Estructuras de Datos y Algoritmos - ELO-320

Primer semestre 2020 - Campus San Joaquín

Francisco Cabezas
francisco.cabezas@xkivertech.cl

---

1. Tarea 2

Contenidos principales

- Algoritmos InsertionSort y HeapSort

Objetivos

- Familiarizarse con el desarrollo modular e incremental de código C
- Implementar algoritmos de ordenamiento y evaluar éstos según el tiempo de ejecución y su complejidad

---

Importante: Las tareas son individuales. Cualquier acción que pueda beneficiar de forma injusta la calificación de su tarea está prohibida, incluyendo la presentación de cualquier componente que no es de su autoría, o la facilitación de esto para otros. Es aceptable discutir -*en líneas generales*- los métodos y resultados con sus compañeros, pero **se prohíbe compartir soluciones de código**. **Utilizar código de internet que no es de su autoría,** también es considerado plagio, **a menos que se indique la fuente. Presten atención a las instrucciones de entrega**, que pueden incluir políticas de nombre de archivos y aspectos que se considerarán en la evaluación.

Descripción del trabajo a realizar

Parte A: *Preparación de datos de entrada (15 pts.)*

Adjunto a la tarea, se le proporcionarán 3 archivos:

```
random_150.txt
random_1500.txt
random_10000.txt
```

Cada uno tiene una cantidad asociada de números generados aleatoreamente. El archivo `random_150.txt` tiene 150 números aleatorios, mientras que el archivo `random_1500.txt` tiene 1500 números aleatorios y finalmente `random_10000.txt` que tiene 10000 números aleatorios. Estos números están separados por un salto de línea (`"\n"`), por lo que cada línea representa en particular un número.

Su programa se debe ejecutar de tal forma que acepte el nombre del archivo como argumento, es decir:

```
./sortingComparer random_150.txt
```

Luego, debe leer el archivo y guardar cada número en un arreglo de **enteros** que permita guardar cada número.

Hint: Se hace énfasis en la palabra entero, puesto que cuando lee la línea de un archivo usted captura un string o arreglo de caractéres, los cuales debe castear a entero. Para ello debe usar la función `atoi()`, importando la librería `stdlib.h`. La función `atoi` se declara como:

```
int atoi(const char *string)
```

Ésta recibe como input el string y devuelve un número entero. Por ejemplo, si el string es `"120"`, devuelve un entero con el valor 120. Si el input es `"18Septiembre"`, devuelve un entero con valor 18.

Parte B: *Ordenamiento de datos (45 pts.)*

Su programa debe implementar los algoritmos `InsertionSort` y `HeapSort` que vimos en clases. Para cada uno, debe utilizar la siguiente estructura de archivos:

```
main.c
  |
  |__ insertionSort.c
  |__ insertionSort.h
  |
  |__ heapSort.c
  |__ heapSort.h
```

Para el caso de `HeapSort`, debe implementar al menos las siguientes funciones:

```
int *heapSort(int *array)

int *buildHeap(int *array)

void siftDown()

void siftUp()

void insert(int value)
```

Si necesita agregar más funciones, modificar los argumentos de las funciones listadas o modificar el tipo de retorno de la función, es libre de hacerlo.
Para el caso de `InsertionSort`:

```
int *insertionSort(int *array)
```

Tanto `heapSort()` como `insertionSort()` recibirán como argumento el array de números obtenidos en el paso previo, y deberán devolver el array ordenado.

Dadas las dos metodologías de ordenamiento, para poder ejecutar cada una, su programa deberá recibir un argumento adicional. Dado lo anterior, el comando para ejecutar su programa con `insertionSort()` debería ser:

```
./sortingComparer random_150.txt insertionsort
```

Para ejecutar `heapSort()` debería ser:

```
./sortingComparer random_150.txt heapsort
```

De acuerdo al tercer argumento, discrimina entre un algoritmo y otro. Después de ordenar el algoritmo debe imprimir en pantalla el array resultante. Para ello, debe implementar una función con la siguiente firma dentro de su `main()`:

```
void printArray(int *array)
```

Parte C: *Análisis de datos (40 puntos)*

Introducción de marcas de tiempo

Para poder realizar un análisis de cada algoritmo, introduciremos marcas de tiempo en determinados puntos de sus implementaciones con el fin de poder medir los tiempos de ejecución. Para ello, utilice la función `time()` provista por la libreria `time.h`. Un ejemplo de implementación:

```
#include <stdio.h>
#include <time.h>

int main(void) {
   printf("Timestamp: %lu\n", (unsigned long) time(NULL));
   return 0;
}
```

Lo anterior retorna el timestamp (en este ejemplo, en segundos):

```
Timestamp: 1595628847
```

La función `time()` devuelve un `timestamp` del tipo `EPOCH`. Éste cuenta la cantidad de segundos desde el 1 de Enero de 1970. El retorno de la función debería ser en segundos, aunque puede devolver milisegundos e incluso nanosegundos.
Se sugiere contar los dígitos del ejemplo de arriba (que está en segundos) y validar.

Dado lo anterior, usted deberá poner una marca de inicio y de término, para luego obtener la diferencia y poder tener así la cantidad de segundos que demora su algoritmo en implementar. Usted deberá definir dónde pone las marcas, y si desea poner más de una incluso. Este dato se necesitará para realizar el análisis de la parte C.

Para analizar los datos, usted deberá crear un informe con sus conclusiones. Para ello, deberá realizar varias mediciones:

- Medición en su computador local

Ejecute su programa un número de n veces definido por usted (debe ser un número prudente de mediciones para tener un resultado más fidedigno), **para cada archivo y para cada algoritmo**.
Para cada medición, anote los valores obtenidos a partir de las marcas de tiempo y calcule la diferencia en segundos para obtener el tiempo de ejecución. Por cada medición ejecute el mismo procedimiento.
Identifique claramente el sistema operativo utilizado y los parámetros de hardware necesarios como cantidad de Memoria Ram, CPU, porcentaje de utilización de CPU al momento de realizar el análisis, entre otros que usted considere relevantes para la medición.

- Medición en servidor remoto

Realice la misma cantidad de mediciones en 2 servidores remotos que se dispondrán en Amazon Web Services. Los servidores tienen la siguiente configuración:

```
- Servidor 1

    - Sistema Operativo: Ubuntu 18.04
    - CPU: 1 núcleo
    - Memoria RAM: 1 GB
    - Dirección IP (host): 34.220.245.41

- Servidor 2

    - Sistema Operativo: Ubuntu 18.04
    - CPU: 4 núcleos
    - Memoria RAM: 16 GB
    - Dirección IP (host): 18.237.176.205
```

Instrucciones de conexión

Para conectarse a estas máquinas, necesita utilizar el protocolo ssh (Ver https://es.wikipedia.org/wiki/Secure_Shell). El protocolo ssh permite generar un tunel de conexión segura entre cliente y servidor. Se ejecuta sobre el puerto 22, y requiere de una autenticación para concretar la conexión. En el caso de nuestros servidores, para ambos casos, las credenciales son:

```
Usuario: elo320
Clave: elo320
```

Si usted tiene sistema operativo Linux/MacOS, abra el terminal y ejecute el comando:

Servidor 1

```
ssh elo320@34.220.245.41
```

Servidor 2

```
ssh elo320@18.237.176.205
```

Cuando le pregunte por la clave, ingrese la clave expuesta más arriba (elo320).

Para el caso de Windows, puede instalar la herramienta PuTTY (Ver https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html), la cual permite conectar de forma fácil usando varios

protocolos. En el caso particular usaremos `ssh`. Para mayor información, revise el siguiente tutorial: https://mediatemple.net/community/products/dv/204404604/using-ssh-in-putty-(windows)

Para mover archivos hacia los servidores, utilize FileZilla (Descargar en https://filezilla-project.org/download.php?type=client). En la barra superior, debe ingresar las credenciales y el host (la dirección IP) del servidor. El programa es sencillo de ocupar, ya que es drag-and-drop. Para mayor información, revise los siguientes tutoriales:

- https://help.one.com/hc/es/articles/115005585709--Cómo-se-conecta-a-un-servidor-SFTP-con-FileZilla-
- https://youtu.be/PvMrP2iQ4fM

Asimismo que con su computador, realice las mediciones necesarias en cada servidor para recabar los datos necesarios.

Importante: el compilador gcc ya se encuentra instalado en los dos servidores. Basta con ejecutar el comando `gcc`:

```
gcc main.c -o sortingComparer
```

---

1. Una vez obtenga toda la información necesaria, realice una tabulación de los datos:
    - por algoritmo
    - por tamaño de array de entrada
    - por Hardware
2. Grafique los resultados considerando tiempo de ejecución versus tamaño de array de entrada, por cada algoritmo y para cada Hardware.
3. Dado los gráficos, ¿es posible identificar un patrón?, ¿cómo se comporta cada algoritmo para distintos tamaños de entrada?, ¿cómo influye el hardware en los resultados?, explíque.
4. ¿Es posible aplicar relación de recurrencia y/o teorema maestro para llegar a obtener la complejidad?, comente
5. Calcule el valor de la complejidad teórica considerando que n es el tamaño del input de entrada y compare con los datos que obtuvo, ¿Qué puede concluir?

Consideraciones y formato de entrega

- Utilice `/* Comentarios */` para documentar lo que se hace en cada etapa de su código. Es vital documentar cada método para entender qué hace, qué parametros recibe y cual es su salida. El código deberá estar *identado* y ordenado.
- Toda tarea debe ser correctamente compilada y ejecutada. Aquellas tareas que no compilen serán evaluadas cualitativamente y tendrán un descuento importante.
- Para agilizar la revisión, esta tarea se entregará en 2 partes: la primera parte considerará los puntos A y B, por tanto, se subirá un archivo comprimido (.zip, .rar, .tar.gz, etc.) que contenga su código, más un archivo README.txt a aula. Posteriormente, se subirá el informe respectivo al análisis de datos (punto C).

¿Cómo entregar?

Su entregable debe ser subido a la plataforma AULA. La primera parte de la tarea (puntos A y B) debe ser entregada **hasta las 23:59 del Domingo 2 de Agosto, 2020**. La segunda parte de la tarea (punto C, informe) se puede entregar **hasta las 23:59 del Miércoles 12 de Agosto, 2020**.