Análisis de Algoritmos 2017/2018

Práctica 3

Victoria Pelayo e Ignacio Rabuñal,  1273.

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

**1. Introducción.**

El objetivo de esta última práctica es estudiar diferentes algoritmos de búsqueda y su eficiencia sobre un diccionario.

**2. Objetivos**

2.1 Apartado 1

En este apartado debemos implementar el TAD diccionario con las funciones requeridas y las tres rutinas de búsqueda que analizaremos posteriormente.

2.2 Apartado 2

En este apartado debemos construir las funciones que nos permiten generar y medir los tiempos de búsqueda. A continuación, los analizaremos utilizando los diferentes generadores de clave dados.

**3. Herramientas y metodología**

En todos los apartados hemos utilizado el entorno Linux, el editor Atom para crear las funciones necesarias o modificar archivos anteriores. A la hora de comprobar nuestros ficheros, hemos utilizado gcc para compilar los archivos, Valgrind para asegurar una correcta gestión de memoria y el programa Gnuplot para representar los resultados obtenidos.

3.1 Apartado 1

Para implementar las funciones del TAD diccionario nos hemos valido de los prototipos de las mismas y de la información aportada por la estructura, con ayuda de los fragmentos de pseudocódigo dados.

Por su parte, los algoritmos BBIN y BLIN han sido implementados con ayuda de los pseudocódigos que aparecen en las transparencias de la parte teórica de la asignatura y para construir BLIN\_AUTO modificamos el código de BLIN sin dificultades.

3.2 Apartado 2

Implementamos las funciones para las medidas de rendimiento y procedemos a generar y medir los tiempos para comparar nuestros algoritmos con los supuestos teóricos.

En este apartado para medir los tiempos sy OBs y posteriormente representar las gráficas necesarias hemos utilizado las función guarda\_tabla\_tiempo de la anterior práctica.

**4. Código fuente**

4.1 Apartado 1

PDICC ini\_diccionario (int tamanio, char orden){

PDICC pdicc = NULL;

pdicc = (PDICC)malloc(sizeof(pdicc[0]));

if(pdicc == NULL)

return NULL;

pdicc->tabla = (int\*)malloc(sizeof(pdicc->tabla[0]) \*tamanio);

if(pdicc->tabla == NULL)

return NULL;

pdicc->tamanio = tamanio;

pdicc->orden = orden;

pdicc->n\_datos = 0;

return pdicc;

}

void libera\_diccionario(PDICC pdicc){

if(pdicc == NULL)

return;

if(pdicc->tabla == NULL){

free(pdicc);

return;

}

free(pdicc->tabla);

free(pdicc);

}

int inserta\_diccionario(PDICC pdicc, int clave){

int ob = 0;

int i = 0;

i = 0;

pdicc->tabla[pdicc->n\_datos] = clave;

pdicc->n\_datos ++;

if(pdicc->orden == 1){

i = pdicc->n\_datos - 1;

if(i > 0){

while (pdicc->tabla[i] < pdicc->tabla[i - 1]){

swap(&pdicc->tabla[i], &pdicc->tabla[i -1]);

i--;

if(i == 0)

break;

}

}

}

return ob;

}

int insercion\_masiva\_diccionario (PDICC pdicc,int \*claves, int n\_claves){

int i;

int ob = 0;

for(i = 0; i < n\_claves; i++){

ob+=inserta\_diccionario(pdicc, claves[i]);

}

return ob;

}

int busca\_diccionario(PDICC pdicc, int clave, int \*ppos, pfunc\_busqueda metodo){

if(pdicc == NULL){

return NO\_ENCONTRADO;

}

return metodo(pdicc->tabla, 0, pdicc->n\_datos, clave, ppos);

}

int bbin(int \*tabla,int P,int U,int clave,int \*ppos){

int ob = 0;

int m = 0;

while (P <= U){

m = (P + U)/2;

ob++;

if(clave == tabla[m]){

\*ppos = m;

return ob;

}

else if (clave > tabla[m]){

P = m + 1;

}

else{

U = m - 1;

}

}

return NO\_ENCONTRADO;

}

int blin(int \*tabla,int P,int U,int clave,int \*ppos){

int i = 0;

int ob = 0;

while (P < U){

ob++;

if(tabla[i] == clave){

\*ppos = i;

return ob;

}

i++;

}

\*ppos = NO\_ENCONTRADO;

return NO\_ENCONTRADO;

}

int blin\_auto(int \*tabla,int P,int U,int clave,int \*ppos){

int ob = 0;

int i = 0;

while (P < U){

ob++;

if(tabla[i] == clave){

if(i > 0){

\*ppos = i - 1;

swap(&tabla[i], &tabla[i] - 1);

return ob;

}

\*ppos = i;

return ob;

}

i++;

}

\*ppos = NO\_ENCONTRADO;

return NO\_ENCONTRADO;

}

4.2 Apartado 2

short genera\_tiempos\_busqueda(

pfunc\_busqueda metodo, pfunc\_generador\_claves generador, int orden, char\* fichero, int num\_min, int num\_max, int incr, int n\_veces){

int n = ((num\_max - num\_min) / incr) + 1;

int j, tamanio;

PTIEMPO tiempos = (PTIEMPO)malloc(n \* sizeof(tiempos[0]));

if(tiempos == NULL)

return ERR;

for (j = 0,tamanio = num\_min; tamanio <= num\_max; j++, tamanio += incr){

short codigo = tiempo\_medio\_busqueda(metodo, generador, orden, tamanio, n\_veces, &tiempos[j]);

if (codigo == ERR){

free(tiempos);

return ERR;

}

}

if (guarda\_tabla\_tiempos(fichero, tiempos, n) == ERR){

free(tiempos);

return ERR;

}

free(tiempos);

return OK;

}

short tiempo\_medio\_busqueda(pfunc\_busqueda metodo, pfunc\_generador\_claves generador, char orden, int N, int n\_veces, PTIEMPO ptiempo){

clock\_t ini,fin;

int i, ppos;

PDICC diccionario;

int \*perm, \*claves;

diccionario = ini\_diccionario(N, orden);

if(diccionario == NULL)

return ERR;

perm = (int\*)genera\_perm(N);

if(perm == NULL){

libera\_diccionario(diccionario);

return ERR;

}

claves = (int\*)malloc(n\_veces\*N\*sizeof(claves[0]));

if(claves == NULL){

free(perm);

libera\_diccionario(diccionario);

return ERR;

}

insercion\_masiva\_diccionario(diccionario, perm, N);

generador(claves, n\_veces\*N, N);

ptiempo->N = N;

ptiempo->n\_elems = n\_veces;

ptiempo->min\_ob = INT\_MAX;

ptiempo->max\_ob = 0;

ptiempo->medio\_ob = 0;

ini = clock( );

for(i = 0; i < n\_veces\*N; i++){

int ob = busca\_diccionario(diccionario, claves[i], &ppos ,metodo);

if(ob == ERR){

libera\_diccionario(diccionario);

free(claves);

free(perm);

return ERR;

}

if (ob < ptiempo->min\_ob)

ptiempo->min\_ob = ob;

if (ob > ptiempo->max\_ob)

ptiempo->max\_ob = ob;

ptiempo->medio\_ob += ob;

}

fin = clock();

ptiempo->medio\_ob /= (n\_veces\*N);

ptiempo->tiempo = (double)(fin - ini)/ (n\_veces\*N) / CLOCKS\_PER\_SEC;

free(claves);

free(perm);

libera\_diccionario(diccionario);

return OK;

}

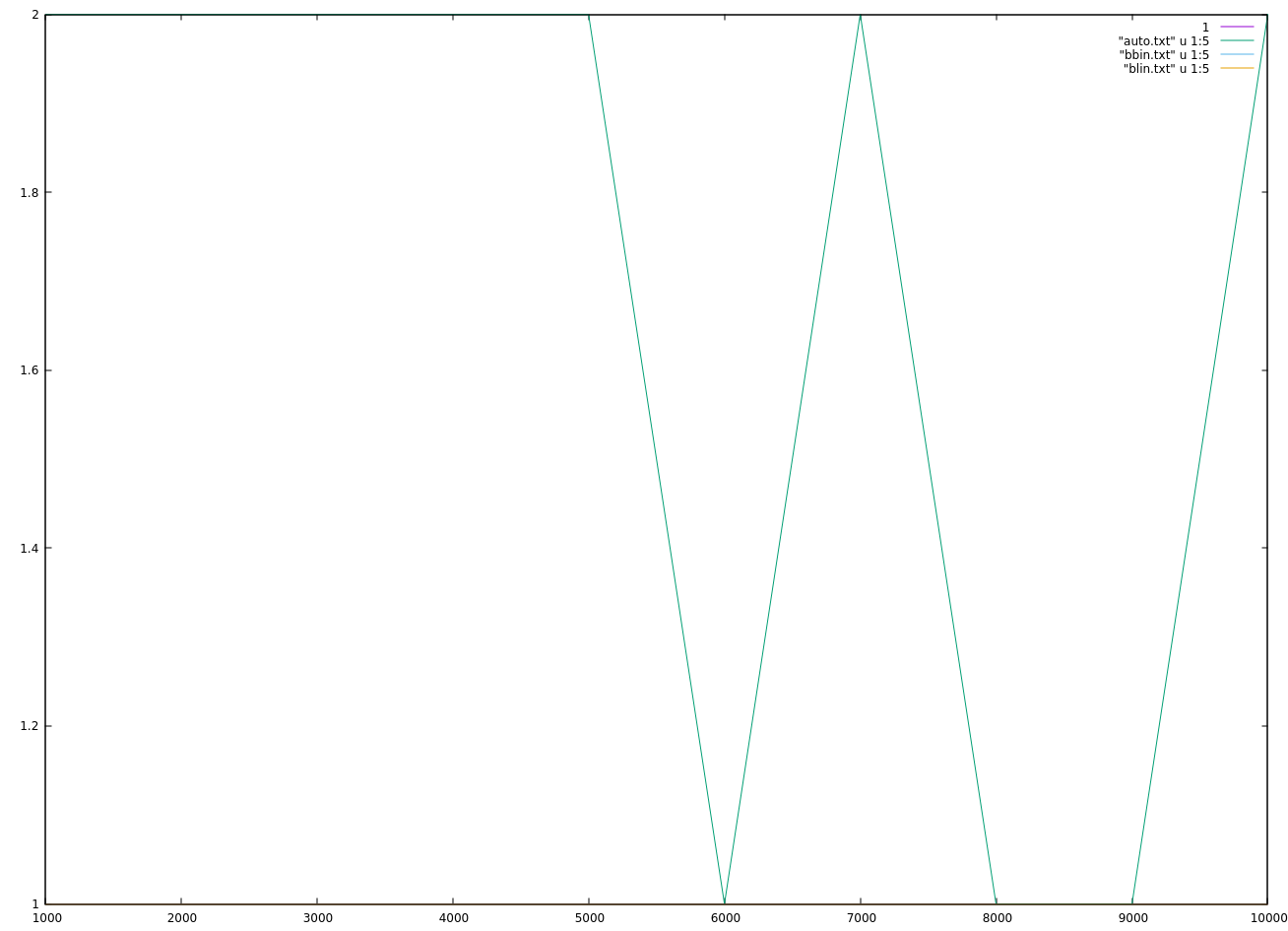
**5. Resultados, Gráficas**

5.1 Apartado 1

En estas gráficas se toma n\_veces = 1 y como generador el no potencial. Estudiaremos los costes de las búsquedas.

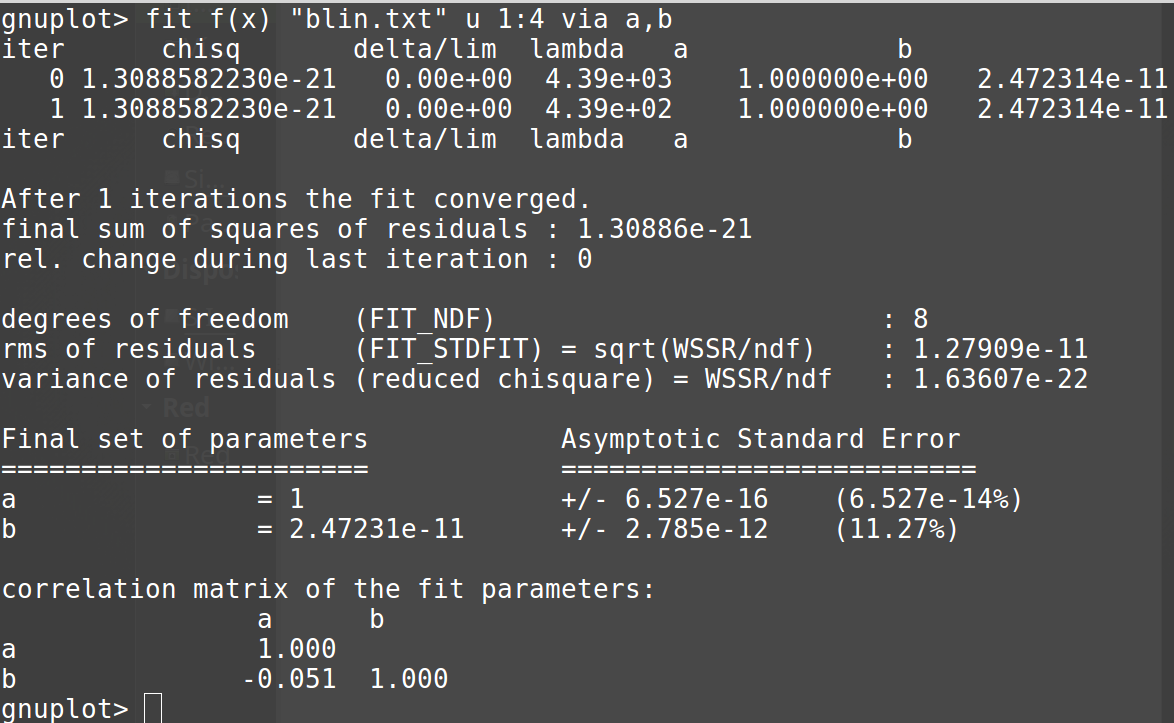
En este apartado de la práctica hemos aproximado las gráficas de las distintas búsquedas a funciones.

En todas su caso mejor es O(1), por ello hemos representado la función f(x) = 1 y el número mínimo de OBs de las tres búsquedas.



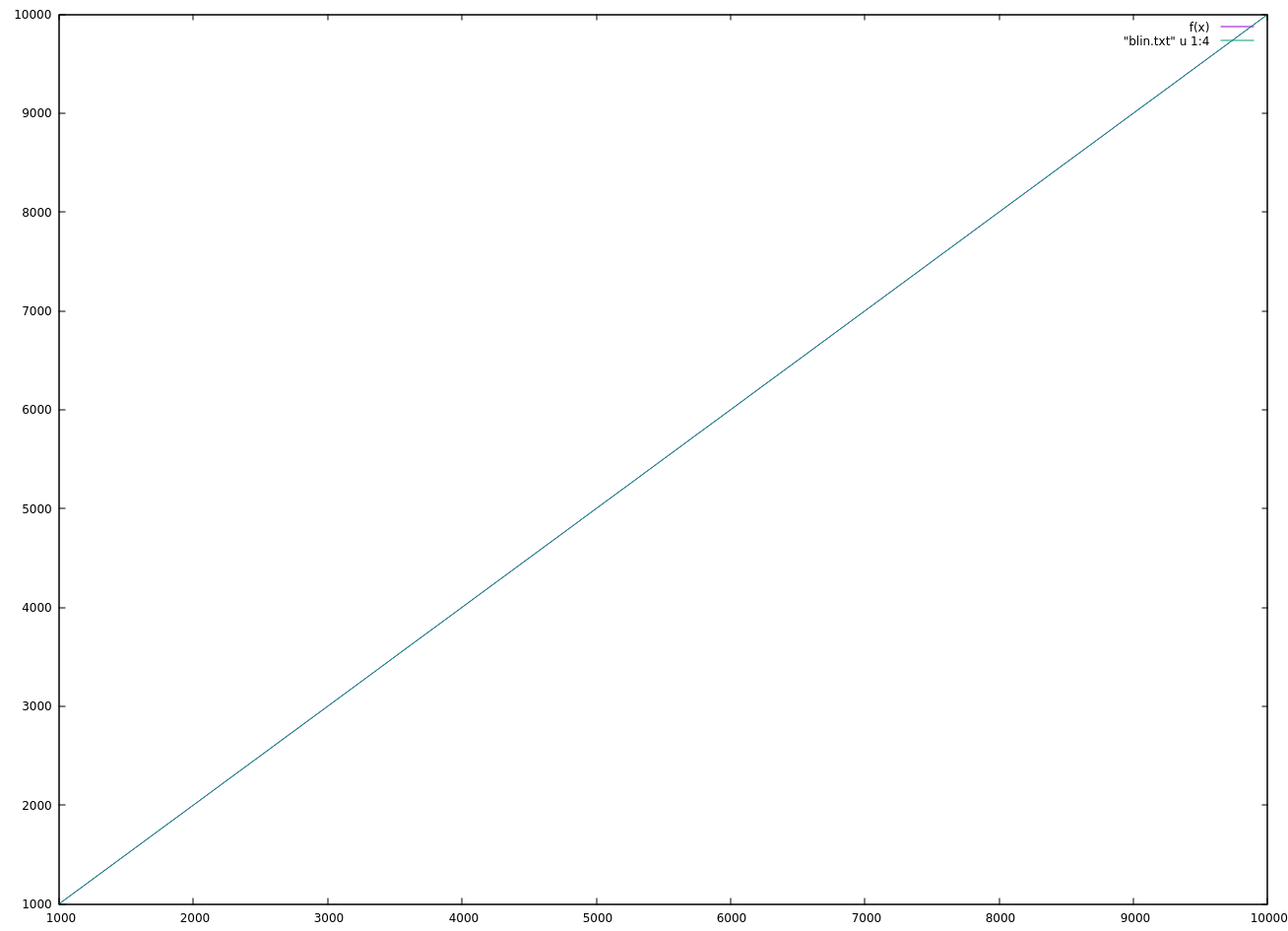
Como vemos todas son iguales que la de 1, es decir, su caso mejor es 1. Los picos son de la auto organizada, pero aunque en la gráfica se aprecie una gran diferencia es debido a que el rango del eje y es de 1 a 2. Los tres casos mejores comprobamos experimentalmente que se aproximan a 1.

Se puede concluir que experimentalmente se ha obtenido Bblin = Bblin\_auto = Bbbin = O(1), que coincide con lo esperado.

Aproximación de f(x) = a\*x + b con el número máximo de OBs de la búsqueda lineal. 

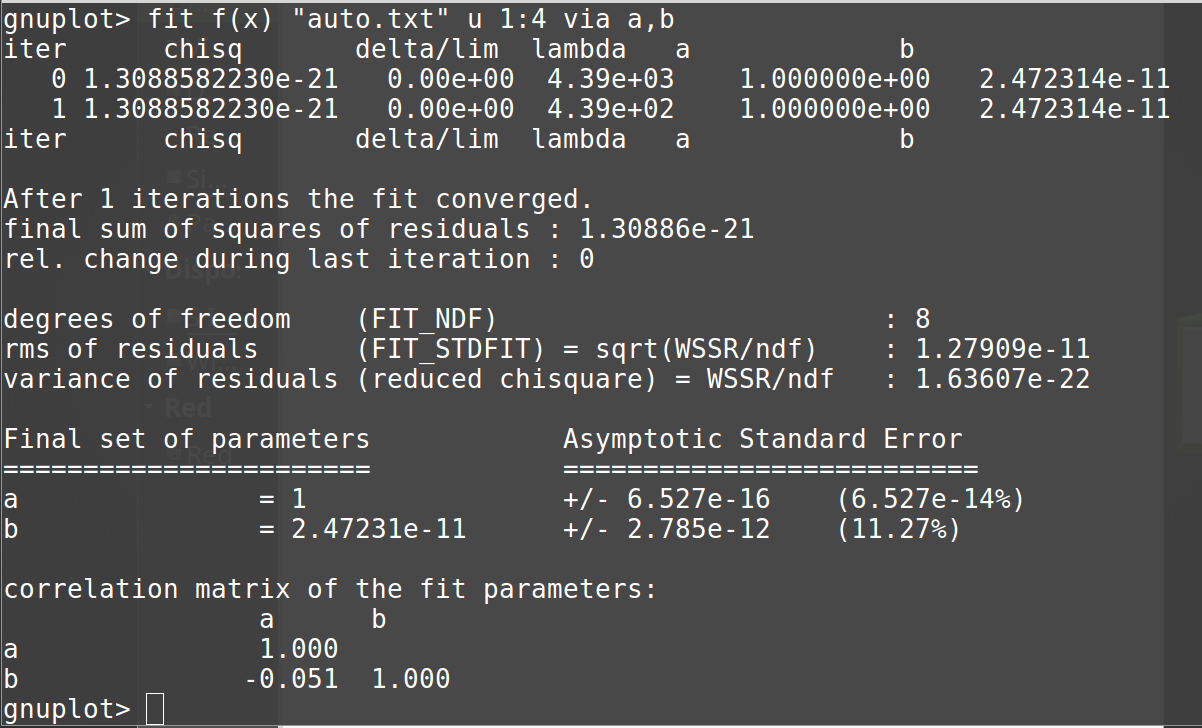
Como vemos se acerca a f(x) = x, lo esperado ya que Wblin(N) = O(N)

Gráfica de la función y el número máximo de OBs.



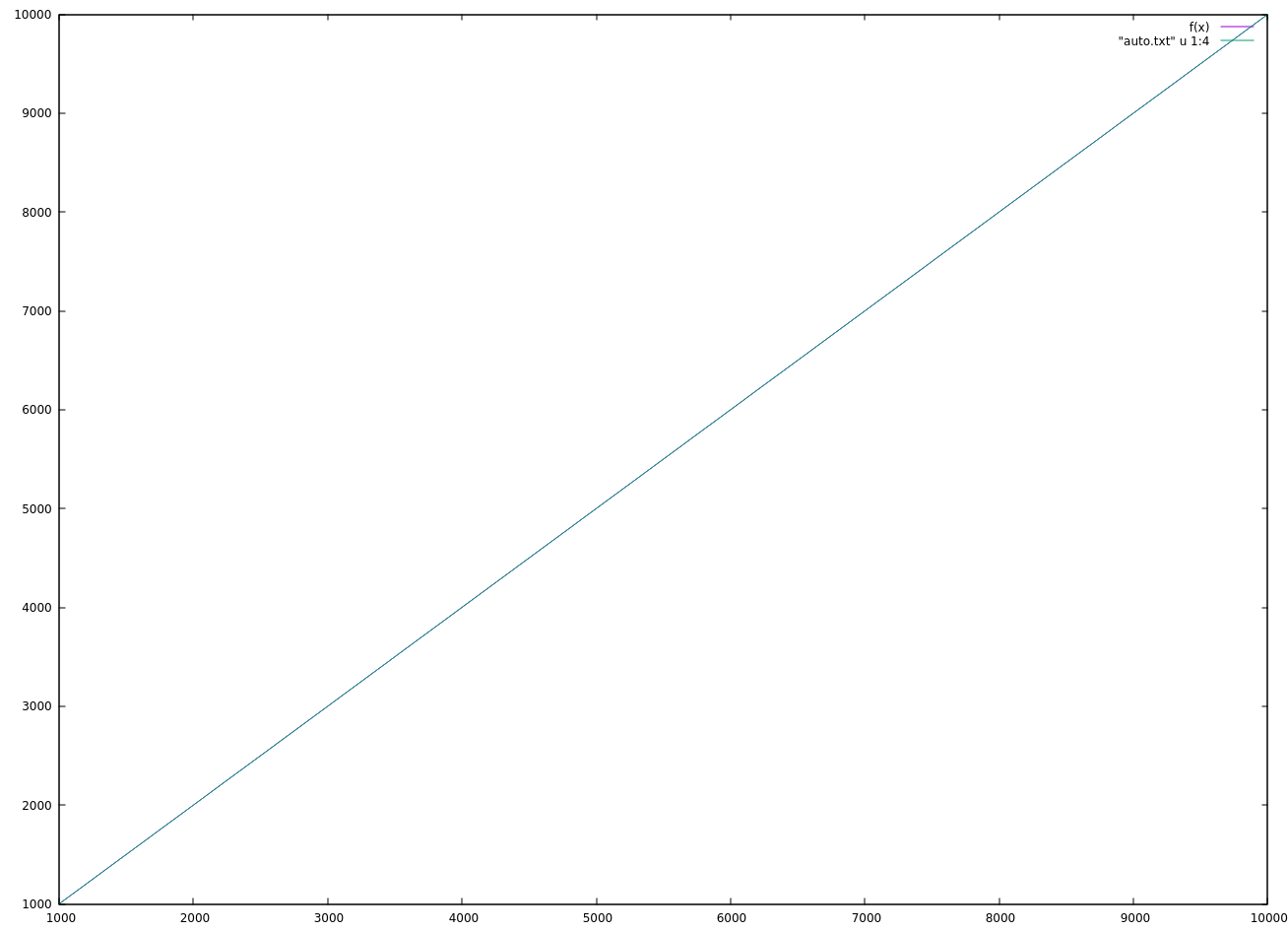
En la imagen parece una sola recta debido a que son prácticamente iguales ya que Wblin = N.

Aproximación de la misma función con el número máximo de la búsqueda auto organizada.



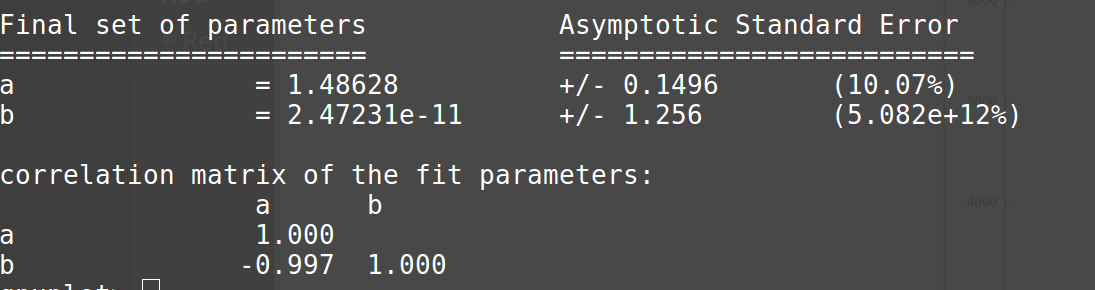
Obtenemos que se acerca a la misma función que la búsqueda lineal, ya que era lo que se esperaba.

Gráfica de f y el número máximo de OBs de búsqueda auto organizada:



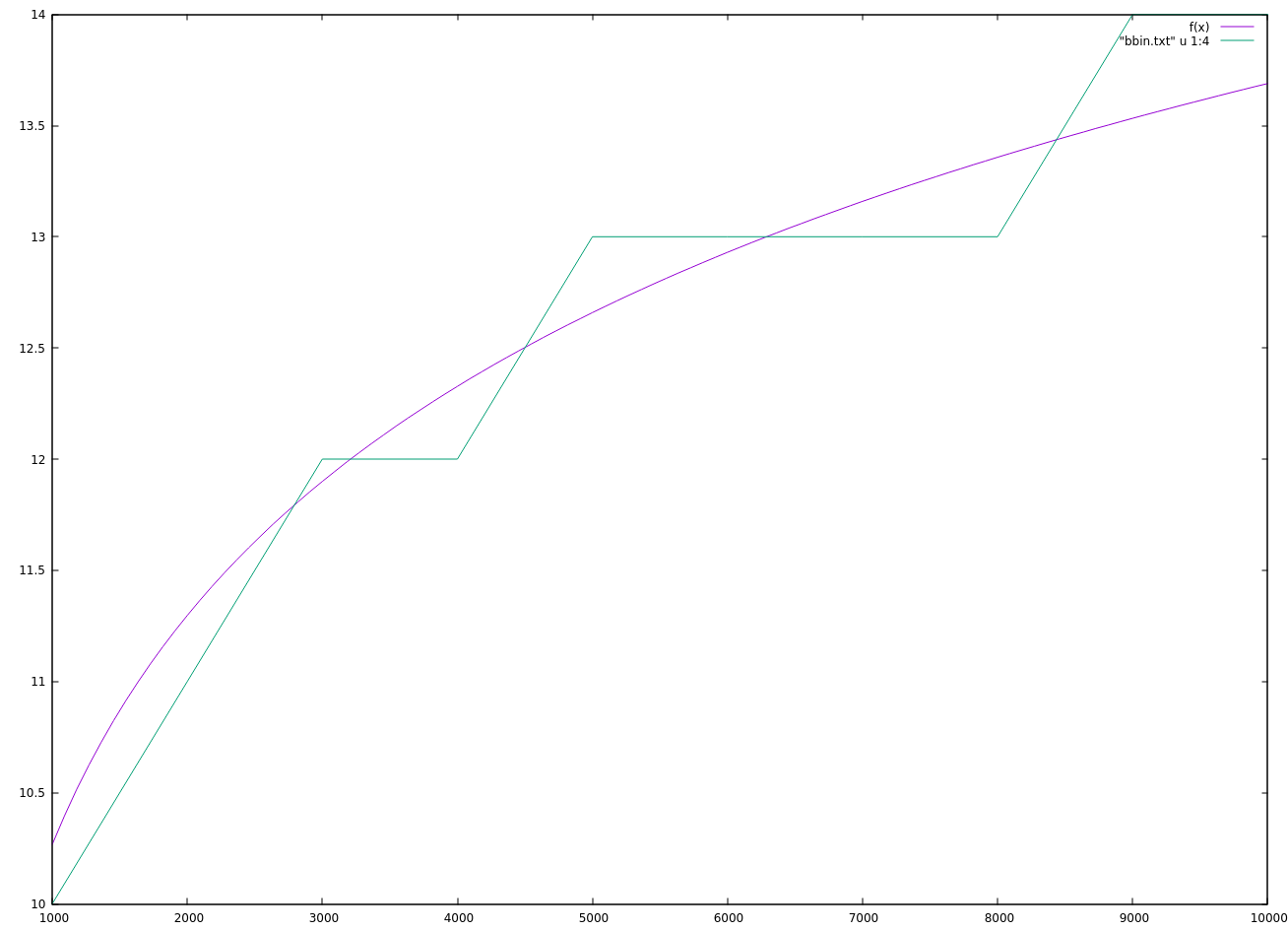
Cuando realizamos una sola búsqueda sobre las claves las búsqueda auto organizada se comporta de manera muy parecida a la búsqueda lineal.

Ahora aproximaremos la función f(x) = a\*log(x) + b con el número máximo de OBs de búsqueda binaria.



El a que obtenemos es exactamente 1/log(2). Esto es lo que esperábamos ya que Wbbin = ⎡log2 (N)⎤= ⎡log(N) / log(2) ⎤

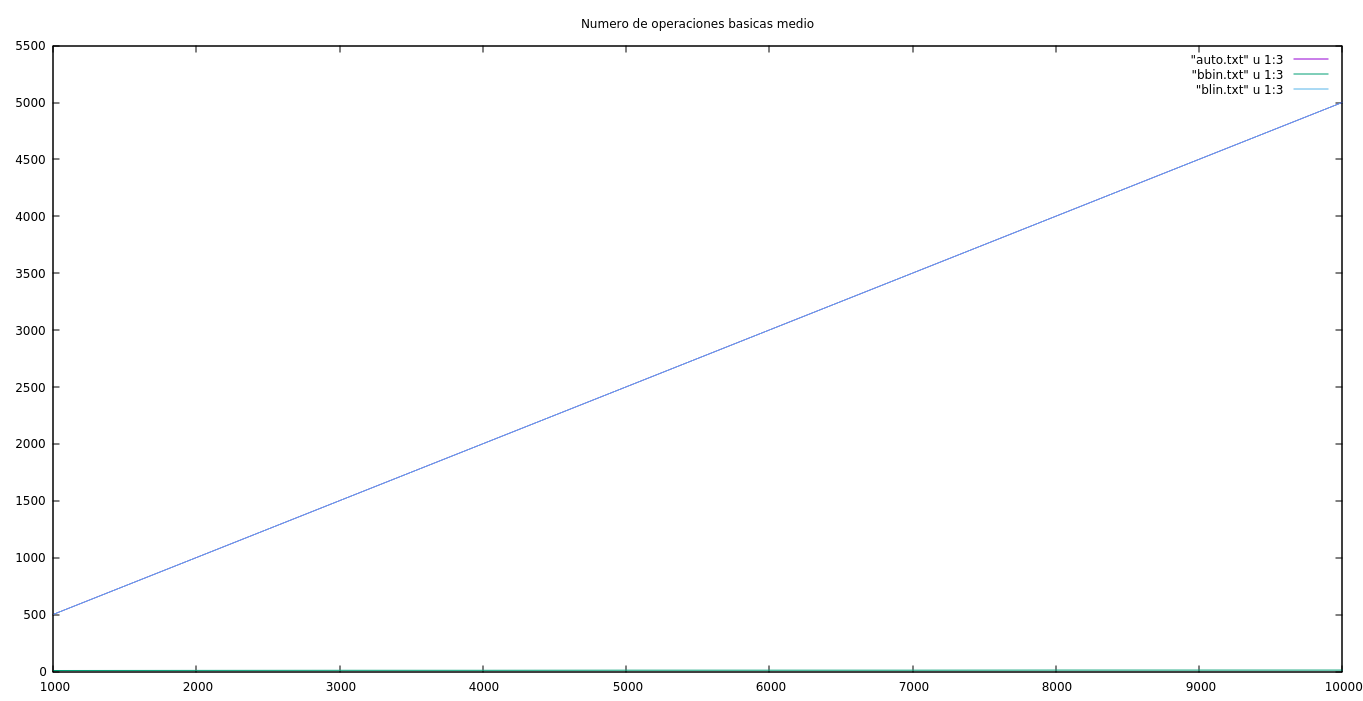
Representación de f con el número máximo de OBs de la búsqueda binaria:



Como vemos aparece escalonada, esto es porque es el techo del logaritmo. Y por eso parece que se aleja de f, esto es debido a que f es la función logarítmica y la gráfica de la búsqueda binaria es el techo de esa función.

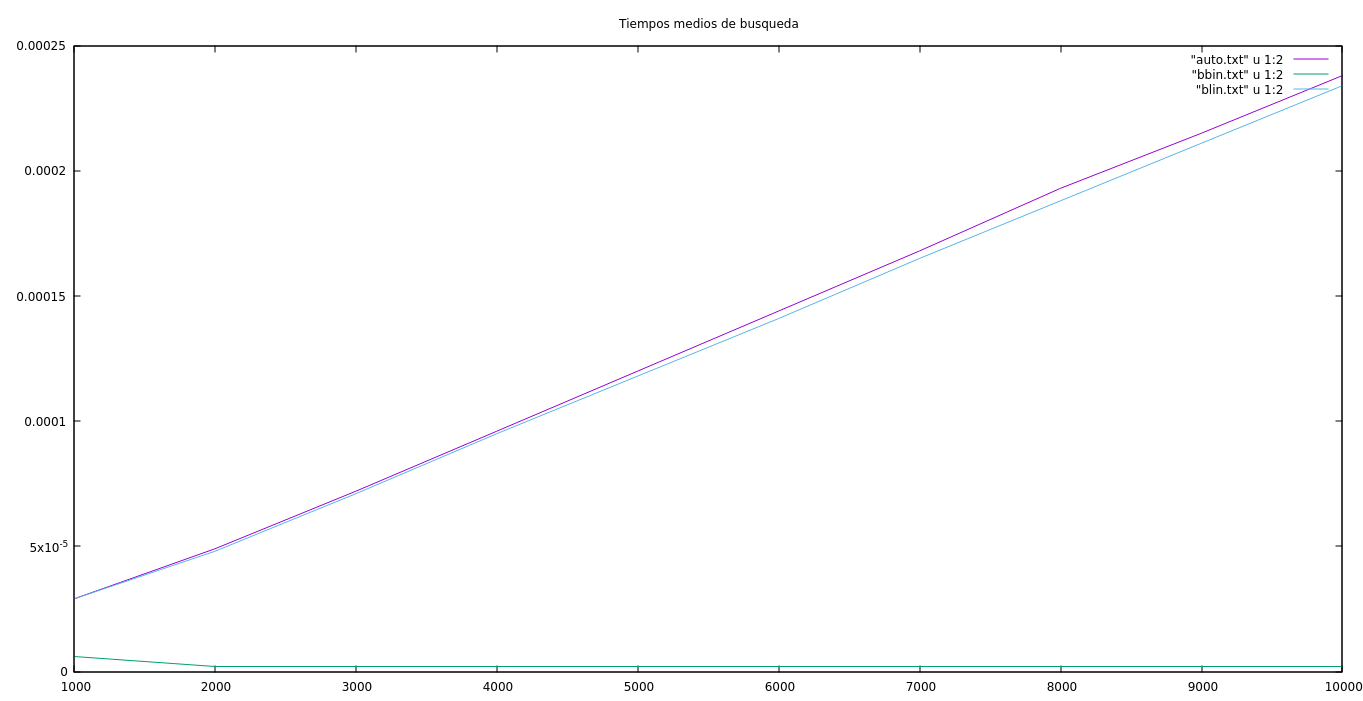
5.2 Apartado 2

Gráfica comparando el número promedio de OBs entre las búsquedas lineales y la búsqueda binaria.



El promedio de la búsqueda lineal y lineal auto organizada es prácticamente igual ya que solo se busca cada clave una vez. Comprobamos como el promedio de la búsqueda binaria es mucho menor. El número máximo de la búsqueda binaria hemos comprobado que estaba en torno a 13, sin embargo, el máximo de las búsquedas lineales es muy cercano a N ya que debe recorrer toda la tabla. Esto explica el por qué de la diferencia entre el promedio de una y las otras.

Gráfica comparando el tiempo promedio de reloj entre la búsqueda lineal y la búsqueda binaria.

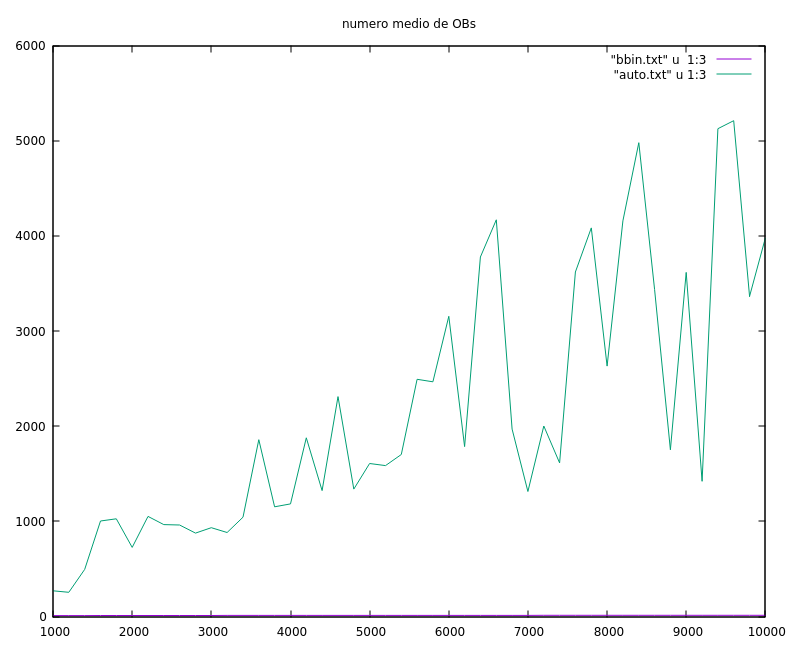


Vemos como la búsqueda binaria tarda menos que las lineales, esto es porque ya que hay menos comparaciones de clave la función acaba mucho antes. La búsqueda lineal auto organizada observamos que tarda prácticamente lo mismo que la lineal. La razón por la que puede que tarde un poco más, despreciable prácticamente ya que nos encontramos en una escala muy pequeña, es debido a que tiene más líneas de código.

La búsqueda auto organizada se estudia en el siguiente apartado, pero se ve cómo aumenta su efectividad según aumenta n\_veces.

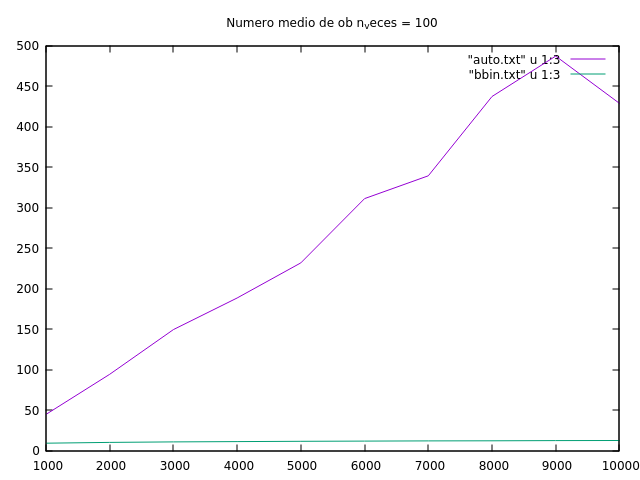
Gráfica comparando el número promedio de OBs entre la búsqueda binaria y la búsqueda lineal auto organizada (para los valores de n\_veces=1, 100 y 10000), comentarios a la gráfica.

-Cuando n\_veces = 1:



Como vemos el promedio de OBs de la búsqueda auto organizada es mucho mayor que en la búsqueda binaria. Hay que tener en cuenta que cuanto mayor es n\_veces más se verá su efectividad, de la búsqueda auto organizada.

-Cuando \_veces = 100 :

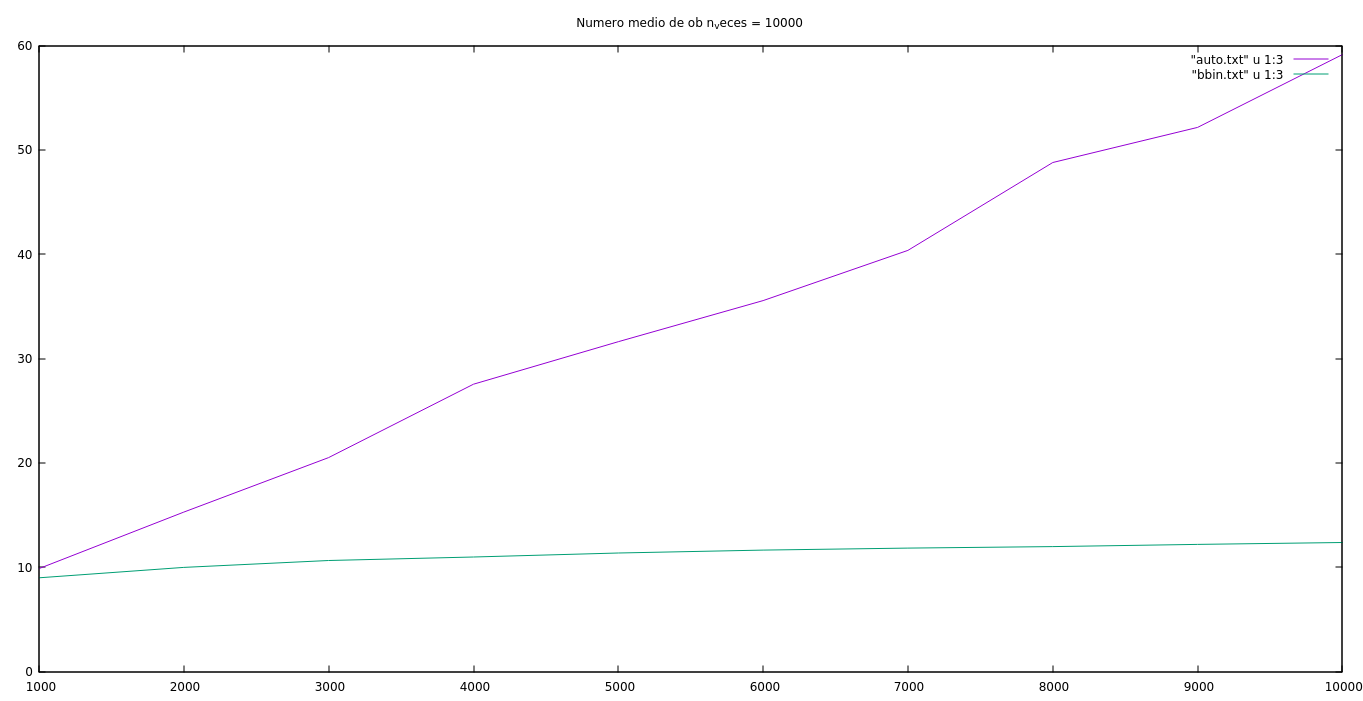


Observamos que el promedio de OBs sigue siendo superior en auto.txt, esto es debido a que cuándo los elementos están al final de la tabla se debe recorrer entera.

Aunque el número mínimo entre ambas sea muy parecido el máximo de la auto organizada es mucho mayor y esto hace que el promedio también sea mucho mayor.

Sin embargo, cabe destacar que ahora el máximo de la gráfica se alcanza en torno a 500 y cuando n\_veces = 1 era en torno a 5500. Por lo que vemos que cuantas más veces se deben buscar las claves más efectivo es este tipo de búsqueda, utilizando un generador de claves potencial.

-Cuando n\_veces = 10000:



Observamos lo mismo que en el anterior apartado y observamos el mismo comportamiento, el punto máximo de la gráfica ahora se encuentra a 60. En búsqueda binaria nos encontramos siempre en torno a 10.

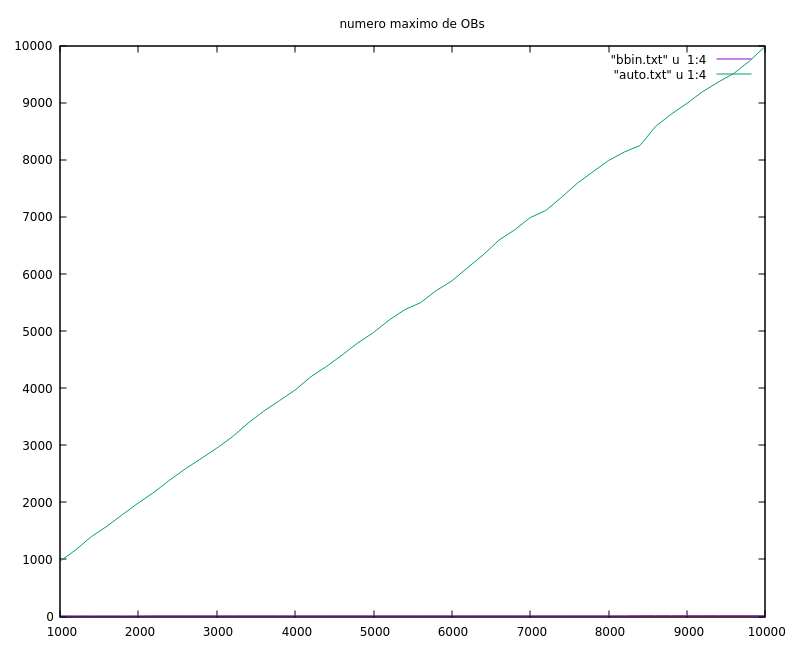
Aunque parezca que búsqueda binaria es mucho más efectiva, ya que los tiempos son mucho menores y el promedio de OBs también es menor, el diccionario debe estar ordenado, por lo que la inserción será más lenta. Y como vemos cuando n\_veces es grande el promedio se sigue diferenciando pero no de manera tan pronunciada y no es necesario operar sobre un diccionario ordenado.

En la búsqueda lineal auto organizada hemos comprobado que, al menos, en el promedio cuántas más veces se busca una clave más efectiva es.

Para decidir que algoritmo de búsqueda es el mejor deberíamos saber en que ocasiones se utilizará y cómo será el diccionario.

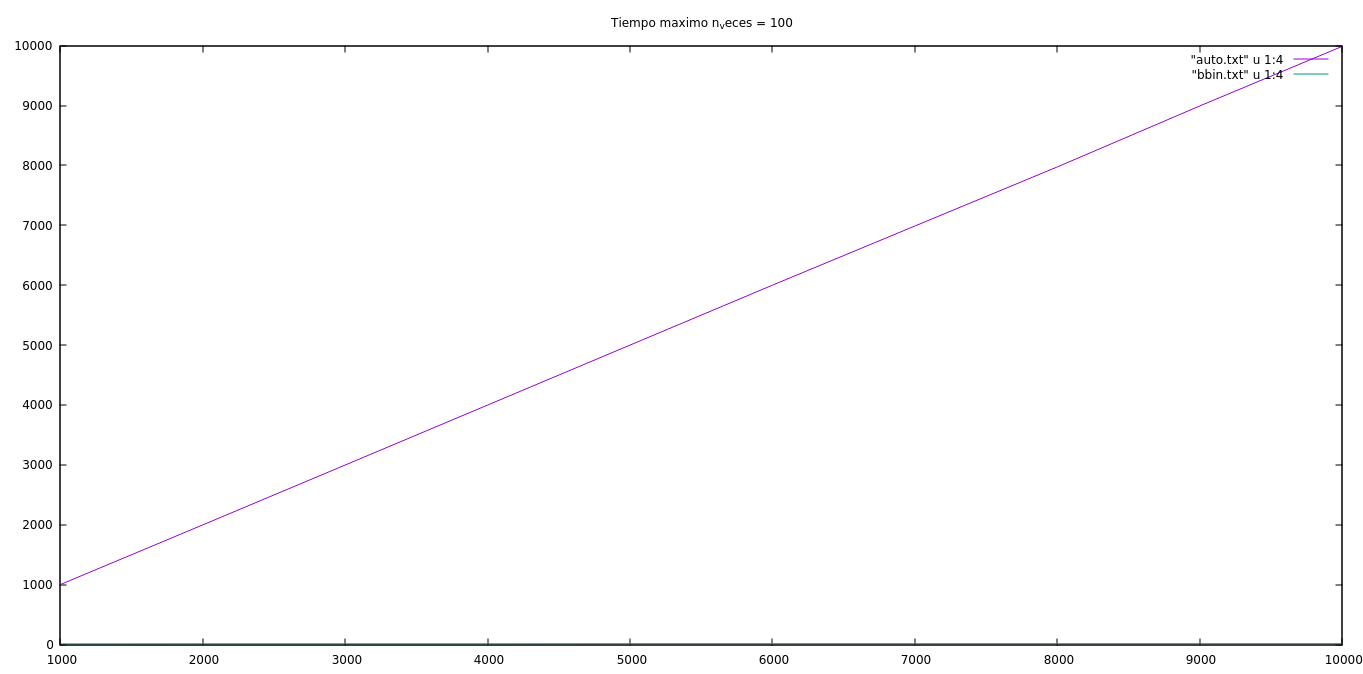
Gráfica comparando el número máximo de OBs entre la búsqueda binaria y la búsqueda lineal auto organizada (para los valores de n\_veces=1, 100 y 10000).

-Cuando n\_veces = 1:



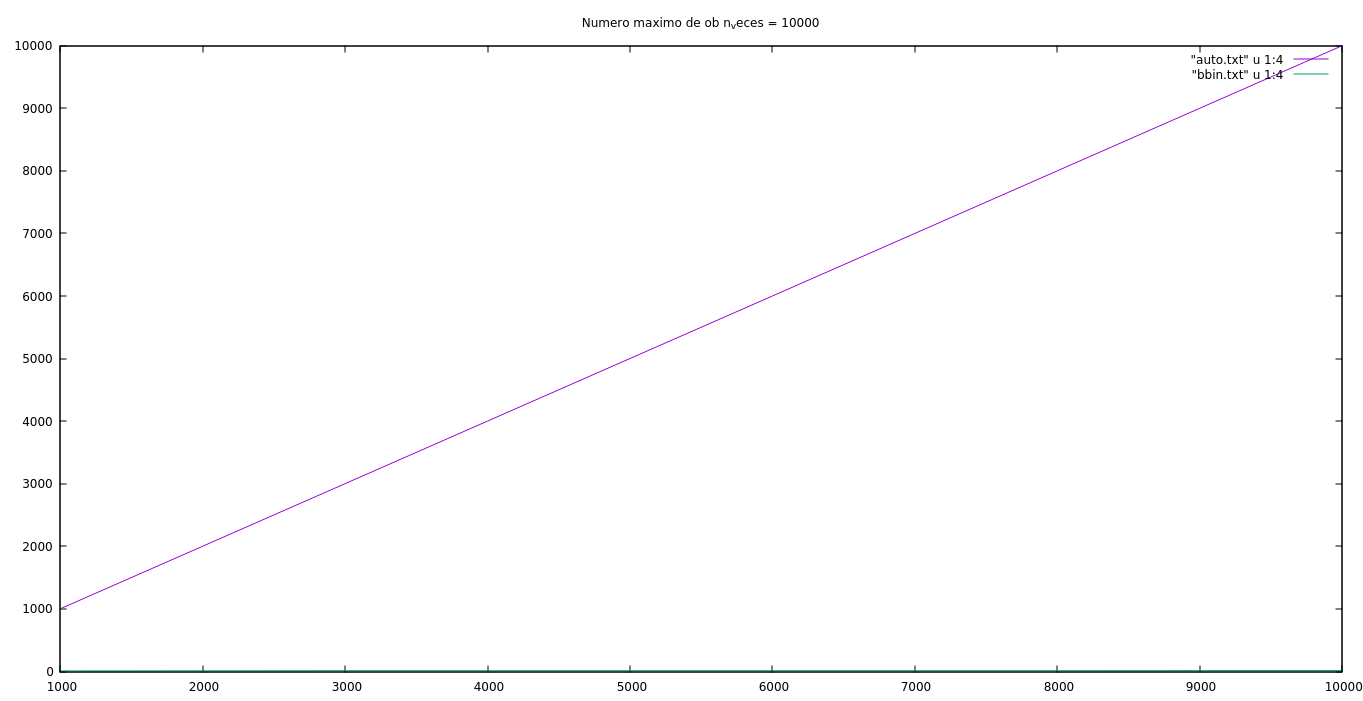
Vemos una diferencia muy grande entre los máximos de ambas gráficas. Esto es debido a que el máximo de OBs en búsqueda binaria se encuentra en torno a 13 y en búsqueda lineal auto organizada es siempre cercano a N.

-Cuando n\_veces = 100



Observamos que en esta gráfica está descompensado el número de OBs ya que el máximo de la búsqueda binaria se encuentra en torno a 14 y el máximo de la búsqueda auto organizada se encuentra en torno a 10000. El máximo de la búsqueda auto organizada sigue estando en torno a N.

-Cuando n\_veces = 10000

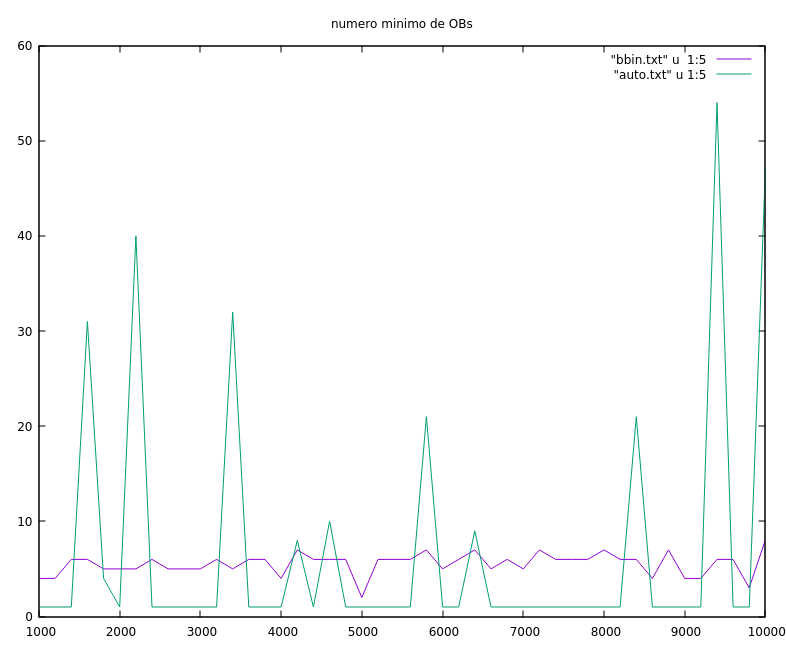


Aquí ocurre lo mismo que en el anterior apartado.

De manera que por lo que hemos observado en la práctica, el número máximo de OBs, el caso peor, no varía, o al menos no lo hace de manera notable, cuándo aumentamos el número de n\_veces. Es decir, si fuéramos siempre a buscar elementos de las últimas posiciones de la tabla (desordenada) la búsqueda auto organizada no es muy eficiente.

Gráfica comparando el número mínimo de OBs entre la búsqueda binaria y la búsqueda lineal auto organizada (para los valores de n\_veces=1, 100 y 10000.

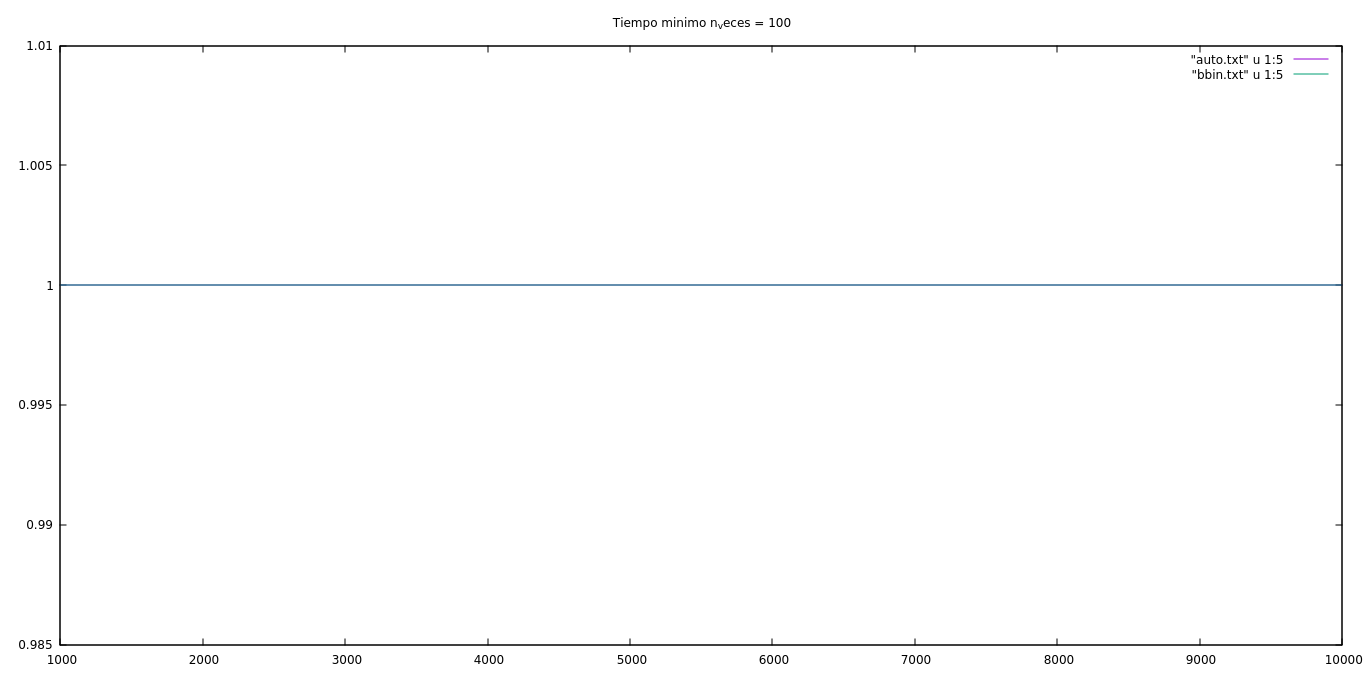
-Cuando n\_veces = 1



En ambas gráficas nos encontramos entre 0 y 10. Los picos de la búsqueda auto organizada es debido a que en n\_veces = 1 cogimos un incremento menor y en algunas tablas se tuvo “mala suerte” y el menor número de OBs realizado fue 20. En esta gráfica vemos como a veces es menor la búsqueda auto organizada que la binaria y otras, la búsqueda binaria.

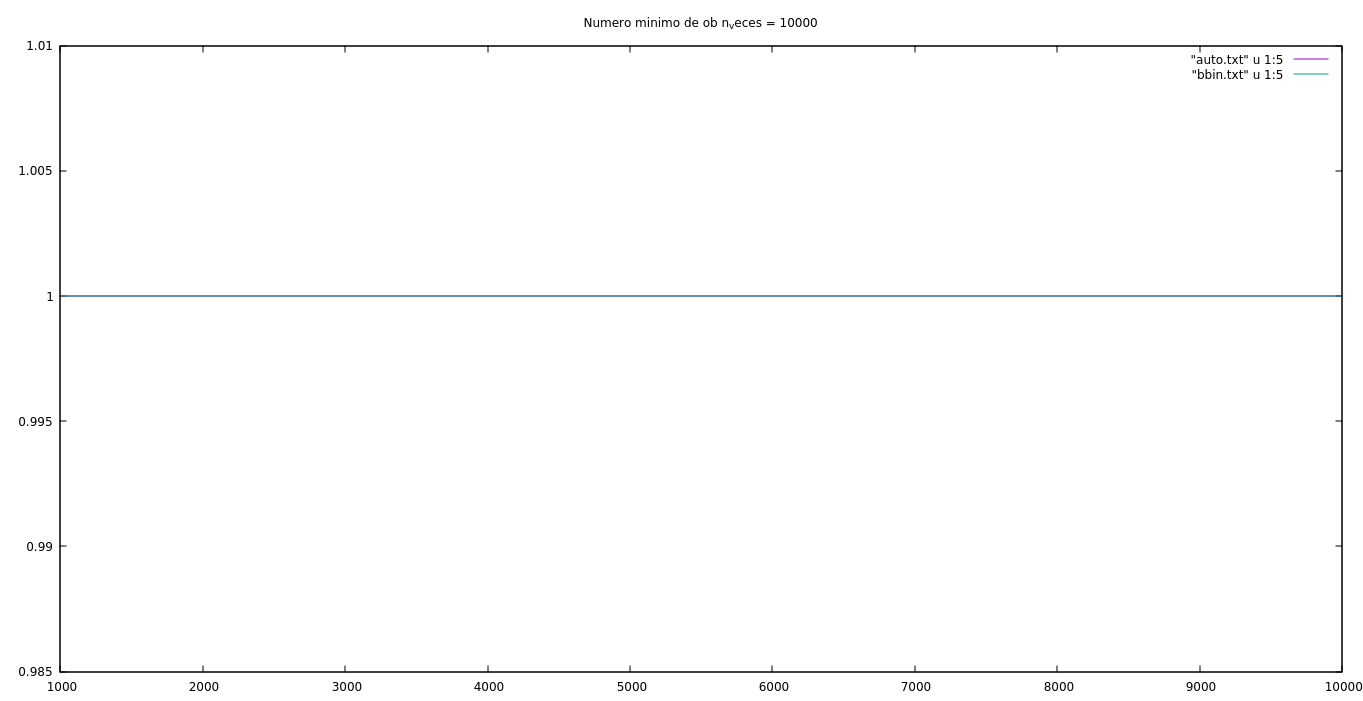
Puede depender el mínimo de que coincida y en el intervalo que analizamos nunca encuentren la clave “a la primera”. Pero, en ambas sabemos de manera teórica que Bbin = Bauto = O(1). Y vemos que eso coincide con lo esperado, aunque no obtengamos siempre como mínimo 1.

-Cuando n\_veces = 100



En ambas búsquedas el número mínimo es 1.

-Cuando n\_veces = 10000



Observamos que el número mínimo de OBs es siempre 1, independientemente del número de veces que busquemos una clave (nunca podría ser menor que 1).

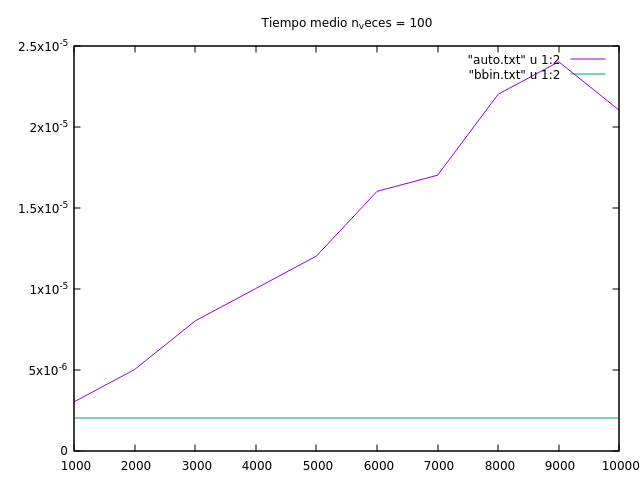
Gráfica comparando el tiempo medio de reloj entre la búsqueda binaria y la búsqueda lineal auto organizada (para los valores de n\_veces=1, 100 y 10000).

-Cuando n\_veces = 1:

En los tiempos medios obtuvimos tanto en búsqueda binaria como en búsqueda auto organizada 0 s, esto es debido a que lo hace tan rápido que no le da tiempo a calcular el tiempo.

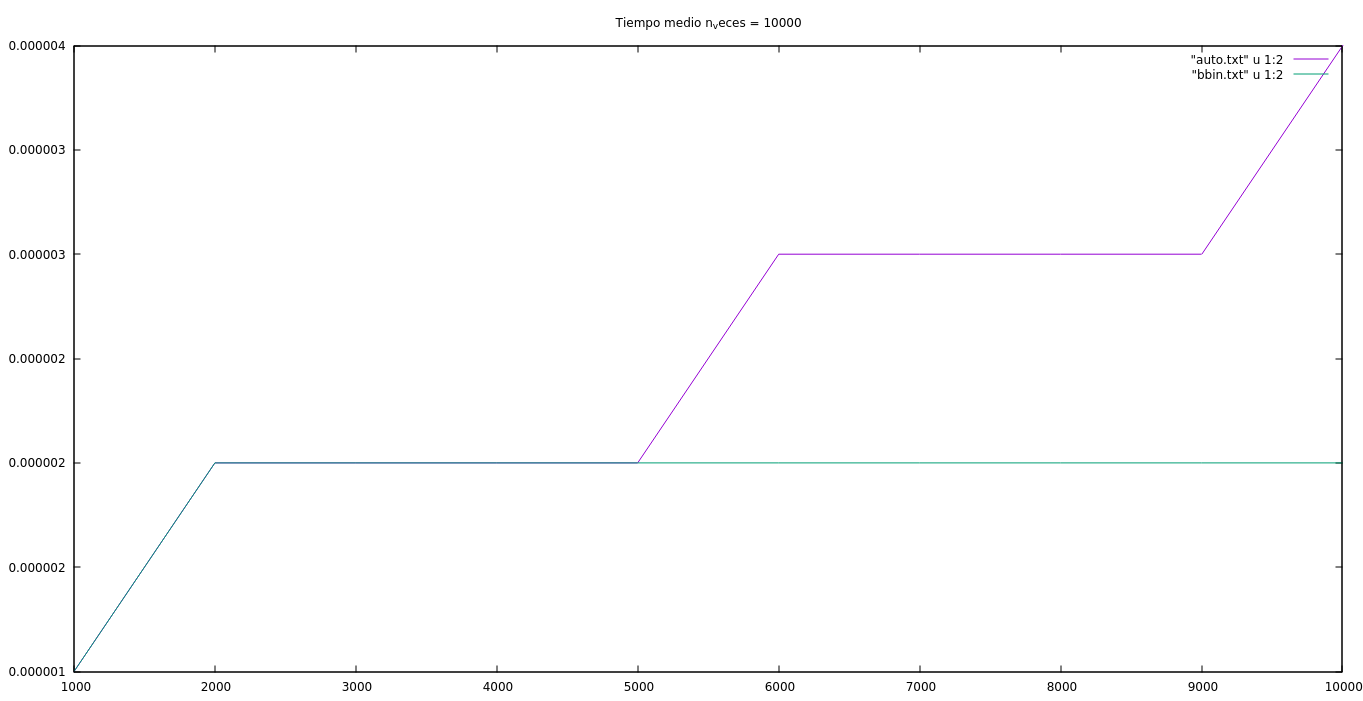
Aunque, podemos deducir que el de la búsqueda binaria sería menor ya que el número de veces que se realiza la función es mucho menor que el de la búsqueda auto organizada. Esto lo sabemos observando los resultados obtenidos al contar el promedio de las OBs realizadas.

-Cuando n\_veces = 100



Observamos que el tiempo medio de la búsqueda auto organizada es mayor, aunque hay que tener en cuenta que estamos en escalas muy bajas a pesar de la gran diferencia que se aprecia en la gráfica. Esto es debido a que en general el número de OBs es siempre mayor, por lo tanto la función se estará realizando durante más tiempo.

-Cuando n\_veces = 10000



Aquí observamos que los tiempos se igualan, más o menos, la escala aquí es mucho menor luego la diferencia entre ambas gráficas es prácticamente inapreciable. El tiempo de la búsqueda auto organizada disminuye ya que al aumentar el número de veces que se busca una clave también disminuye el promedio de comparaciones de clave y por lo tanto también el tiempo que se está realizando el algoritmo.

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

Aquí respondéis a las preguntas teóricas que se os han planteado en la práctica.

5.1 Pregunta 1

La comparación de clave.

En bbin cuando comparamos el elemento de la posición media de la tabla con la clave que buscamos.

En búsqueda lineal y auto organizada, la comparación entre el elemento de la posición que comparamos (va aumentando linealmente la posición desde el primero hasta que se encuentre, o el último si no está) y la clave que buscamos.

5.2 Pregunta 2

Wblin = Θ(N)

Wbbin = ⎡log2 (N)⎤

Bblin = O(1)

Bbbin = O(1)

5.3 Pregunta 3

En la búsqueda lineal auto organizada el elemento que se busca, una vez que se encuentra se realiza un swap con el elemento anterior. De manera que los elementos que más se busquen estarán siempre en las primeras posiciones de la tabla y por tanto el tiempo como el número de OBs que se realice para encontrarlo será menor respecto al resto de elementos.

5.4 Pregunta 4

5.5 Pregunta 5

Búsqueda binaria actúa solo en tablas ordenadas. De manera que mira en el elemento de la posición media de la tabla su valor y si la clave que buscamos es menor hará lo mismo con la primera mitad de la tabla y si no con la segunda, y así sucesivamente. Y si es igual pues ya ha encontrado el elemento.

Cuando se le pasa una tabla de un elemento y ese elemento no es la clave que se busca el algoritmo devuelve NO\_ENCONTRADO, porque no se cumpliría la condición del while.

Va comparando con los valores de las posiciones medias de las tablas y así evita tener que recorrerla entera.

**6. Conclusiones finales.**

En esta práctica hemos comparado distintas formas de búsqueda.

Hemos implementado el TAD Diccionario, hay dos tipos uno ordenado y otro no ordenado. De esta manera hemos comparado la búsqueda lineal y la búsqueda auto organizada con la búsqueda binaria.

Hemos visto con los resultados experimentales que la búsqueda auto organizada es más útil cuando se realiza un número de búsquedas mayor en ciertas claves, de esta manera disminuyen los tiempos.

En las gráficas hemos visto que los tiempos de la búsqueda binaria eran siempre menores y aparentemente en cuanto a las comparaciones de clave también era más eficiente. Sin embargo, la búsqueda binaria solo sirve en tablas ordenadas de manera que solo nos serviría para diccionarios ordenados, y esto conlleva cierto coste ya que para insertar cada clave se deberían de hacer ciertas comparaciones para colocarla en orden.

Cuando no se van a realizar muchas búsquedas sobre cierta clave, en este caso, puede ser más eficiente un diccionario ordenado y utilizar la búsqueda binaria ya que aunque para añadir cada clave al diccionario se realicen ciertas comparaciones de clave, hemos comprobado que el número máximo de las comparaciones de clave de la búsqueda binaria se encontraba en torno a 13 frente a N en búsqueda lineal y en búsqueda lineal auto organizada.

Sin embargo, si los elementos del diccionario son una permutación totalmente “dada la vuelta”, es decir, de la forma {N, N – 1, …, 2, 1}, entonces hay que tener en cuenta que si contáramos las OBs de comparación del algoritmo de ordenación para introducir claves en un diccionario ordenado sería N^2 / 2 – N/2. En estos casos, si por ejemplo se van a buscar ciertas claves con mayor regularidad, puede ser más eficiente la búsqueda auto organizada y utilizar un diccionario NO\_ORDENADO.

O también, si se fueran a buscar claves cercanas a N, búsqueda lineal o auto organizada, auto organizada si se van a buscar ciertas claves con más frecuencia, ya que como no ordenaríamos la taba estos elementos se encontrarían al principio.

Otra solución, para permutaciones totalmente “dadas a vuelta”, o muy parecida sería ordenar el diccionario de mayor a menor y realizar una modificación en la búsqueda binaria, de manera que la eficiencia sería similar a la que tiene actualmente cuando le pasamos tablas muy ordenadas