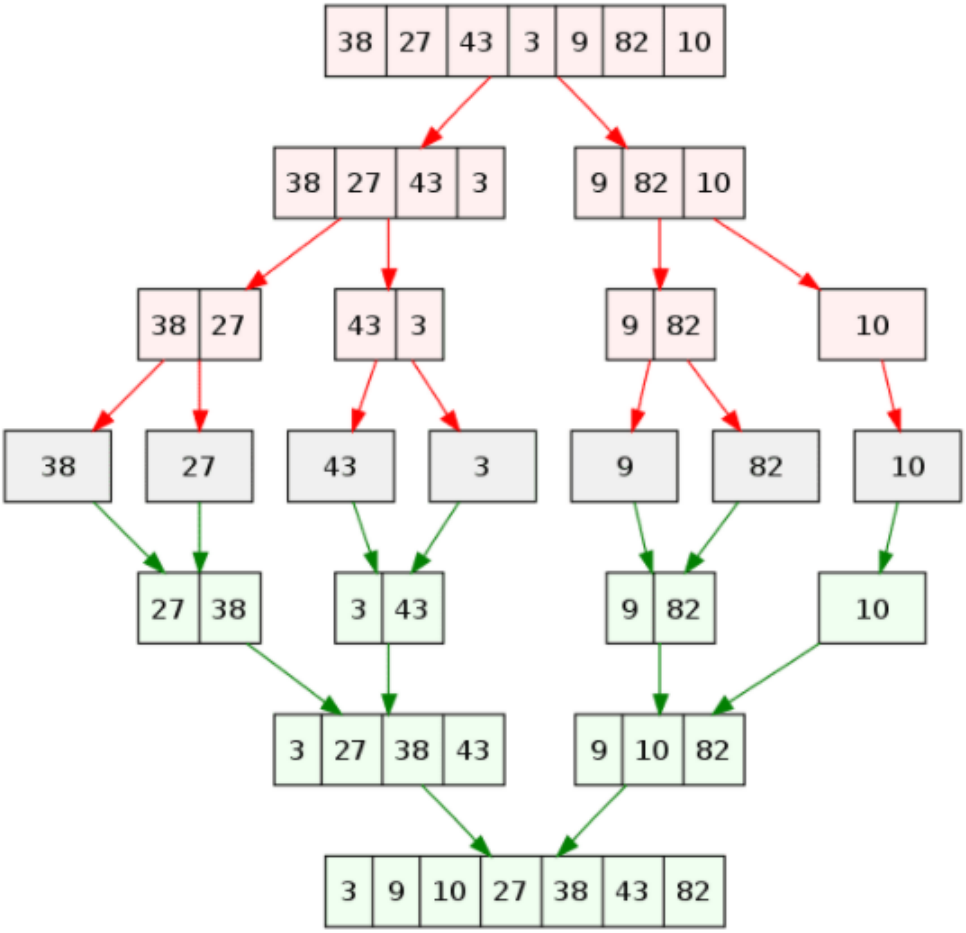
b) **BubbleSort**

Programe un algoritmo que ordene una lista con **BubbleSort**.



c) **MergeSort**

Programe un algoritmo que ordene una lista con **MergeSort**.



Problema 3 - Visualización y Comparación

a) Agregue contadores de iteraciones a las funciones de ordenamiento: ahora en lugar de no retornar nada deberán retornar la cantidad de iteraciones hicieron para ordenar la lista.

b) Cree una función reciba un entero y que dentro de ella genere una lista aleatoria del tamaño de este entero, luego la ordene con los dos algoritmos (la misma lista) y luego devuelva las iteraciones usadas por cada algoritmo en una lista de dos coordenadas (de la forma return [iteraciones_bubblesort, iteraciones_mergesort])

c) Finalmente cree una función que reciba una lista con enteros que representan los tamaños de arreglo a ordenar y que, llamando a la función calcule las iteraciones usadas por ambos algoritmos para todos los tamaños pedidos. Estas iteraciones se deberán guardar en listas separadas (por ejemplo: listaIteracionesBubbleSort, listaIteracionesMergeSort) para luego ser graficadas. Para esto puede usar la libreria *pyplot* de *matplotlib*. A continuación se presenta un ejemplo.

```
def f(x):
    return x**2

X = list(range(100))
Y = [f(x) for x in X]

plt.plot(X,Y,label = 'funcion de x')
plt.xlabel('Coordenada x')
plt.ylabel('Coordenada y')
plt.title('y en función de x')
plt.legend()
plt.show()
```

Este código genera el siguiente gráfico: