Análisis de Algoritmos 2018/2019

Práctica 3

Daniel Brito y Manuel Suarez. Grupo 1201

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

**1. Introducción.**

En esta práctica estudiamos el desarrollo de algoritmos de búsqueda sobre diccionarios, también analizaremos su eficiencia, el tiempo de ejecución y las operaciones básicas que realizan

**2. Objetivos**

**2.1 Apartado 1**

En este apartado debemos crear un TAD diccionario y las siguientes funciones para poder trabajar con estos diccionarios: ini\_diccionario, inserta\_diccionario, inserción\_masiva\_diccionario, busca\_diccionario y libera\_diccionario. Además, debemos crear también las rutinas bbin y blin que corresponden a la búsqueda binaria y búsqueda lineal respectivamente.

Finalmente probaremos estas rutinas con el programa C ejercicio1.c.

**2.2 Apartado 2**

Ahora debemos implementar funciones para medir el rendimiento de las funciones que creamos en el apartado 1, añadiendo al archivo tiempo.c que hemos utilizado a lo largo de las prácticas la función tiempo\_medio\_busqueda, que creará un diccionario y una permutación de tamaño N insertará en el diccionario, reservará memoria para la tabla de n\_veces\*N, rellenará esta tabla con el generador de claves y finalmente mediremos el tiempo que tarda en buscar las n\_veces\*N claves almacenadas en la tabla anterior.

**3. Herramientas y metodología**

Para realizar esta práctica hemos empleado el editor de texto Atom en Ubuntu al considerarlo el más versátil y cómodo de entre los que disponemos en la EPS. Para compilar haremos uso de gcc y para comprobar que nuestro código no tuviera fugas de código empleamos Valgrind. Como herramienta de para la generación de gráficos empleamos GNUPlot, puesto que es el generador de gráficos más cómodo y que nos permite utilizar scripts para generarlos automáticamente.

**3.1 Apartado 1**

En el primer apartado desarrollamos el TAD diccionario y varias funciones para iniciar diccionarios, liberarlos, insertar tanto un elemento como varios, etc.

Veamos cómo hemos afrontado la creación de estas funciones. En primer lugar, tenemos la función de iniciar diccionario, ini\_diccionario, en la que simplemente creamos un nuevo diccionario de tamaño el tamaño de su TAD, además rellenaremos el TAD con el tamaño y el orden que recibimos como argumentos, el número de datos a 0 y reservamos memoria para una tabla con tamaño que recibimos como argumento. Finalmente devolvemos el TAD después de haberle asignado estos valores iniciales.

Ahora libera\_diccionario que simplemente comprueba la existencia del diccionario que recibe y su correspondiente tabla y los libera. En inserta\_diccionario comprobamos que el elemento que queremos introducir cabe en la tabla del diccionario que recibimos, en cuyo caso insertamos la clave en la última posición libre y comprobamos si se trata de un diccionario ordenado o no, si no lo es, saldremos de la función, pero si se trata de un diccionario ordenado antes de salir debemos ordenarlo, colocando la clave recién introducida entre las 2 que correspondan. Insercion\_masiva\_diccionario simplemente llama tantas veces como claves vayamos a introducir a la función inserta\_diccionario. Implementaremos también la función busca\_diccionario que recibe un diccionario, la clave a buscar además devuelve la posición por referencia, y un método de búsqueda, que puede ser búsqueda lineal o binaria.

Esta función únicamente hace una llamada a la función de búsqueda lineal o binaria, mediante punteros a función dependiendo de qué método recibe como argumento.

Ahora explicaremos las funciones de búsqueda. Empezamos con la búsqueda lineal, esta función consiste simplemente en hacer un bucle que recorra la tabla comparando cada elemento con la clave, si alguno coincide devolvemos su posición por referencia, si no, la clave no se encuentra en la tabla. Ahora

El algoritmo de búsqueda binaria es ligeramente más complejo que el anterior. Primero, si la tabla tiene una sola clave, comprobamos si se trata de la que estamos buscando y devolveremos, de nuevo por referencia, su posición o NO\_ENCONTRADO, macro que definimos en búsqueda.h. Ahora creamos un bucle while, del que no saldrá hasta que la subtabla que buscamos tenga 1 solo elemento, esta subtabla se creará dependiendo de si el elemento que se encuentra en medio es mayor o menor que la clave, si es mayor entonces la subtabla resultante será la que va de la primera clave a la clave en la posición justo antes del medio, si en cambio es menor la subtabla tendrá como primer elemento el elemento siguiente al del medio en la anterior tabla y llegará hasta el mismo. Si en algún momento el elemento del medio es igual a la clave, ya la habremos encontrado y solo faltara devolver su posición por referencia.

Finalmente queda hablar de la función generador\_claves\_uniforme que generará claves de 1 a un numero max que recibe como argumento, además recibe otro argumento, el número de claves, si el número de claves es igual al máximo aparecerá cada clave exactamente una vez. Simplemente crearemos un bucle for en el que en cada iteración introduciremos, en la tabla de claves, el resto de hacer i % max y sumarle 1, i siendo la variable que va de 0 al número de claves.

**3.2 Apartado 2**

Ahora crearemos una nueva función en tiempos.c para probar el tiempo de ejecución al usar diccionarios. genera\_tiempos\_busqueda, que recibe un método, un generador de claves, un fichero, el número mínimo y máximo de elementos, el incremento, y el número de veces que se repite la ejecución del programa que, a solicitud del enunciado, será siempre 1. Empezamos hallando el número de tiempos que tendremos que medir, esto se hace hallando la diferencia entre el número máximo y mínimo y dividiendo ese número entre el incremento, con este valor podemos reservar memoria para un puntero a la estructura TIEMPO. Ahora llamaremos a la función de tiempo\_medio\_busqueda y tras eso a guarda\_tabla\_tiempos para guardar los tiempos obtenidos con la función anterior.

Ahora tiempo\_medio\_busqueda que recibe un método, un generador de claves, un orden, un tamaño de diccionario, el número de veces, y una estructura puntero a TIEMPO donde guardar los tiempos obtenidos, esta función crea el diccionario y genera unas permutaciones de tamaño N que luego inserta masivamente en el recientemente creado diccionario, después crea una tabla y la rellena empleando el generador de claves, la tabla tendrá tamaño N\*n\_veces, después crea una variable de tiempo, y la iguala a clock(), a continuación realiza la búsqueda sin éxito de una clave. Asigna el valor de las operaciones básicas al máximo, mientras que el mínimo será 1 operación básica. Para hallar el medio hace el máximo entre el número de elementos de la tabla, en cuanto al tiempo creamos otra variable de tipo clock\_t y la inicializamos, y haciendo la resta de los dos tiempos y la dividimos tanto por (N\*n\_veces) como por CLOCKS\_PER\_SEC para hallar el tiempo real que tarda en realizar la búsqueda. Introducimos estos valores en la estructura ptiempo y liberamos el diccionario, permutación, tabla y pos donde.

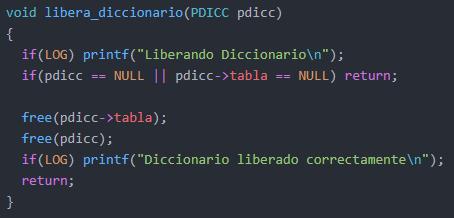
**4. Código fuente**

**4.1 Apartado 1**

4.1.1 ini\_diccionario



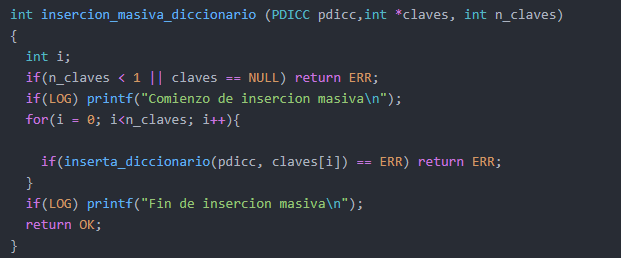
4.1.2 libera\_diccionario



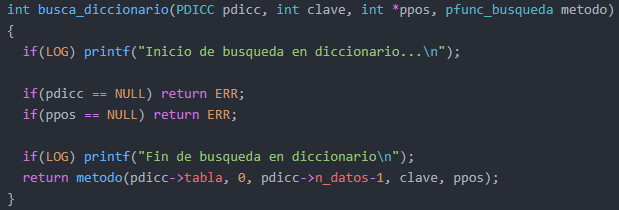
4.1.3 inserta\_diccionario



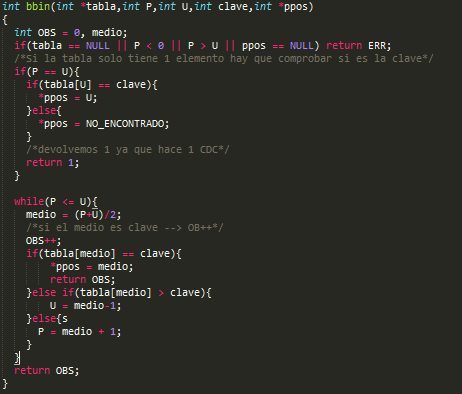
4.1.4 insercion\_masiva\_diccionario



4.1.5 busca\_diccionario

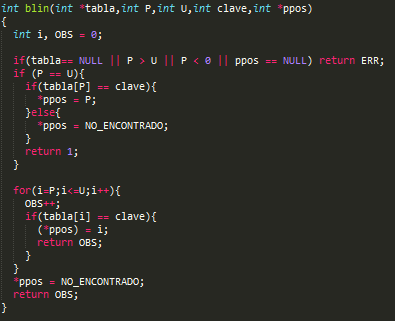


4.1.6 bbin

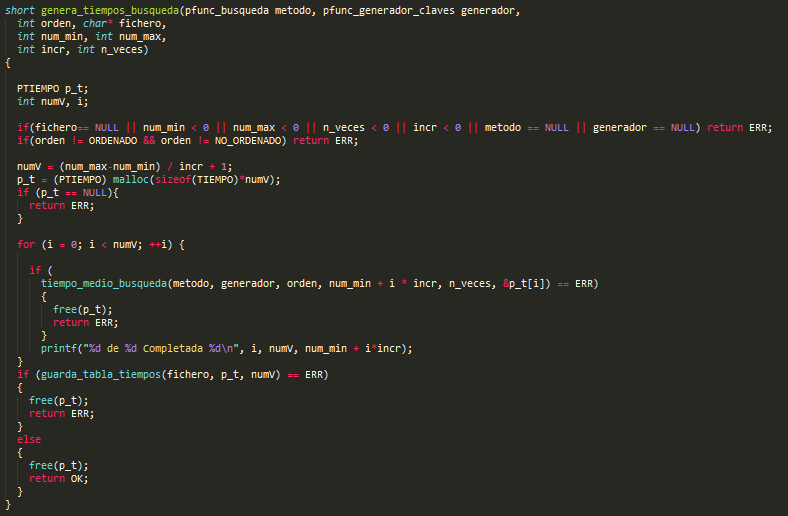


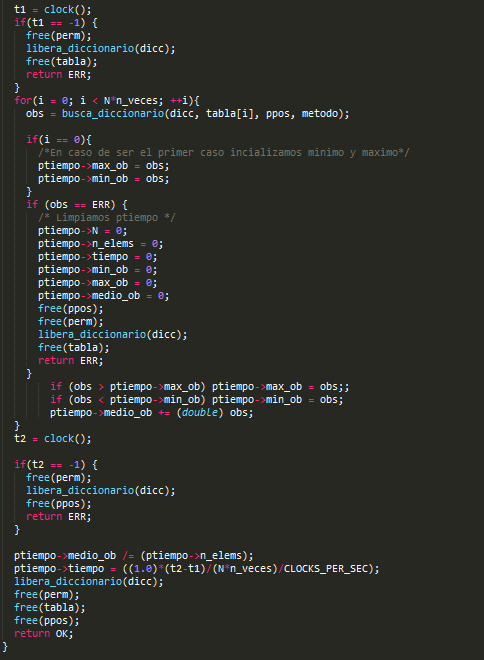
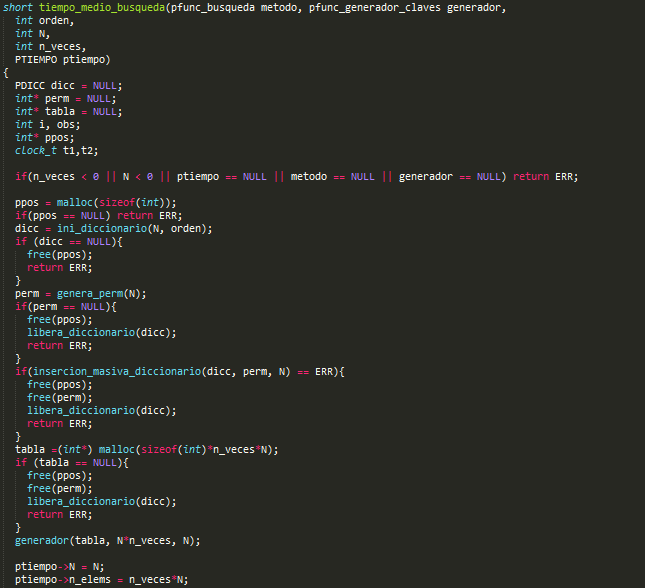
Hemos tomado el criterio de que solo se cuente una OBS cada vez que se realiza una cláusula if+else if+else. En caso de que no se debiera hacer así, sería tan fácil como añadir un OBS++ en la línea de después del else if y de después del else.

4.1.7 blin



**4.2 Apartado 2**

4.2.1 genera\_tiempos\_busqueda 

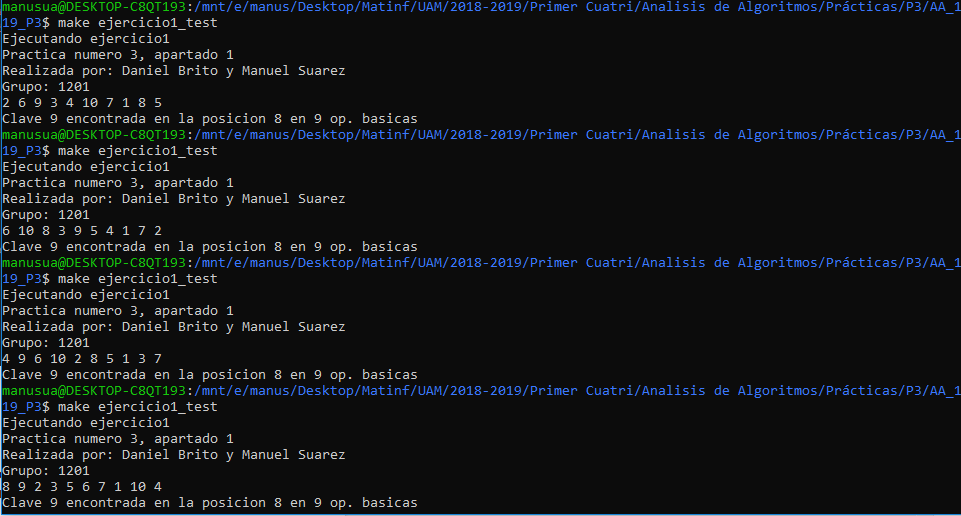
4.2.2 tiempo\_medio\_busqueda

**5. Resultados, Gráficas**

5.1 Resultados Apartado 1

Podemos comprobar, al ejecutar ejercicio1.c tanto con blin y bbin (con una tabla de ordenada), que el programa funciona correctamente. A continuación, se muestran algunos ejemplos de ello:

**Búsqueda Lineal:**



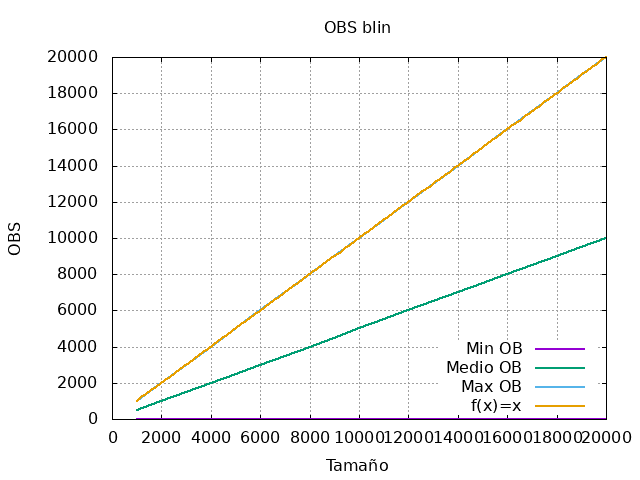
Podemos ver que, efectivamente, al tratarse del 9 y al estar la tabla ordenada (nos referimos a la tabla del diccionario, no a la tabla que se muestra por pantalla), se encuentra en la posición 8 del vector (el noveno elemento pues el primero es el 0) y realiza, efectivamente, 9 operaciones básicas que se corresponden con las 9 CDC que se realizan.

**Búsqueda Binaria**

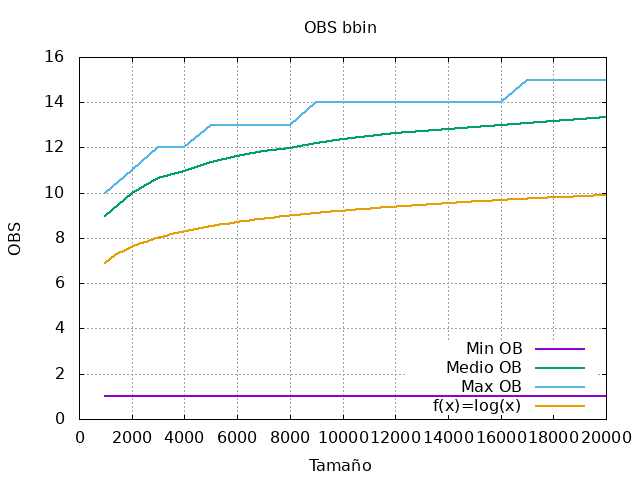
****

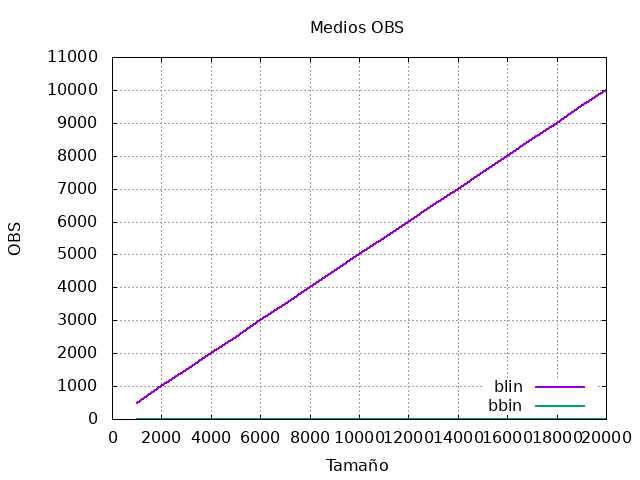
Nuevamente los resultados encajan con lo esperado pues estamos, de nuevo, ante la tabla ordenada que se almacena en la estructura diccionario. Por lo tanto, este tiene una tabla de los números del 1 al 10. La primera CDC se realizaría con el elemento que está en la posición 4, el 5. La segunda CDC se realizaría con el elemento de la posición 7, el 8. La tercera con el de la posición 8, el 9.

5.2 Resultados del apartado 2.

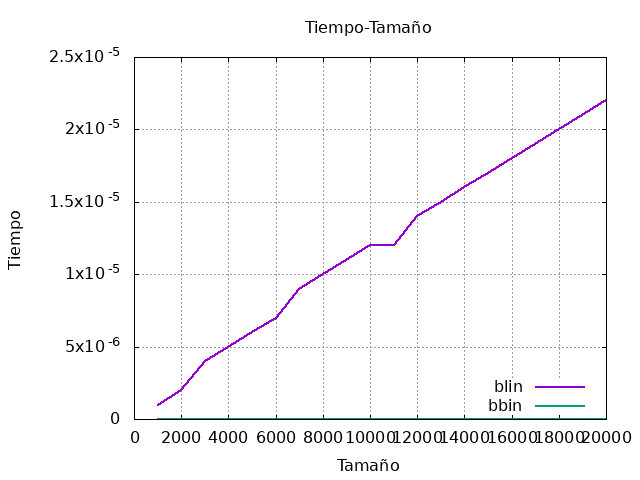
**Gráfica con el número máximo, mínimo y medio de OBs de la búsqueda lineal.**

Podemos observar que, efectivamente el valor máximo de OBS coincide con un ajuste lineal y que, aunque no esté representado, el número medio de OBS corresponde con un ajuste lineal de f(x)=x/2.

**Gráfica con el número máximo, mínimo y medio de OBs de la búsqueda binaria.**

**Gráfica comparando el número promedio de OBs entre la búsqueda lineal y la búsqueda binaria, comentarios a la gráfica.**

Podemos observar que, efectivamente, la búsqueda binaria tiene un promedio de OBS muchísimo inferior que el de la búsqueda lineal. No en vano sabemos que la búsqueda binaria es de los sistemas más eficientes a la hora de buscar un elemento dentro de una tabla ordenada.

**Gráfica comparando el tiempo promedio de reloj entre la búsqueda lineal y la búsqueda binaria, comentarios a la gráfica**.

El tiempo en ambos casos es bastante pequeño, sin embargo, podemos observar que los valores obtenidos en bbin son mucho menores (de hecho muchos de ellos eran de 0), pues se trata de un algoritmo tan sumamente eficiente que el tiempo que tarda en realizar las operaciones supera con creces a la precisión ofrecida por el formato *double* que tiene el tipo pTiempo empleado.

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

Aquí respondemos a las preguntas teóricas que se nos han planteado en la práctica.

**5.1 ¿Cuál es la operación básica de bbin y blin?**

La operación básica de la búsqueda binaria y la búsqueda lineal es idéntica a la del resto de algoritmos que hemos observado, la comparación de claves

**5.2 Dar tiempos de ejecución en función del tamaño de entrada n para el caso peor WSS(n) y el caso mejor BSS(n) de bbin y blin. Utilizar la notación asintótica (O, Θ, o, Ω,etc) siempre que se pueda.**

Empezamos con la búsqueda lineal, su caso peor es simple de observar, es aquel en el que la clave este en la última posición o no este en la tabla. El caso mejor es que este en la primera posición que comparemos. De aquí obtenemos que Wblin(N) = N y Bblin(N) = 1.

Los costes de ejecución de la búsqueda binaria son los siguientes, el caso peor se da cuando el elemento que buscamos no está en la tabla, o si ese elemento es una hoja en el árbol de decisión equivalente a la tabla. El caso mejor es cuando el elemento se encuentra en la primera posición que comparamos, en el bbin se trata del elemento en mitad de la tabla. Por tanto tenemos: Wbbin(N) = SUELO(log2N) y Bbbin(N) = 1.

**5.3 ¿Cuál es el orden de ejecución medio de blin y bbin en función del tamaño de elementos en el diccionario?**

El caso medio de blin es aquel en el que el elemento que buscamos esté en el medio de la tabla y por tanto la gráfica de ejecución se corresponde con el ajuste lineal de . El caso medio para la búsqueda binaria, según lo aprendido en clase de teoría es, log2N - 1 + , si lo medimos experimentalmente obtenemos que es únicamente log2N - 1 aproximadamente, por lo que la diferencia de ambos es de , pero este término decrece a medida que crece N y cuando N llega a valores grandes este término será despreciable.

Con todo esto tenemos que, para blin 

El tiempo de ejecución medio de cualquiera algoritmo es directamente proporcional al número de OBs que realiza, por tanto sabiendo sus OBs para el caso medio sabremos, con independencia de compilador y computadora, el tiempo de ejecución.

5.4 **Justifica lo más formalmente que puedas la corrección (o dicho de otra manera, el por qué buscan bien) los algoritmos bbin y bbin.**

Empezamos con el algoritmo más sencillo, blin, este algoritmo recorre una tabla elemento a elemento comparando cada uno de ellos con la clave que se busca, al recorrer todos los elementos y compararlos uno a uno, si la clave se encuentra en la tabla blin la encontrará, y si, al llegar al final de la tabla no ha devuelto nada, significa que no encontró la clave.

Ahora el algoritmo bbin. Antes de explicar su corrección hay que avisar que la tabla que recibe debe estar ordenada. En resumen, este algoritmo elimina la parte de la tabla en la que la clave no puede estar, hallando el elemento del medio y comparando con la clave para ver si la clave es mayor que el elemento en medio con lo que descarta la parte que incluye a todos los elementos menores que el medio. Repite este procedimiento partiendo las tablas en subtablas por la mitad hasta que llega a una subtabla de un solo elemento, si se trata de la clave, ya la hemos encontrado, si no, significa que no se encuentra la clave en la tabla. Esto es posible de implementar de esta forma ya que la tabla está ordenada, ya sea de menor a mayor o al revés.

**6. Conclusiones finales.**

En esta práctica nos apartamos de la línea seguida por la anteriores, dejamos atrás la ordenación de tablas, para dedicarnos a la búsqueda de claves en ellas mediante el uso de diccionarios.

Durante la realización de la práctica pudimos observar las diferencias entre la búsqueda lineal y binaria dentro de los diccionarios y las ventajas que cada uno tiene, la búsqueda lineal es menos eficiente que la binaria pero no requiere que la tabla está ordenada, mientras que la búsqueda binaria si, por lo que si una tabla estuviera desordenada, al coste de ejecución de la búsqueda binaria habría que añadirle el de ordenar la tabla, que dependiendo de la implementación es N2 o Nlog(N), aprendiendo así las diferencias que existen entre dos algoritmos de búsqueda.

Además, de nuevo afianzamos conceptos sobre la medición de los tiempos de ejecución, al tener que crear funciones nuevas que midieran el tiempo de ejecución de las