Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2692 - Laboratorio de Algoritmos y Estructuras II
Trimestre Enero-Marzo 2017

Implementación de algoritmos de ordenamientos de orden cuadrático

1. Descripción de la actividad

El objetivo del laboratorio es realizar la implementación de los algoritmos de ordenamiento *Insertionsort*, *Bubblesort* y *Selectionsort*. Se les proporcionará de un código base está compuesto por los siguientes archivos:

- `ordenamiento.py:` Librería que debe contener la implementación de los algoritmos de ordenamientos.
- `cliente_ordenamiento.py:` Cliente que ejecuta todos los algoritmos de ordenamiento que se encuentran en el módulo `ordenamiento.py`, y muestra una gráfica con los resultados obtenidos.
- `arrayT.py:` Estructura de datos arreglo a utilizar.

Se requiere que usted implemente los algoritmos de ordenamiento en el archivo `ordenamiento.py:`. La aplicación `cliente_ordenamiento.py` se ejecuta con la siguiente llamada:

```
>./cliente_ordenamiento.py [-h] [-g] [-t NT] [n_elems [n_elems ...]]
```

donde:

- **-h**: Muestra la ayuda para ejecutar la aplicación.
- **-g**: Muestra una gráfica con los resultados. Por defecto no se muestra la gráfica.
- **-t NT** : Ejecuta cada uno de los algoritmos NT veces, para cada arreglo de datos. Por defecto NT = 1.
- **[n_elems [n_elems ...]]** : Lista con los números de elementos de los arreglos a ordenar. Si no se indica ninguna lista, por defecto se ordenará un arreglo con 200 elementos.

Por ejemplo, una llamada válida en donde se ejecutaría la aplicación es:

```
>./cliente_ordenamiento.py -g 100 200 300 400 -t 3
```

En este caso se ejecutarían los algoritmos de ordenamiento sobre arreglos, con números enteros generados aleatoriamente, con 100, 200, 300 y 400 elementos, y cada algoritmo se efectuará 3 veces. El programa tomará como tiempo del algoritmo el promedio de los tres intentos. Al final de la ejecución de todos los algoritmos, se muestra una gráfica con la tendencia de los tiempos promedios obtenidos, para cada tamaño de arreglo. Internamente lo que ocurre dentro del programa es lo siguiente. Cuando se realiza una prueba se genera un arreglo de enteros con valores aleatorios, y en cada prueba los valores generados son diferentes. Luego ese arreglo de enteros lo recibe cada uno de los algoritmos de ordenamiento y una vez concluida la ejecución del algoritmo se procede a comprobar si los elementos están ordenados o no, y finalmente se muestra el tiempo de corrida del algoritmo. Esto se hace para cada uno de los intentos indicados en la entrada, para luego tomar como tiempo de la corrida el tiempo promedio de todos los intentos.

Otros ejemplos de llamadas válidas son:

```
>./cliente_ordenamiento.py -h
>./cliente_ordenamiento.py 100 200 300 400
>./cliente_ordenamiento.py
>./cliente_ordenamiento.py -t 2 50 100
```

2. Detalles de la implementación y de las pruebas

Los algoritmos deben ser implementados siguiendo los pseudocódigos presentados en el libro de Cormen et al [1], o los que se muestran en el libro de Kaldewaij [2].

Se desea estudiar el tiempo de ejecución de los algoritmos. Debe ejecutar con al menos 5 arreglos de diferentes tamaños, tal que estos sean representativos, es decir, tamaños con los cuales se pueda observar que el tiempo obtenido por los algoritmos se corresponde con lo esperado en la teoría. Para cada prueba, es decir, para cada número de elementos del arreglo, debe realizar al menos tres corridas.

3. Condiciones de entrega

El trabajo es por equipos de laboratorio. Debe entregar los códigos fuentes de sus programas y la gráfica con los resultados de los algoritmos de ordenamiento, en un archivo comprimido llamado `LabSemana2-X-Y.tar.gz` donde X y Y son los números de carné de los integrantes del grupo. La entrega se realizará por medio del aula virtual **antes** de la 4:30 pm del jueves 19 de Enero de 2017.

Referencias

[1] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. *Introduction to algorithms*, 3rd ed. MIT press, 2009.

[2] Kaldewaij, A. *Programming: the derivation of algorithms*. Prentice-Hall, Inc., 1990.

Guillermo Palma / gvpalma@usb.ve / Enero 2017