Análisis de Algoritmos 2021/2022

Práctica 2

Alberti Vicente y Miguel Ibáñez,  Grupo 126

| Código | Gráficas | Memoria | Total |
| --- | --- | --- | --- |
|  |  |  |  |

**1. Introducción.**

En esta segunda práctica vamos a seguir programando funciones en distintos ficheros, todos ellos en el lenguaje de programación C: permutations.c, sorting.c y times.c, en los cuales seguiremos trabajando con distintos algoritmos de ordenación, esta vez con MergeSort y QuickSort. En esta práctica también seguiremos interpretando las funciones con gráficas (según el tiempo de ejecución o las operaciones básicas…).

**2. Objetivos**

2.1 Apartado 1

En la función sorting.c se implementa el algoritmo de ordenación MergeSort: **int mergesort(int\* tabla, int ip, int iu),** y también la función **int merge(int\* tabla, int ip, int iu, int imedio)**. Las dos rutinas deben devolver ERR en caso de error, en caso de que no lo haya devuelve el número de operaciones básicas. En las funciones **int tabla** es la tabla a ordenar, **ip** es el primer elemento de la tabla, **iu** es el último elemento de la tabla y **imedio** es el índice del punto medio de la tabla. Se utiliza el programa exercise.4 para comprobar que el algoritmo funciona correctamente.

2.2 Apartado 2

En este apartado se debe modificar el programa exercise5.c para utilizar el algoritmo MergeSort, y así obtener la tabla correspondiente a la variación del tiempo promedio de ejecución, máximo, mínimo y promedio de operaciones básicas. Después se representan dichos valores y se comparan con el resultado teórico del algoritmo.

2.3 Apartado 3

En la función sorting.c se implementa la función **int quicksort(int\* tabla, int ip, int iu)** para el método de ordenación QuickSort, esta rutina devuelve ERR si hay error, o en caso de que no haya error devuelve el número de operaciones básicas si la tabla se ordena correctamente. En la función **int tabla** es la tabla a ordenar, **ip** es el primer elemento de la tabla e **iu** es el último elemento de la tabla. También se deben realizar las rutinas **partition** y **median** de apoyo a quicksort, se deben declarar como int **partition(int\* tabla, int ip, int iu,int \*pos)** y **int median(int \*tabla, int ip, int iu,int \*pos)**, estas devolverán ERR en caso de error o el número de operaciones básicas si no ha habido error. En esta primera implementación se usa como pivote el elemento situado en la primera posición de la tabla, por tanto, median debería devolver el valor 0 y devolver el valor ip. Modificar el programa exercise4.c para aplicarlo al método de ordenación QuickSort y comprobar que el algoritmo generado ordena correctamente.

2.4 Apartado 4

En este apartado se debe modificar el programa exercise5.c para obtener la tabla de tiempo promedio de ejecución, máximo, mínimo y promedio de operaciones básicas del algoritmo QuickSort, y representar dichos valores comparándolos con los resultados teóricos.

2.5 Apartado 5

Verificar si se produce sobrecarga (overhead) debido al número de llamadas a las funciones, para evitar que esto ocurra, se debe modificar la rutina quicksort de tal forma que se elimine la recursión de cola. Esta nueva función se llamará quicksort src (QuickSort sin recursión de cola). Comparar este nuevo algoritmo con el de la rutina quicksort original.

**3. Herramientas y metodología**

3.1 Apartado 1

Hemos utilizado los entornos de desarrollo Windows y Linux, como herramientas de programación hemos usado Visual Studio, Valgrind, y como repositorio Github. También se ha utilizado la presentación de teoría en moodle para apoyarse en el pseudocódigo de MergeSort.

3.2 Apartado 2

Hemos utilizado los entornos de desarrollo Windows y Linux, como herramientas de programación hemos usado Visual Studio, Valgrind y Gnuplot para la gráfica, y como repositorio Github.

3.3 Apartado 3

Hemos utilizado los entornos de desarrollo Windows y Linux, como herramientas de programación hemos usado Visual Studio, Valgrind, y como repositorio Github. También se ha utilizado la presentación de teoría en moodle para apoyarse en el pseudocódigo de QuickSort.

3.4 Apartado 4

Hemos utilizado los entornos de desarrollo Windows y Linux, como herramientas de programación hemos usado Visual Studio, Valgrind y Gnuplot para la gráfica, y como repositorio Github.

3.5 Apartado 5

Hemos utilizado los entornos de desarrollo Windows y Linux, como herramientas de programación hemos usado Visual Studio, Valgrind y Gnuplot para la gráfica, y como repositorio Github. En este apartado se mejora el algoritmo del apartado 3, eliminando la recursión impidiendo así que se produzca una sobrecarga, comprobamos las óptimas mejoras en los resultados.

**4. Código fuente**

Aquí ponéis el código fuente **exclusivamente de las rutinas que habéis desarrollado vosotros** en cada apartado.

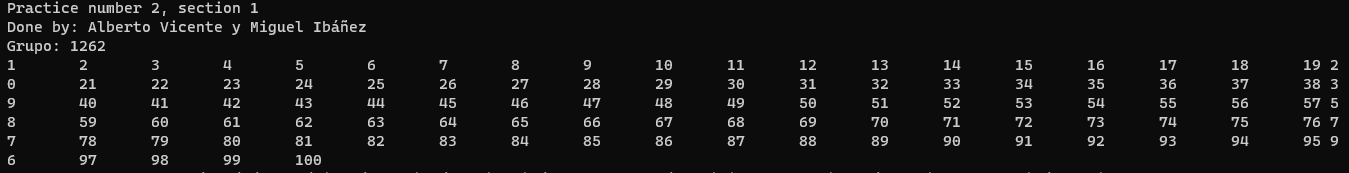
4.1 Apartado 1

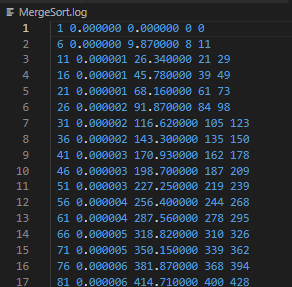
4.3 Apartado 3

4.5 Apartado 5

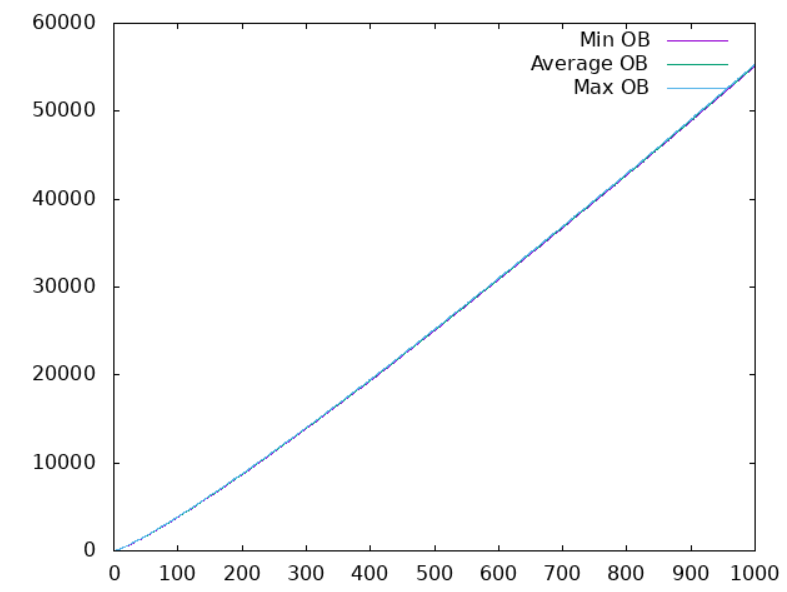
**5. Resultados, Gráficas**

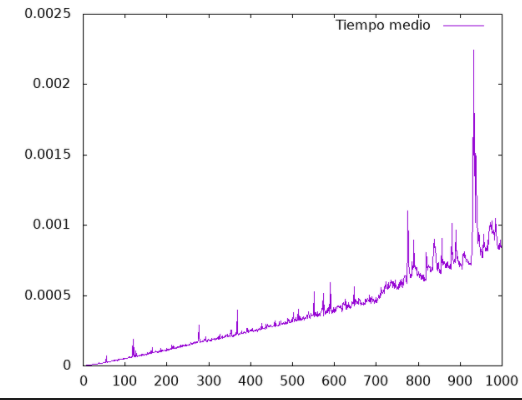
Aquí ponis los resultados obtenidos en cada apartado, incluyendo las posibles gráficas.

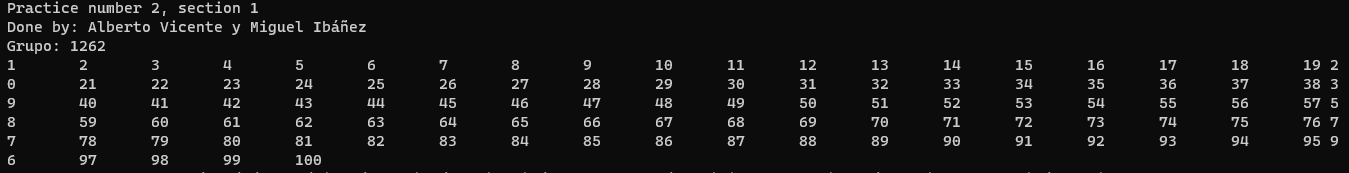
5.1 Apartado 1

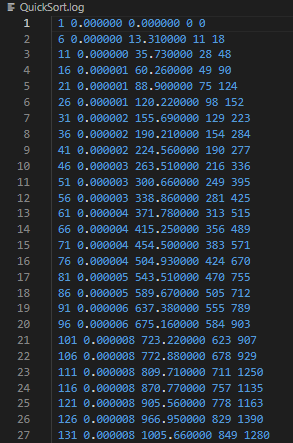
5.2 Apartado 2 

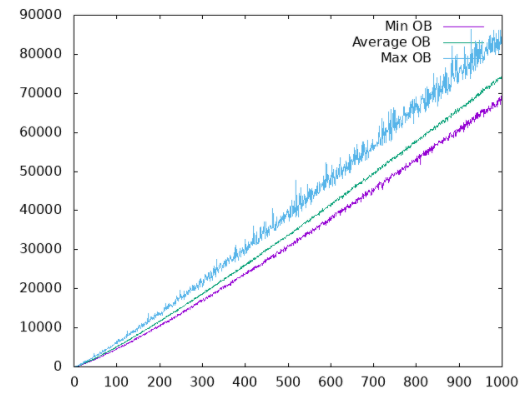
Gráfica comparando los tiempos mejor peor y medio en OBs para MergeSort, comentarios a la gráfica.

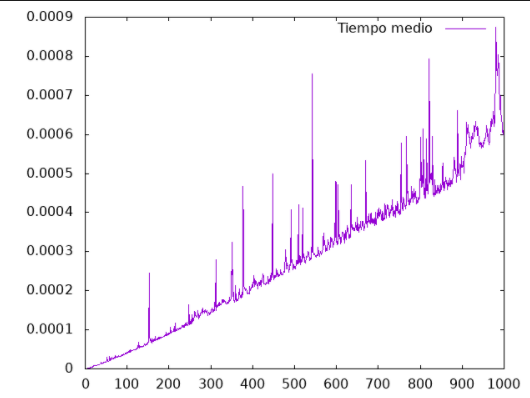


Gráfica con el tiempo medio de reloj para MergeSort, comentarios a la gráfica.

5.3 Apartado 3

5.4 Apartado 4





5.5 Apartado 5

Resultados del apartado 5.

Gráfica con el tiempo medio de reloj comparando las versiones de Quicksort con y sin recursión de cola.

**5. Respuesta a las preguntas teóricas.**

5.1 Pregunta 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 porque ocurre esto:

5.2 Pregunta 2

Razonad el resultado obtenido al comparar las versiones de quicksort con quicksort src tanto si se obtienen diferencias apreciables como si no:

5.3 Pregunta 3

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

5.4 Pregunta 4

¿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:

**6. Conclusiones finales.**

En esta segunda práctica seguimos analizando los tiempos de ejecución de los algoritmos, en este caso del MergeSort y del QuickSort, también se han desarrollado distintas funciones de apoyo para estos algoritmos.