Análisis de algoritmos
Escuela Politécnica Superior, UAM, 2024–2025

Conjunto de Prácticas no. 2

Fecha de entrega de la práctica

- Grupos del Miércoles: 19 de Noviembre.
- Grupos del Viernes: 21 de Noviembre.

Esta práctica trata de determinar experimentalmente los tiempos de ejecución de algoritmos que utilizan la metodología de *divide y vencerás*. Los algoritmos que se van a estudiar son *MergeSort y HeapSort*. Cada uno de los métodos se analizará sobre tablas de diferentes tamaños y se compararán los resultados obtenidos con el estudio teórico del algoritmo.

Algoritmo MergeSort

1. Implementad en `sorting.c` el algoritmo de ordenación correspondiente al método conocido como *MergeSort*. El prototipo de las funciones que se necesitan serán `int mergesort(int* tabla, int ip, int iu)` y la función de combinación `int merge(int* tabla, int ip, int iu, int imedio)`. Las dos rutinas devuelven `ERR` en caso de error o el número de operaciones básicas si no ha habido error, `tabla` es la tabla a ordenar, `ip` es el primer elemento de la tabla, `iu` es el último elemento de la tabla, `imedio` es el índice del punto medio de la tabla.

   Utilizando el programa exercise4.c de la práctica anterior, realizad la correspondiente sustitución del método de ordenación y comprobad que el algoritmo *MergeSort* ordena correctamente.

2. A continuación, modificando el programa exercise5.c de la práctica anterior para utilizar el algoritmo *MergeSort*, obtener la tabla correspondiente a la variación del tiempo promedio de ejecución, máximo, mínimo y promedio de operaciones básicas en función del tamaño de la permutación. Representad dichos valores y compararadlos con el resultado teórico del algoritmo.

Algoritmo HeapSort

3. Implementad en `sorting.c` la rutina `int heapsort(int* tabla, int ip, int iu)` para el método de ordenación *HeapSort*; esta rutina devuelve `ERR` en caso de error o el número de operaciones básicas en el caso de que la tabla se ordene correctamente, donde `tabla` es la tabla a ordenar, `ip` es el primer elemento de la tabla e `iu` es el último elemento de la tabla.

   Las rutinas `CrearHeap`, `OrdenarHeap` y `Heapify`, necesarias para realizar una ordenación eficiente, se deben declarar como `int CrearHeap(int* tabla, int n)`, `int OrdenarHeap(int* tabla, int n)` y `int Heapify(int *tabla, int n, int i)`, donde el argumento `n` es el tamaño de la tabla y el argumento `i` de `Heapify` es el elemento sobre el que se aplica la rutina. La rutinas devolverán `ERR` en caso de error o el número de operaciones básicas si no ha habido error.

   Modificad el programa exercise4.c de la práctica anterior para aplicarlo al método de ordenación *HeapSort* implementado en `heapsort` y comprobad que el algoritmo *HeapSort* generado ordena correctamente.

4. Modificad el programa exercise5.c de la práctica anterior para obtener la tabla de tiempo promedio de ejecución, máximo, mínimo y promedio de operaciones básicas del algoritmo *HeapSort* en función del tamaño de la permutación y representad dichos valores comparándolos con los resultados teóricos.

Comparación de ambos algoritmos

5. Comparad los tiempos de ejecución de las rutinas `MergeSort` y `HeapSort`. Para ello, comparad los tiempos medios de reloj y los casos peor mejor y medio de ambas rutinas.

Cuestiones sobre la práctica

1. Compara el rendimiento empírico de los algoritmos con el caso medio teórico en cada caso. Si las trazas de las gráficas del rendimiento son muy picudas razonad porqué ocurre esto.

2. ¿Cuáles son los casos mejor y peor para cada uno de los algoritmos? ¿Qué habrá que modificar en la práctica para calcular estrictamente cada uno de los casos (también el caso medio)?

3. ¿Cuál de los dos algoritmos estudiados es más eficiente empíricamente? Compara este resultado con la predicción teórica. ¿Cual(es) de los algoritmos es/son más eficientes desde el punto de vista de la gestión de memoria? Razona este resultado.

Material a entregar en cada uno de los apartados

Documentación: La documentación constará de los siguientes apartados:

1. **Introducción:** Consiste en una descripción técnica del trabajo que se va a realizar, qué objetivos se pretenden alcanzar, que datos de entrada requiere vuestro programa y que datos se obtienen de salida, así como cualquier tipo de comentario sobre la práctica.

2. **Código:** El código de la rutina según el apartado. Como código también va incluida la cabecera de la rutina.

3. **Resultados:** Descripción de los resultados obtenidos, gráficas comparativas de los resultados obtenidos con los teóricos y comentarios sobre los mismos.

4. **Cuestiones:** Respuestas a las cuestiones sobre la práctica.

Todos los ficheros necesarios para compilar la práctica y la documentación se guardarán en un único fichero comprimido, en formato zip, o tgz (tgz representa un fichero tar comprimido con gzip) que se entregará al profesor de forma conjunta con la memoria de la práctica.
La entrega de los códigos fuentes correspondientes a las prácticas de la asignatura AA se realizará por medio de la página web **https://moodle.uam.es/**.
Adicionalmente, las prácticas deberán ser guardadas en algun medio de almacenamiento (lápiz usb, CD o DVD, disco duro, disco virtual remoto, etc) por el alumno para el día del examen de prácticas en Enero.

Ojo: Se recalca la importancia de llevar un lápiz usb **ademas de otros medios de almacenamiento como discos usb, cd, disco virtual remoto, email a direccion propia, etc**, ya que no se garantiza que puedan montarse y accederse todos y cada uno de ellos durante el examen, lo cual supondría la calificación de suspenso en prácticas.