Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2692 - Laboratorio de Algoritmos y Estructuras de Datos II
Trimestre Enero-Marzo 2020

Algoritmos de Ordenamiento Parte III

1. Introducción

El objetivo de este laboratorio es el de agregar nuevos algoritmos de ordenamientos y extender la librería ORDENAMIENTO.PY. En específico se quiere agregar variantes de los algoritmos de ordenamiento *Heapsort* y *Quicksort*.

2. Algoritmos de ordenamiento

El módulo ORDENAMIENTO debe contener los siguientes algoritmos de ordenamiento, implementados siguiendo el pseudo código usado en las referencias:

- INSERTION-SORT; página 18 de [3].
- SELECTION-SORT; en las páginas 256-257 de [1].
- SHELL-SORT; página 290 de [1].
- BUBBLE-SORT; página 40 de [3].
- MERGESORT; página 34 de [3].
- MERGESORT-INSERTION; página 229 de [2]. Cuando el número de elementos a ordenar es menor o igual a 100, entonces se ordena el arreglo con Insertion-Sort.
- MERGESORT-ITERATIVO; página 183 de [4].
- HEAPSORT; página 159 de [3].
- QUICKSORT-ITERATIVO; página 179 de [4].
- QUICKSORT; página 171 de [3], con la modificación de que si el arreglo o subarreglo de entrada tiene un número de elementos a ordenar que es menor o igual a 100, entonces se ordena el mismo con Insertion-Sort.
- RANDOMIZED-QUICKSORT; página 179 de [3], con la modificación de que si el arreglo o subarreglo de entrada tiene un número de elementos a ordenar que es menor o igual a 100, entonces se ordena el mismo con Insertion-Sort.

Para los algoritmos QUICKSORT y RANDOMIZED-QUICKSORT, debe crear una versión modificada de INSERTION-SORT, llamada INSERTION-SORT-INDEX, que permita ordenar un arreglo o subarreglo, dados el índice inicial y final del arreglo. Es decir, la entrada del INSERTION-SORT-INDEX, es igual a la de QUICKSORT y RANDOMIZED-QUICKSORT.

3. Programa de pruebas

Se quiere que implemente un programa, llamado PRUEBA_ORD.PY para probar la librería ORDENAMIENTO.PY. El programa debe ejecutar varias pruebas sobre todos los algoritmos de ordenamiento de la librería. Las pruebas tienen diferentes tipos de secuencias que van a recibir los algoritmos de ordenamiento y se describen como sigue:

1. Secuencia de números reales en el intervalo $[0, 1]$ generados aleatoriamente.
2. Secuencia de números enteros en el intervalo $[0, N]$ generados aleatoriamente, donde N es el tamaño de la secuencia a ordenar.
3. Una secuencia de tipo 1, pero en donde todos los elementos están ordenados.
4. Una secuencia de tipo 2, pero en donde todos los elementos están ordenados.
5. Una secuencia de tipo 1, pero en donde todos los elementos están ordenados de forma inversa.
6. Una secuencia de tipo 2, pero en donde todos los elementos están ordenados de forma inversa.
7. Una secuencia que contiene ceros y unos generados aleatoriamente.

Para ordenar las secuencias de tipo 3, 4, 5 y 6, que van a ser las entradas de los algoritmos de ordenamiento, se puede usar los métodos *sort* y *sorted* de Python.

El usuario puede seleccionar la prueba a aplicar y el número de veces, o intentos, que se va aplicar esa prueba sobre los algoritmos de ordenamiento. También el usuario **puede indicar una lista con el tamaño de la secuencias de entrada** a ordenar. El resultado del programa **es el tiempo promedio de ejecución y la desviación estándar, de cada uno de los algoritmos** sobre las secuencias generadas de un tipo específico. Es importante notar que una vez escogida un tipo de secuencia, **en cada nuevo intento se debe generar una nueva secuencia, y esta misma nueva secuencia debe ser ordenada por todos los algoritmos de ordenamientos seleccionados**. De esta forma se puede comparar el tiempo de cada uno de los algoritmos de ordenamiento, sobre la misma secuencia de entrada. Se quiere que el programa de pruebas **pueda graficar los tiempos de ejecución de los algoritmos de ordenamiento, dada la lista de tamaños de secuencias de entrada**. Para graficar los resultados de los algoritmos de ordenamiento debe hacer uso de la librería GRAFICAR_PUNTOS.PY. Esta librería permite tener un en solo gráfico, las curvas del tiempo de ejecución de varios algoritmos de ordenamiento. De esta manera es posible ver gráficamente el desempeño de los algoritmos de ordenamiento, y determinar cual es más rápido para un tipo de entrada específica

La ejecución de PRUEBA_ORD.PY se hace por medio de la siguiente línea de comando:

```
>./prueba_ord.py [-i #num] [-t #num] [--all|--nlgn|--On2] [-g] tamaño_1 .. tamaño_N
```

La semántica de los parámetros de entrada es la siguiente:

- **-t** Prueba a realizar. Los valores posibles están en el intervalo $[1, 7]$. Cada valor corresponde a una de las pruebas descrita anteriormente.
- **-i**: Número de pruebas o intentos a realizar de la prueba seleccionada.

- **–all**: Indica que se van a ejecutar todos algoritmos del módulo ORDENAMIENTO.
- **–nlgn**: Indica que se van a ejecutar los algoritmos con tiempo de ejecución $O(n \log n)$, para el peor caso, del módulo ORDENAMIENTO.
- **–On2**: Indica que se van a ejecutar los algoritmos con tiempo de ejecución $O(n^2)$, para el peor caso, del módulo ORDENAMIENTO.
- **tamaño_1 .. tamaño_N** lista con los números de elementos, o tamaño, de la secuencias de entradas que van a ser ordenadas.
- **-g**: indica que van a ser graficados los resultados, de todas los diferentes tamaños de secuencias ejecutados. Esta opción solo es válida si hay dos o más tamaños de secuencia. En caso contrario, el programa aborta con mensaje de error.

Los parámetros **–all**, **–nlgn** y **–On2**, son mutuamente exclusivos, es decir, solo puede haber uno de ellos en la línea de comandos. En caso de haber más de uno de ellos, el programa debe abortar con un mensaje de error al usuario. Cuando se aplique la opción **-g**, primero el programa debe mostrar todos los resultados por la salida estándar y luego, se debe mostrar la gráfica resultante. Es obligatorio chequear que debe haber al menos un valor de tamaño de la secuencia. Todos los demás argumentos de la línea de comandos son opcionales. Para manejar los argumentos de entrada del programa, debe hacer uso del módulo argparse o del módulo getopt. Los valores por de defecto son:

- **-t**: 1, es decir, el tipo de secuencia de entrada, es la generada por el tipo 1.
- **-i**: 3, es decir, se ejecutará tres veces sobre los algoritmos de ordenamiento la prueba seleccionada.
- **–all**: falso, es decir, no se ejecutarán todos los algoritmos de ordenamiento
- **–nlgn**: cierto, es decir, se ejecutarán los algoritmos de ordenamiento con tiempo de ejecución $O(n \log n)$ para el peor caso.
- **–On2**: falso, es decir, no se ejecutarán los algoritmos de ordenamiento con tiempo de ejecución $O(n^2)$ para el peor caso.
- **-g**: false, es decir, el programa no mostrará las gráficas del tiempo de ejecución.

Las siguientes líneas de comando son ejemplos de llamadas válidas del programa.

```
>./prueba_ord.py 1000
```

El programa ordena una secuencia con 1000 elementos usa los valores de entrada por defecto.

```
>./prueba_ord.py  -i 6 --nlgn 1000 2000 3000
```

El programa ejecutará la prueba 1. En esta prueba se ejecutará con tres secuencias que contienen 1000, 2000 y 3000 elementos. Para cada tamaño de secuencia se generarán seis secuencias diferentes. Se ejecutarán todos los algoritmos de ordenamiento del módulo ORDENAMIENTO, con tiempo de ejecución $O(n \log n)$ para el peor caso, sobre las seis secuencias generadas, para cada uno de los tres tamaños de secuencia.

La salida del programa es tiempo **promedio y desviación estándar**, de cada uno de los algoritmos de ordenamiento ejecutados, **los cuales deben estar en una misma línea para su mejor lectura**.

4. Estudio experimental

Debe ejecutar todos los algoritmos de ordenamiento, con tiempo de ejecución $O(n \log n)$ para el peor caso, sobre todos los tipos de secuencia. Se ejecutarán secuencias de tamaño 10000, 20000, 30000, 40000 y 50000. Cada secuencia debe ser ejecutada tres veces. Es decir, el programa debe ser ejecutado con la siguiente línea de comando:

```
>./prueba_ord.py  -t X -i 3 --nlgn 10000 20000 30000 40000 50000
```

Donde X se sustituye por el tipo de prueba a realizar. El objetivo es poder observar claramente en una gráfica el crecimiento del tiempo de ejecución de los algoritmos, tal que, sea posible distinguir la tendencia de crecimiento y determinar cual de los algoritmos de ordenamiento, presenta el mejor desempeño. En específico, debe presentar un reporte que conteniendo lo siguiente:

1. Descripción de la plataforma utilizada para correr los experimentos, es decir, sistema de operación, modelo de CPU y memoria RAM del computador.
2. Siete tablas, una tabla por cada tipo de secuencia. Cada tabla debe contener el tiempo promedio todos los algoritmos de ordenamiento para los tamaños de secuencia usados.
3. La gráfica de tiempo de ejecución correspondiente para cada tabla de resultados.

El informe deberá estar en formato PDF.

5. Condiciones de entrega

La versión final del código del laboratorio y el informe deben estar contenidos en un archivo comprimido, con formato *tar.xz*, llamado *LabSem4_X.tar.xz*, donde X es el número de carné del estudiante. La entrega del archivo *LabSem4_X.tar.xz*, debe hacerse al profesor del laboratorio por email, antes de las 9:00 pm del día domingo 02 de febrero de 2020.

Referencias

[1] Aho, A., Hopcroft, J., and Ullman, J. *Estructuras de Datos y Algoritmos*. Addison-Wesley, 1998.

[2] Brassard, G., and Bratley, P. *Fundamentals of Algorithmics*. Prentice Hall, 1996.

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

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