UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

ELO 320
Estructura de Datos y Algoritmos
Tarea #02
Julio 12, 2020

DEPARTAMENTO DE ELECTRONICA

Semestre 2020 - 1

Profesor (Casa Central) P1: Fernando Auat Cheein

Profesor (Casa Central) P2: Ioannis Vourkas (coordinador)

Contenidos principales

- Estructuras de datos: Lista enlazada
- Algoritmos: insertSort(), mergeSort(), y binarySearch()

Objetivos

- Familiarizarse con el desarrollo modular/incremental de código C
- Practicar con el desarrollo de algoritmos de ordenamiento y de búsqueda de datos y validar su requerimiento en tiempo de ejecución

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 Aα y Ωω: *Preparación de datos de entrada* (20 Pts)

Su programa debe **generar una base de datos** con 15.000 (o más) números enteros aleatorios con valores entre 0 y 1.000.000 y escribirlos separados por coma (o de otra manera, de su preferencia) en un archivo llamado "`datos.txt`".

Para simular diferentes tamaños del problema dado, tendrán que leer desde el archivo "`datos.txt`" diferentes cantidades de números (por ejemplo, primero 150, después 1500, y por último todos los 15000 números). A medida que se vayan leyendo los números del archivo, y para cada cantidad diferente, crearán una **lista doblemente enlazada** y luego repetirán las distintas tareas que a continuación se describen en las partes ***Bβ*** y ***Γγ***.

Los nodos de la lista deben ser del siguiente tipo:

```
typedef struct nd {
  int number;
  struct nd *previous;
  struct nd *next;
}Nodo;
```

Ayuda: El archivo "`datos.txt`" se tiene que generar solamente UNA vez. Una vez terminado todo lo que se pide hacer con la lista de 150 números, tienen que eliminar dicha lista, crear otra lista con 1500 números, y repetir los mismos pasos.

Parte Bβ: *Ordenamiento de datos* (40 Pts)

En su programa deben implementar los algoritmos **insertSort** y **heapSort** que vimos en clases. La interfaz completa para el manejo de ambos algoritmos debe hacerse en los archivos **`mySortAlg.h`** y `mySortAlg.c` (en clases se entregaron templates para trabajar). Como mínimo, será necesario implementar las funciones que se mencionan (en forma genérica) a continuación, las cuales pueden complementar con otras auxiliares si estiman necesario:

```
insertSort()      // ejecuta insertSort sobre la lista
heapSort()        // ejecuta heapSort sobre la lista
buildMaxHeap()    // convierte la lista en un max-heap
maxHeapify()      // mantiene la propiedad base de un max-heap
parent()          // retorna índice (o puntero) del nodo padre
leftChild()       // retorna índice (o puntero) del hijo izquierdo
rightChild()      // retorna índice (o puntero) del hijo derecho
```

Para los 3 distintos tamaños de entrada antes descritos, ¿puede validar la complejidad asintótica de los algoritmos que estudiamos teóricamente en clases? Para lograr esto, para cada tamaño de entrada deben ejecutar varias veces cada algoritmo y luego calcular el promedio del tiempo que cada uno tomó para ordenar los datos.

Ayuda: El promedio es necesario ya que cada ejecución en su computador puede tomar diferentes tiempos, debido a varias razones como, por ejemplo, el qué tan ocupado se encuentra su computador en un momento dado. Por eso, **tome una marca de tiempo antes y otra después de ejecutar las funciones de interés**. Luego almacena la diferencia de ellas en un arreglo. Elimina y crea exactamente la misma lista de nuevo, y repite. Al finalizar, calcule el promedio de los tiempos que almacenó. Puede usar el header time.h.

Parte Γγ: *Búsqueda de datos* (30 Pts)

Para esta última parte, en su programa debe implementar el algoritmo de búsqueda binaria **binarySearch**. Todas las funciones relacionadas con este algoritmo deben estar en los archivos `mySearchAlg.h` y `mySearchAlg.c`.

Con la lista ya ordenada desde la parte anterior, **realice como mínimo 100 búsquedas** de números ejecutando **binarySearch** en la lista ordenada. Igual que en la parte anterior, repita varias veces la búsqueda para cada uno de los 100 números y luego almacene el promedio del tiempo que cada búsqueda tomó. Para los 3 distintos tamaños de entrada antes descritos, ¿puede validar la complejidad asintótica del algoritmo que vimos teóricamente en clases?

Ayuda: Para asegurarse que el número que buscan se encuentra en la lista, para cada una de las 100 (o más) búsquedas, simplemente seleccione aleatoriamente un número de la misma lista. De lo contrario sus intentos podrían tener varios casos con números que no existen y el promedio del tiempo calculado se verá afectado.

Parte Δδ: *Resumen de resultados* (10 Pts)

En el archivo "`README.txt`", junto con sus datos personales, reporte todos los tiempos de ejecución medidos en cada uno de los casos vistos en las partes ***Bβ*** y ***Γγ***, junto con sus **conclusiones sobre los tiempos y la dificultad en implementar heapSort en una lista en lugar de un arreglo como vimos en clases**.

Consideraciones y Formato de entrega

- Todas las funciones que NO forman parte de la interfaz de los algoritmos de ordenamiento y de búsqueda, serán definidas en el archivo `main.c` en la parte inferior, después de la función `main()`.

- Utilice **/*comentarios*/** para documentar lo que se hace en cada etapa de su código. **Se debe además usar el estilo de documentación Doxygen** utilizado en TODOS los ejemplos de código que encontrarán en AULA.
- El código deberá estar perfectamente *indentado* **con espacios**, no tabuladores.
- Toda tarea debe ser correctamente **compilada** y ejecutada **en el servidor Aragorn (**tareas que **no compilan** se evaluarán cualitativamente y **tendrán como máximo nota 54)**.
- Para la entrega, suba **un archivo** comprimido (.zip, .rar, .tar.gz, etc.) que contenga su código (es decir, **SOLAMENTE** sus archivos **.c** y **.h**), más el archivo `datos.txt` que usó para el análisis de los tiempos de ejecución, y el archivo `README.txt` donde especificará sus datos personales (nombre, apellido, ROL), el reporte solicitado y, de ser necesario, explicará cualquier particularidad que pueda tener su programa, como por ejemplo cómo debe ser compilado y ejecutado.

¿Cómo ENTREGAR?

El entregable debe ser subido a la plataforma **AULA**, **hasta las 23:50 del sábado 01 de Agosto, 2020**.