Análisis de Algoritmos 2019/2020

Práctica 3

Guillermo López Rodríguez, Javier Fernandez de Alarcón Gervás, Grupo 1261

1. Introducción.

En esta práctica el objetivo principal es implementar un diccionario, para poder implementar algoritmos de búsqueda como son la búsqueda lineal, búsqueda binaria o la búsqueda lineal autoorganizada.

2. Objetivos

2.1 Apartado 1

El objetivo de este apartado es el de implementar el diccionario creando el archivo busqueda.c y busqueda.h con las funciones necesarias para crear,liberar y manejar el diccionario como queramos.

Posteriormente implementaremos los algoritmos de búsqueda binaria, busqueda lineal y búsqueda autoorganizada.

2.2 Apartado 2

El objetivo de este apartado es el de comparar los rendimientos de búsqueda de los algoritmos empleados mediante el archivo tiempo.c por lo que debemos implementar dos nuevas funciones: tiempo_medio_busqueda y genera_tiempos_busqueda.

3. Herramientas y metodología

En esta Práctica hemos utilizado tanto Linux como MacOS. En cuanto a herramientas,un miembro del grupo usó Visual Studio y el otro Sublime Text, evidentemente para comprobar las fugas de memoria se ha utilizado Valgrind. En el caso de las gráficas/histograma se ha hecho uso de Gnuplot y Excel.

3.1 Apartado 1

En este apartado debemos implementar varias funciones de búsqueda.c como las del diccionario y los algoritmos de búsqueda, para implementar los algoritmos de búsqueda nos hemos fijado en la diapositivas de teoría para poder crearlos en lenguaje C.

3.2 Apartado 2

En este apartado debemos implementar dos funciones tiempo_medio_busqueda y genera_tiempos_busqueda. Para lograrlo hemos seguido las instrucciones aportadas en el pdf del enunciado de la práctica.

4. Código fuente

4.1 Apartado 1

```
PDICC ini_diccionario (int tamanio, char orden){
        /* vuestro codigo */
  PDICC dicc = NULL;

  dicc=(PDICC)malloc(tamanio*sizeof(DICC));

  if(!dicc)return NULL;


  dicc->tamanio=tamanio;

  dicc->n_datos=0;

  dicc->orden= orden;


  dicc->tabla=(int*)malloc(tamanio*sizeof(int));

  if(!dicc->tabla)return NULL;

  return dicc;

}



void libera_diccionario(PDICC pdicc){
```

```
        /* vuestro codigo */

 free(pdicc->tabla);

 free(pdicc);

}

int inserta_diccionario(PDICC pdicc, int clave){
        /* vuestro codigo */
  int ob=0;
  /*int A,j; */
  if(!pdicc->tabla || clave < 0)
                return ERR;

        pdicc->tabla[pdicc->n_datos] = clave;




        if(pdicc->orden == ORDENADO) {

                ob= InsertSort(pdicc->tabla, 0, pdicc->n_datos);

    /* A=pdicc->tabla[pdicc->n_datos];
    j=pdicc->n_datos-1;

    while (j >= 0 && pdicc->tabla[j]>A){
      pdicc->tabla[j+1]=pdicc->tabla[j];
      j--;

    }
    pdicc->tabla[j+1]=A;
    */

  }

  else if(pdicc->orden == NO_ORDENADO){
    pdicc->n_datos++;
                return ob;
  }
    else {
                  return ERR;
  }

  pdicc->n_datos++;

  return ob;
```

```
}

int insercion_masiva_diccionario (PDICC pdicc,int *claves, int n_claves){
        /* vuestro codigo */

  int i;

        if(!pdicc->tabla || !claves || n_claves < 1)
                return ERR;

        for(i = 0; i < n_claves; i++) {
                if(inserta_diccionario(pdicc, claves[i]) == ERR)
                        return ERR;
        }



        return OK;

}

int busca_diccionario(PDICC pdicc, int clave, int *ppos, pfunc_busqueda metodo){
        /* vuestro codigo */


  return metodo(pdicc->tabla,0,pdicc->n_datos,clave,ppos);

}


/* Funciones de busqueda del TAD Diccionario */
int bbin(int *tabla,int P,int U,int clave,int *ppos){
        /* vuestro codigo */
  int mitad,i=0;


  while(P<=U){


    mitad=(P+U)/2;
    i++;


      if(tabla[mitad]==clave){

        *ppos=mitad;
        return i;

      }
```

```
    else if(tabla[mitad]>clave){

       U=mitad-1;
    }

    else{


       P=mitad+1;
    }



 }

   *ppos= NO_ENCONTRADO;

   return i;

}

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

        /* vuestro codigo */

 int i;

        if(!tabla || P < 0 || U < P || clave < 0 || !ppos)
                return ERR;

        for(i = 0; i < U; i++) {

        if(clave == tabla[i]){
                        *ppos = i;
                        return i + 1;
                }

        }

 *ppos= NO_ENCONTRADO;
        return i;

}



int blin_auto(int *tabla,int P,int U,int clave,int *ppos){
```

```
        /* vuestro codigo */


        int i;

        if(!tabla || P < 0 || U < P || clave < 0 || !ppos)
                return ERR;

   for(i = 0; i < U; i++) {

    if(clave == tabla[i]){

      if (i > 0) {
        swap(&tabla[i],&tabla[i-1]);
      }

      *ppos = i;
      return i + 1;

    }

   }

        *ppos= NO_ENCONTRADO;
        return i;

}
```

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){

        PTIEMPO tiempo = NULL;

        int i;

        int espacio;



        espacio=((num_max-num_min)/incr) +1;

        tiempo=(PTIEMPO)malloc(espacio * sizeof(TIEMPO));
```

```
	if(!tiempo)
		return ERR;

	for(i = 0; num_min <= num_max; i++) {
		tiempo_medio_busqueda(metodo, generador,orden,num_min,n_veces,
&tiempo[i]);
		num_min += incr;
	}

	if(guarda_tabla_tiempos(fichero, tiempo, i)==ERR) return ERR;

	free(tiempo);

	return OK;
}

short tiempo_medio_busqueda(pfunc_busqueda metodo, pfunc_generador_claves
generador,
int orden,int N,int n_veces,PTIEMPO ptiempo){

	int i,ob,min,max,sum,num=n_veces*N;
	PDICC dicc=NULL;
	int* perm;
	int* pos;
```

```
int* claves;
clock_t start, stop;
double total_time,total=0;

pos=(int*)malloc(sizeof(int));

if(!pos)return ERR;

claves=(int*)malloc(num*sizeof(int));

if(!claves)return ERR;


for(i=0;i<n_veces;i++){
        dicc=ini_diccionario(N, orden);
        perm=genera_perm(N);
        insercion_masiva_diccionario(dicc,perm, N);
        if(!claves)return ERR;
        generador(claves,num,N);
        for(i=0,min=0,max=0,sum=0,ob=0;i<num;i++){
                start= clock();
                ob=busca_diccionario(dicc,claves[i], pos,  metodo);
                stop=clock();
```

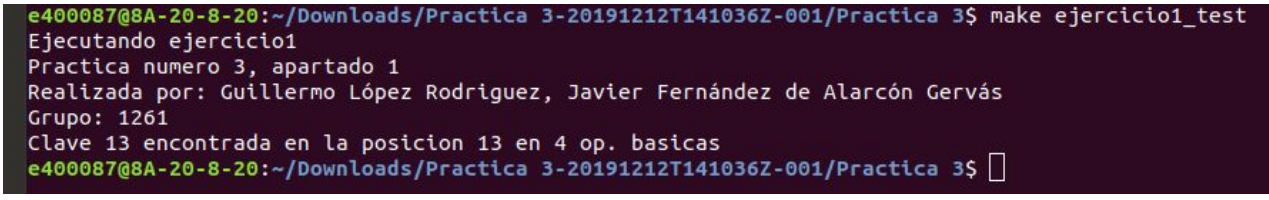
```
                        if(max == 0 && min == 0) {
                                max = min = ob;
                        }
                        if(ob > max)
                                max = ob;
                        if(ob < min)
                                min = ob;

                        sum += ob;



                        total+= stop-start;
        }

        free(perm);
        libera_diccionario(dicc);
}

total_time=(double)(total/num)/(double)CLOCKS_PER_SEC;

ptiempo->N = N;
ptiempo->n_elems = num;
ptiempo->min_ob = min;
ptiempo->max_ob = max;
```

```
        ptiempo->medio_ob = 1.*sum/i;
        ptiempo->tiempo = total_time;


        free(claves);
        free(pos);
        return OK;
}
```

5. Resultados, Gráficas

5.1 Apartado 1

Resultado utilizando Búsqueda Binaria:



Resultado utilizando Búsqueda lineal:

5.2 Apartado 2

Resultados utilizando la búsqueda lineal en tablas desordenadas entre 1000 y 10000 con incremento de 250.

ejercicio2.log

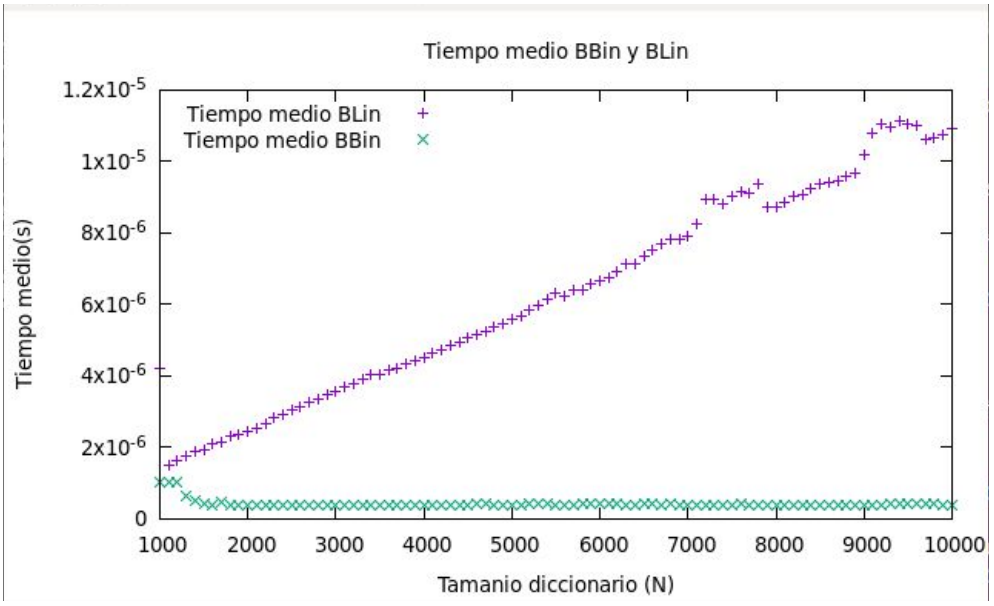
```
N= 1000 Tiempo= 0.000003686000  MedioOb= 500.000 MinOB= 1 MaxOB= 1000
N= 1250 Tiempo= 0.000001784800  MedioOb= 625.000 MinOB= 1 MaxOB= 1250
N= 1500 Tiempo= 0.000002156667  MedioOb= 750.000 MinOB= 1 MaxOB= 1500
N= 1750 Tiempo= 0.000002491429  MedioOb= 875.000 MinOB= 1 MaxOB= 1750
N= 2000 Tiempo= 0.000002815500  MedioOb= 1000.000 MinOB= 1 MaxOB= 2000
N= 2250 Tiempo= 0.000003200444  MedioOb= 1125.000 MinOB= 1 MaxOB= 2250
N= 2500 Tiempo= 0.000003209600  MedioOb= 1250.000 MinOB= 1 MaxOB= 2500
N= 2750 Tiempo= 0.000003325818  MedioOb= 1375.000 MinOB= 1 MaxOB= 2750
N= 3000 Tiempo= 0.000003521333  MedioOb= 1500.000 MinOB= 1 MaxOB= 3000
N= 3250 Tiempo= 0.000003722462  MedioOb= 1625.000 MinOB= 1 MaxOB= 3250
N= 3500 Tiempo= 0.000004116000  MedioOb= 1750.000 MinOB= 1 MaxOB= 3500
N= 3750 Tiempo= 0.000004364533  MedioOb= 1875.000 MinOB= 1 MaxOB= 3750
N= 4000 Tiempo= 0.000004637500  MedioOb= 2000.000 MinOB= 1 MaxOB= 4000
N= 4250 Tiempo= 0.000004898118  MedioOb= 2125.000 MinOB= 1 MaxOB= 4250
N= 4500 Tiempo= 0.000005176889  MedioOb= 2250.000 MinOB= 1 MaxOB= 4500
N= 4750 Tiempo= 0.000005452000  MedioOb= 2375.000 MinOB= 1 MaxOB= 4750
N= 5000 Tiempo= 0.000005720800  MedioOb= 2500.000 MinOB= 1 MaxOB= 5000
N= 5250 Tiempo= 0.000005992190  MedioOb= 2625.000 MinOB= 1 MaxOB= 5250
N= 5500 Tiempo= 0.000006262182  MedioOb= 2750.000 MinOB= 1 MaxOB= 5500
N= 5750 Tiempo= 0.000006528174  MedioOb= 2875.000 MinOB= 1 MaxOB= 5750
N= 6000 Tiempo= 0.000006709833  MedioOb= 3000.000 MinOB= 1 MaxOB= 6000
N= 6250 Tiempo= 0.000006888480  MedioOb= 3125.000 MinOB= 1 MaxOB= 6250
N= 6500 Tiempo= 0.000007145231  MedioOb= 3250.000 MinOB= 1 MaxOB= 6500
N= 6750 Tiempo= 0.000007418519  MedioOb= 3375.000 MinOB= 1 MaxOB= 6750
N= 7000 Tiempo= 0.000007742286  MedioOb= 3500.000 MinOB= 1 MaxOB= 7000
N= 7250 Tiempo= 0.000008011862  MedioOb= 3625.000 MinOB= 1 MaxOB= 7250
N= 7500 Tiempo= 0.000008416267  MedioOb= 3750.000 MinOB= 1 MaxOB= 7500
N= 7750 Tiempo= 0.000008664516  MedioOb= 3875.000 MinOB= 1 MaxOB= 7750
N= 8000 Tiempo= 0.000008975875  MedioOb= 4000.000 MinOB= 1 MaxOB= 8000
N= 8250 Tiempo= 0.000009057939  MedioOb= 4125.000 MinOB= 1 MaxOB= 8250
N= 8500 Tiempo= 0.000009285647  MedioOb= 4250.000 MinOB= 1 MaxOB= 8500
N= 8750 Tiempo= 0.000009527657  MedioOb= 4375.000 MinOB= 1 MaxOB= 8750
N= 9000 Tiempo= 0.000009780444  MedioOb= 4500.000 MinOB= 1 MaxOB= 9000
N= 9250 Tiempo= 0.000010123784  MedioOb= 4625.000 MinOB= 1 MaxOB= 9250
N= 9500 Tiempo= 0.000010329158  MedioOb= 4750.000 MinOB= 1 MaxOB= 9500
N= 9750 Tiempo= 0.000010577641  MedioOb= 4875.000 MinOB= 1 MaxOB= 9750
N= 10000 Tiempo= 0.000010850600  MedioOb= 5000.000 MinOB= 1 MaxOB= 10000

```

Resultados utilizando la búsqueda binaria en tablas ordenadas entre 1000 y 10000 con incremento de 250.

ejercicio2.log

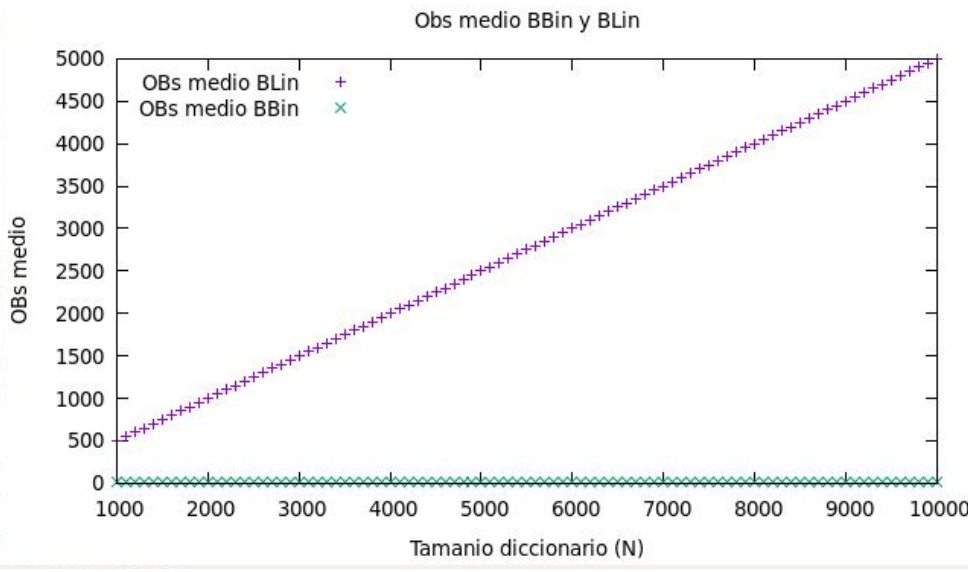
```
N= 1000 Tiempo= 0.000000372000  MedioOb= 8.978 MinOB= 1 MaxOB= 10
N= 1250 Tiempo= 0.000000388800  MedioOb= 9.364 MinOB= 1 MaxOB= 11
N= 1500 Tiempo= 0.000000374667  MedioOb= 9.636 MinOB= 1 MaxOB= 11
N= 1750 Tiempo= 0.000000421714  MedioOb= 9.831 MinOB= 1 MaxOB= 11
N= 2000 Tiempo= 0.000000388000  MedioOb= 9.977 MinOB= 1 MaxOB= 11
N= 2250 Tiempo= 0.000000377333  MedioOb= 10.181 MinOB= 1 MaxOB= 12
N= 2500 Tiempo= 0.000000392800  MedioOb= 10.363 MinOB= 1 MaxOB= 12
N= 2750 Tiempo= 0.000000388727  MedioOb= 10.511 MinOB= 1 MaxOB= 12
N= 3000 Tiempo= 0.000000392000  MedioOb= 10.635 MinOB= 1 MaxOB= 12
N= 3250 Tiempo= 0.000000388923  MedioOb= 10.740 MinOB= 1 MaxOB= 12
N= 3500 Tiempo= 0.000000390000  MedioOb= 10.830 MinOB= 1 MaxOB= 12
N= 3750 Tiempo= 0.000000379467  MedioOb= 10.908 MinOB= 1 MaxOB= 12
N= 4000 Tiempo= 0.000000383750  MedioOb= 10.977 MinOB= 1 MaxOB= 12
N= 4250 Tiempo= 0.000000386118  MedioOb= 11.073 MinOB= 1 MaxOB= 13
N= 4500 Tiempo= 0.000000384000  MedioOb= 11.180 MinOB= 1 MaxOB= 13
N= 4750 Tiempo= 0.000000384421  MedioOb= 11.276 MinOB= 1 MaxOB= 13
N= 5000 Tiempo= 0.000000385400  MedioOb= 11.362 MinOB= 1 MaxOB= 13
N= 5250 Tiempo= 0.000000381524  MedioOb= 11.440 MinOB= 1 MaxOB= 13
N= 5500 Tiempo= 0.000000384727  MedioOb= 11.511 MinOB= 1 MaxOB= 13
N= 5750 Tiempo= 0.000000388870  MedioOb= 11.576 MinOB= 1 MaxOB= 13
N= 6000 Tiempo= 0.000000387667  MedioOb= 11.635 MinOB= 1 MaxOB= 13
N= 6250 Tiempo= 0.000000384480  MedioOb= 11.690 MinOB= 1 MaxOB= 13
N= 6500 Tiempo= 0.000000385077  MedioOb= 11.740 MinOB= 1 MaxOB= 13
N= 6750 Tiempo= 0.000000387259  MedioOb= 11.787 MinOB= 1 MaxOB= 13
N= 7000 Tiempo= 0.000000385857  MedioOb= 11.830 MinOB= 1 MaxOB= 13
N= 7250 Tiempo= 0.000000404000  MedioOb= 11.870 MinOB= 1 MaxOB= 13
N= 7500 Tiempo= 0.000000407733  MedioOb= 11.908 MinOB= 1 MaxOB= 13
N= 7750 Tiempo= 0.000000387355  MedioOb= 11.943 MinOB= 1 MaxOB= 13
N= 8000 Tiempo= 0.000000399250  MedioOb= 11.976 MinOB= 1 MaxOB= 13
N= 8250 Tiempo= 0.000000384485  MedioOb= 12.014 MinOB= 1 MaxOB= 14
N= 8500 Tiempo= 0.000000388353  MedioOb= 12.073 MinOB= 1 MaxOB= 14
N= 8750 Tiempo= 0.000000390286  MedioOb= 12.128 MinOB= 1 MaxOB= 14
N= 9000 Tiempo= 0.000000390111  MedioOb= 12.180 MinOB= 1 MaxOB= 14
N= 9250 Tiempo= 0.000000387027  MedioOb= 12.229 MinOB= 1 MaxOB= 14
N= 9500 Tiempo= 0.000000388526  MedioOb= 12.276 MinOB= 1 MaxOB= 14
N= 9750 Tiempo= 0.000000395487  MedioOb= 12.320 MinOB= 1 MaxOB= 14
N= 10000 Tiempo= 0.000000389100  MedioOb= 12.362 MinOB= 1 MaxOB= 14
```

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



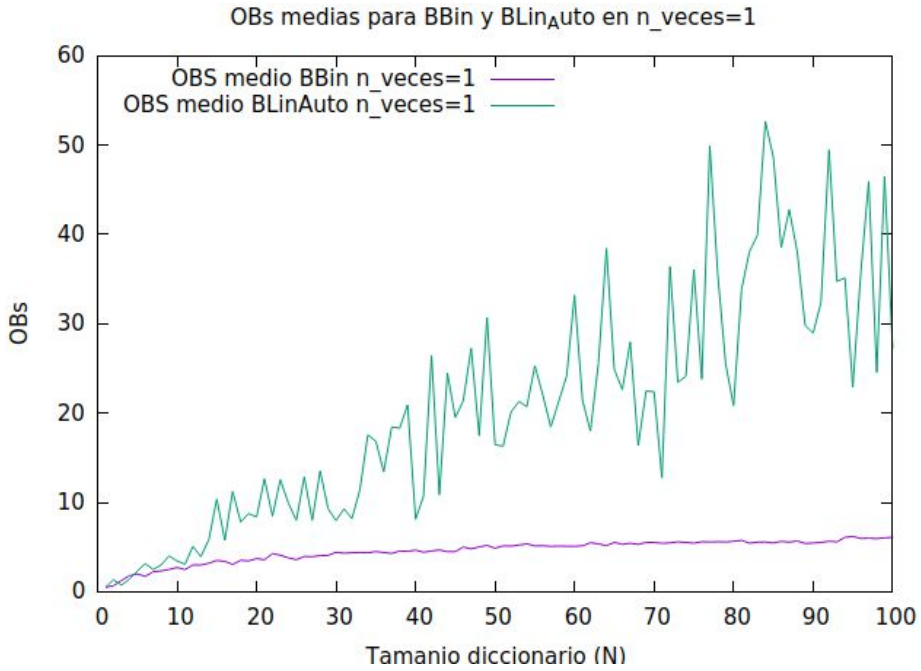
En esta gráfica vemos como el tiempo promedio de la Búsqueda Lineal es casi constante en valores muy bajos mientras que la Búsqueda Lineal aumenta considerablemente con tamaños de diccionarios altos.

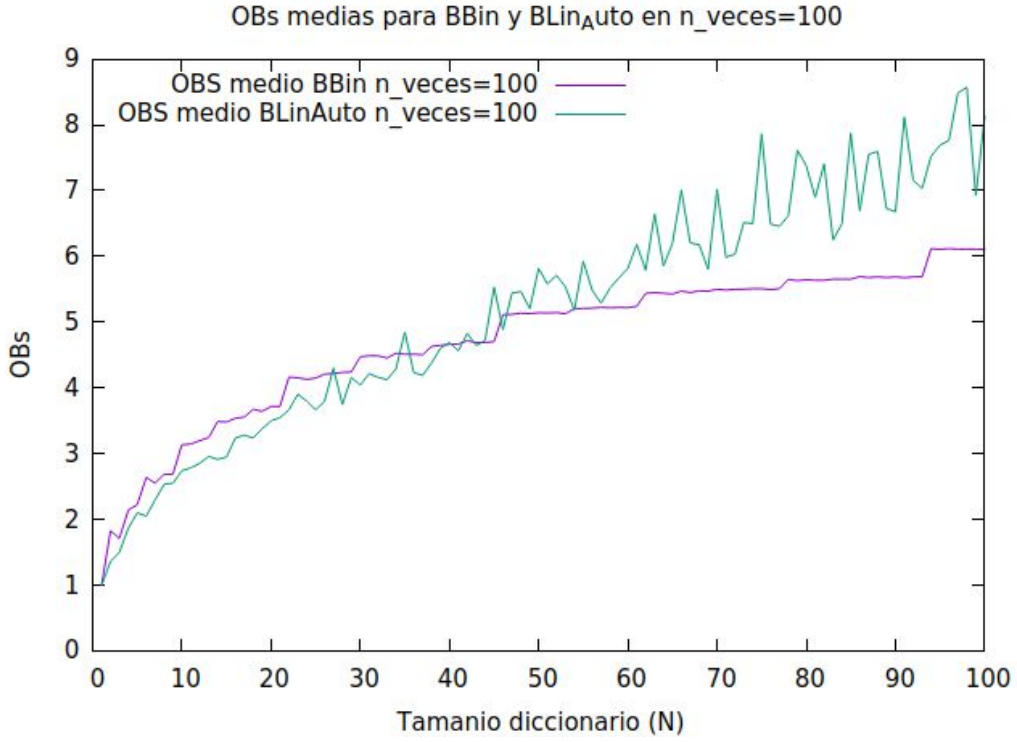
Gráfica comparando las OBs promedio entre la búsqueda lineal y la búsqueda binaria, comentarios a la gráfica.

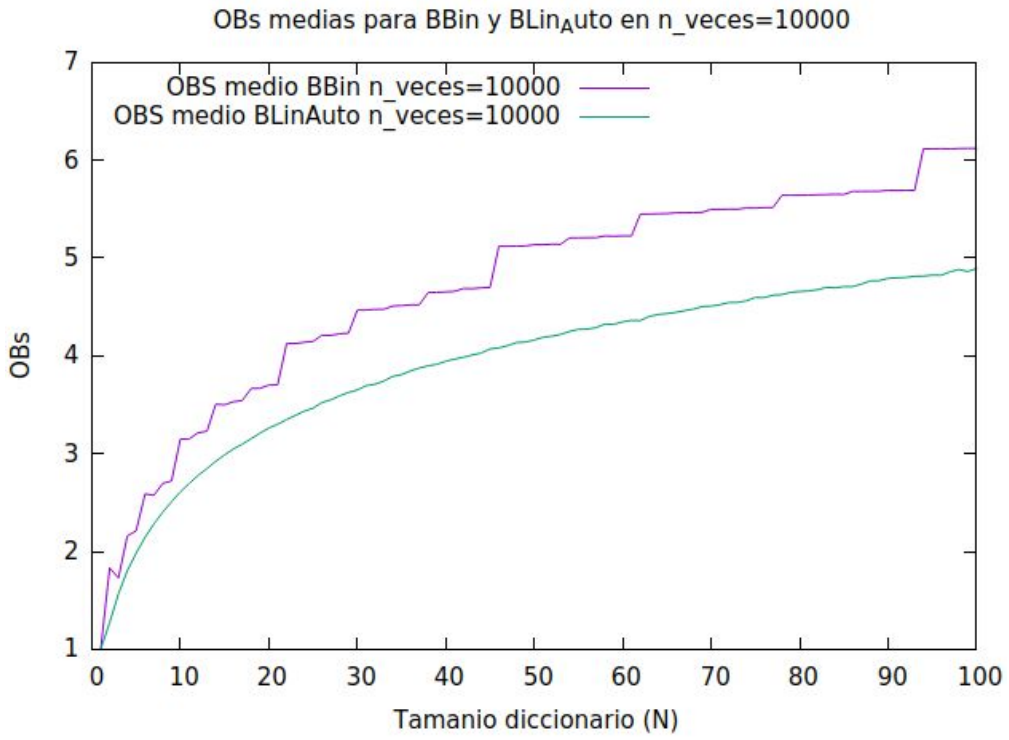


Aquí vemos como la Búsqueda Binaria es muy eficiente ya que el promedio de OBs que emplea en sus búsquedas es mucho menor que las que emplea el algoritmo Búsqueda Lineal que crece linealmente.

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.

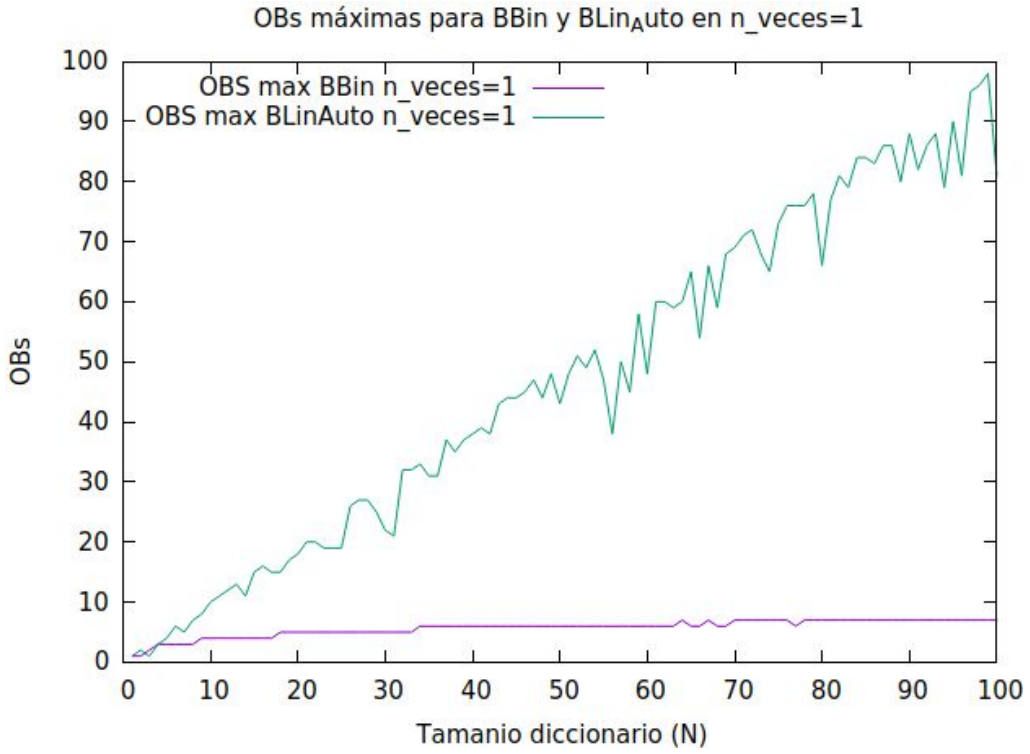


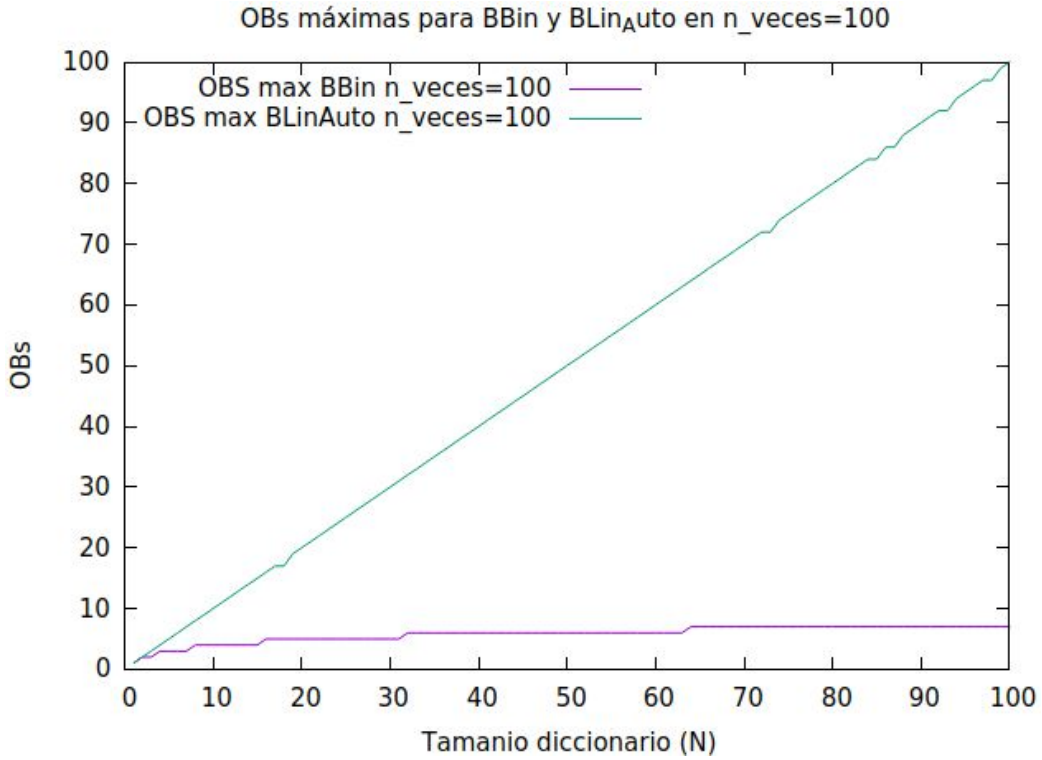


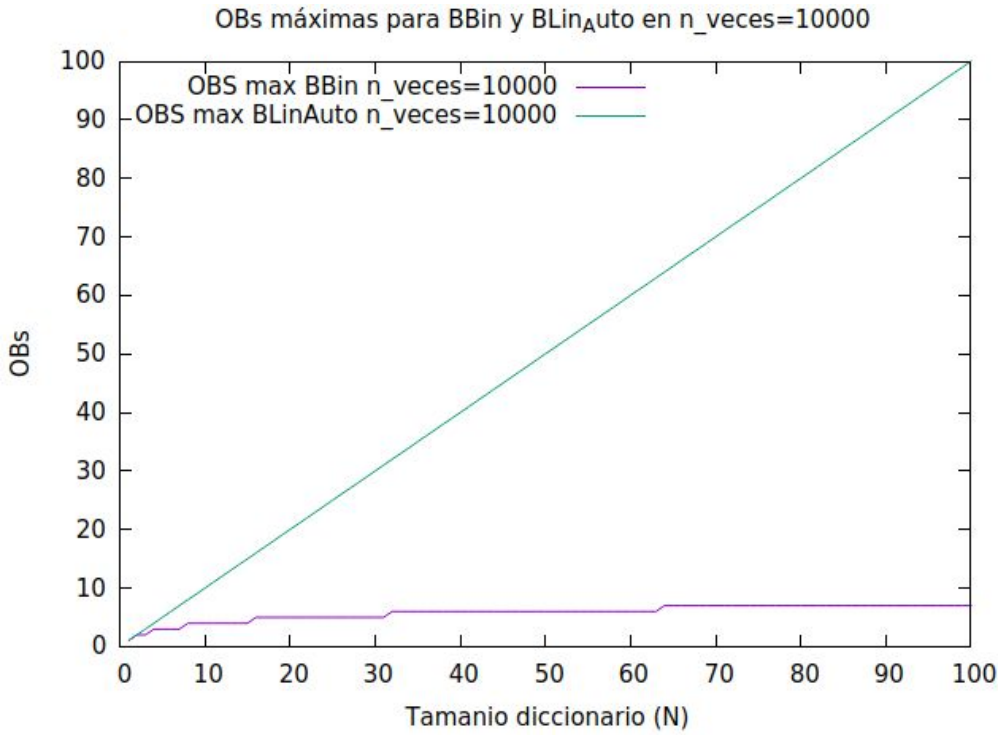


Podemos ver en estas gráficas como para n_veces=1 la BLin_auto no da valores estables y con valores de n_veces superiores se estabiliza sin lograr ser más eficaz que la BBin.

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), comentarios a la gráfica.

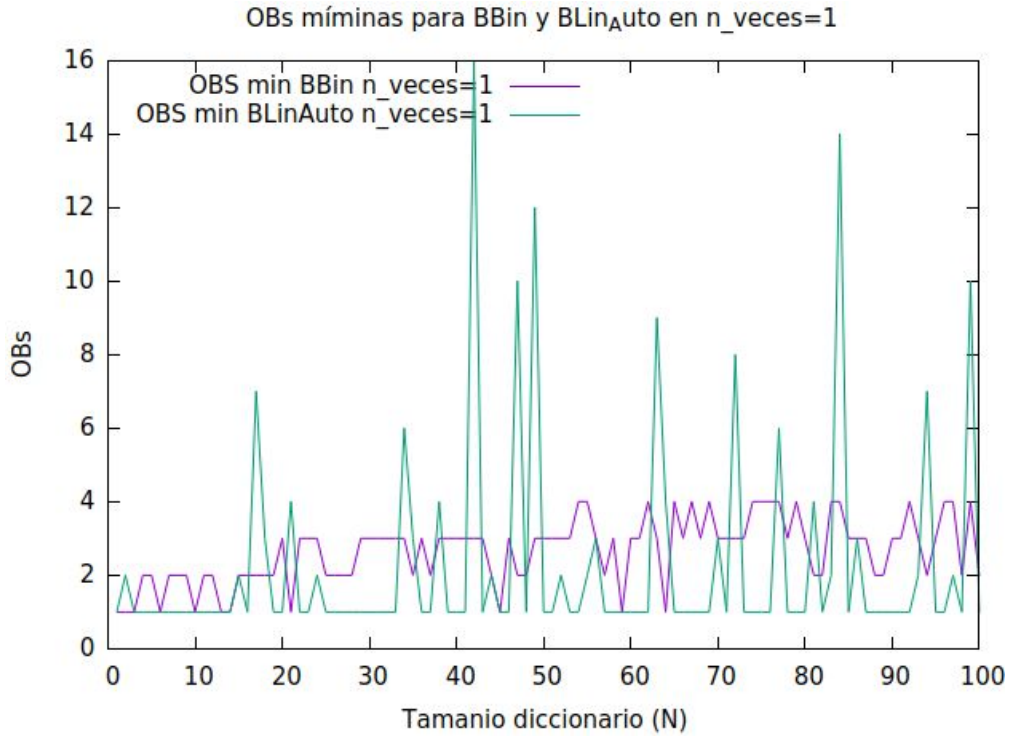


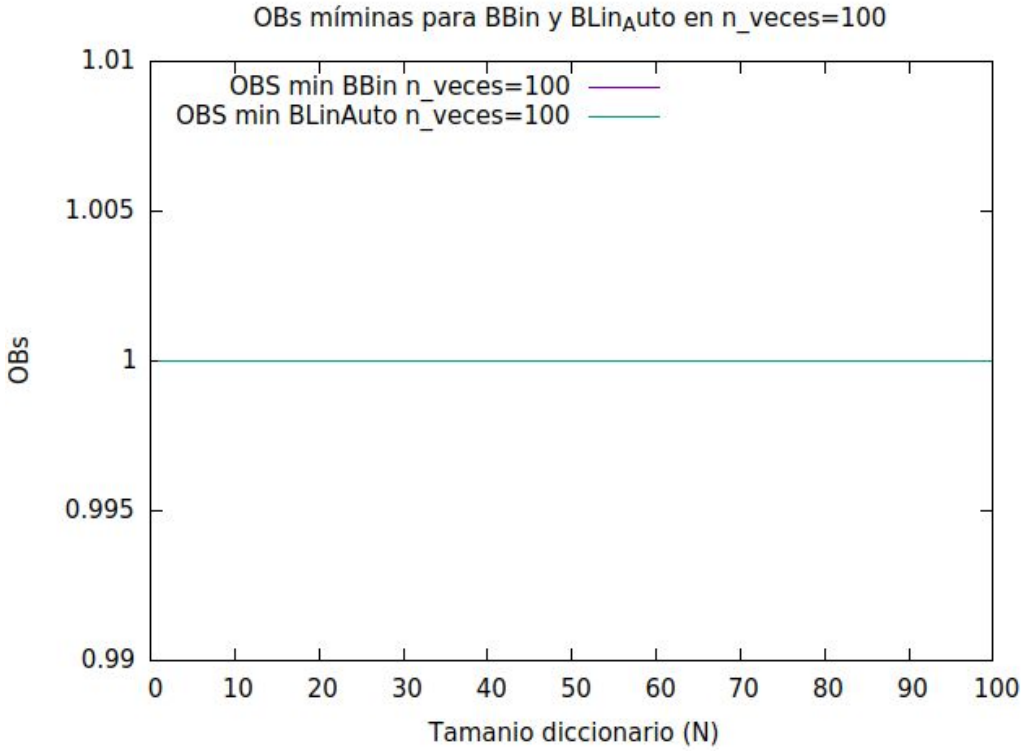


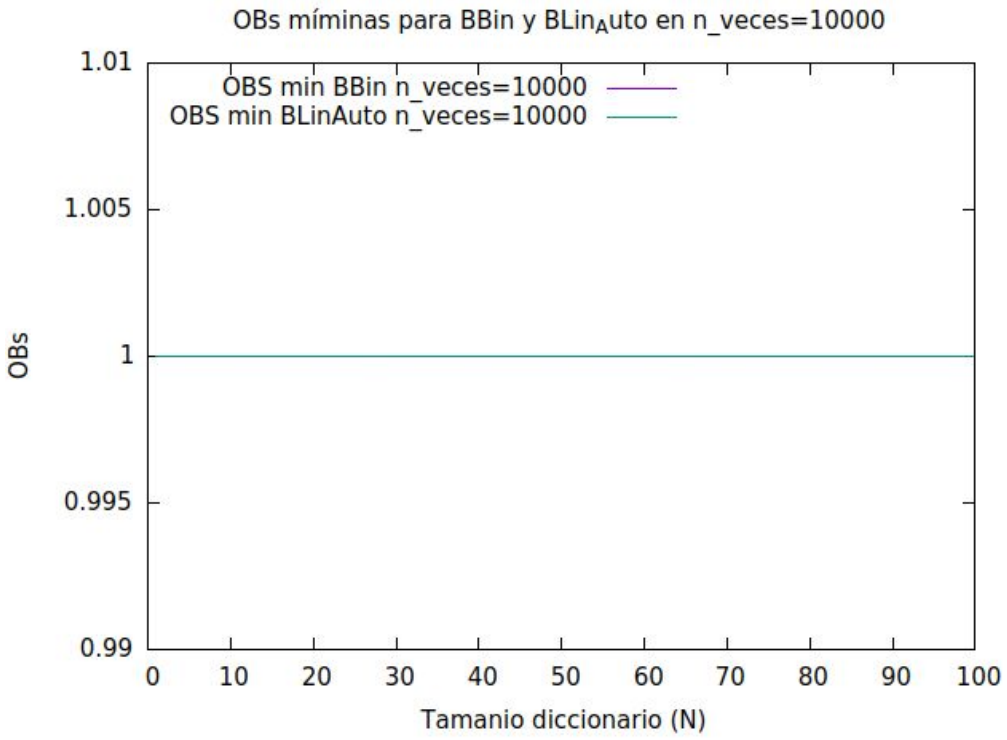


Vemos como las OBs máximas de la Búsqueda Binaria no sobrepasa las 7 OBs siendo considerablemente menor a la Búsqueda Lineal Autoorganizada que crece linealmente con pendiente igual a uno.

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), comentarios a la gráfica.

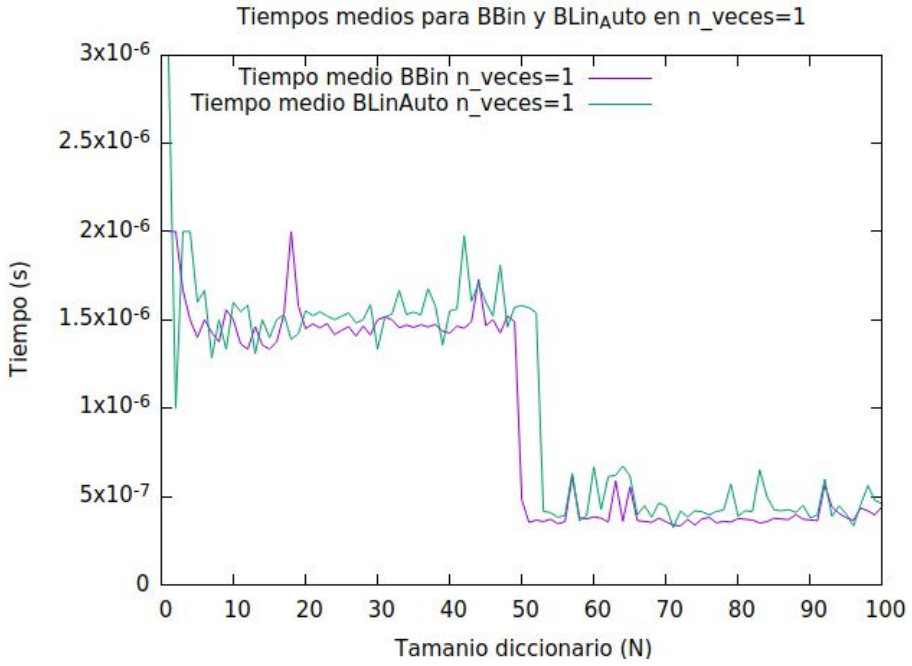


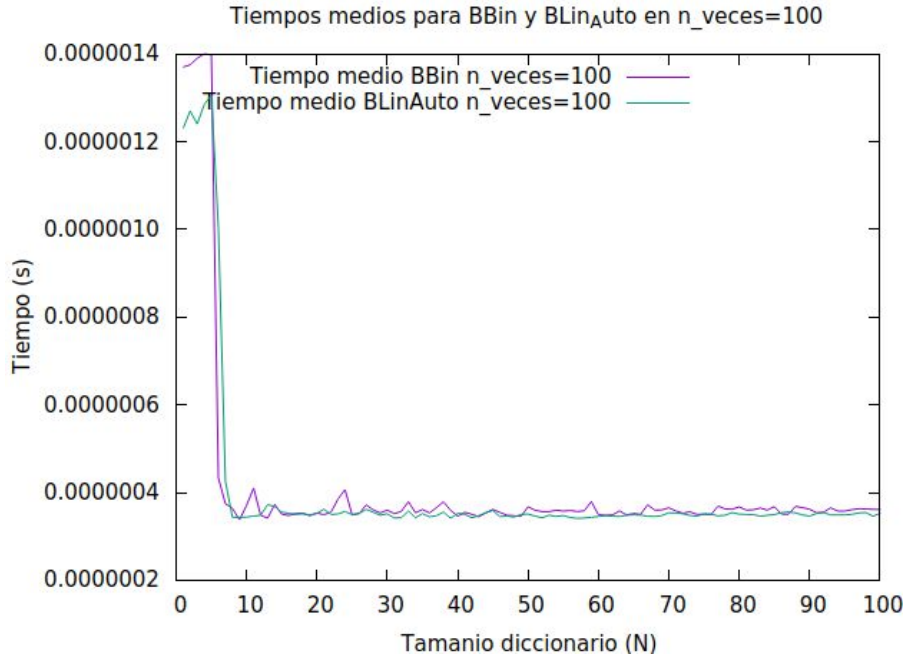


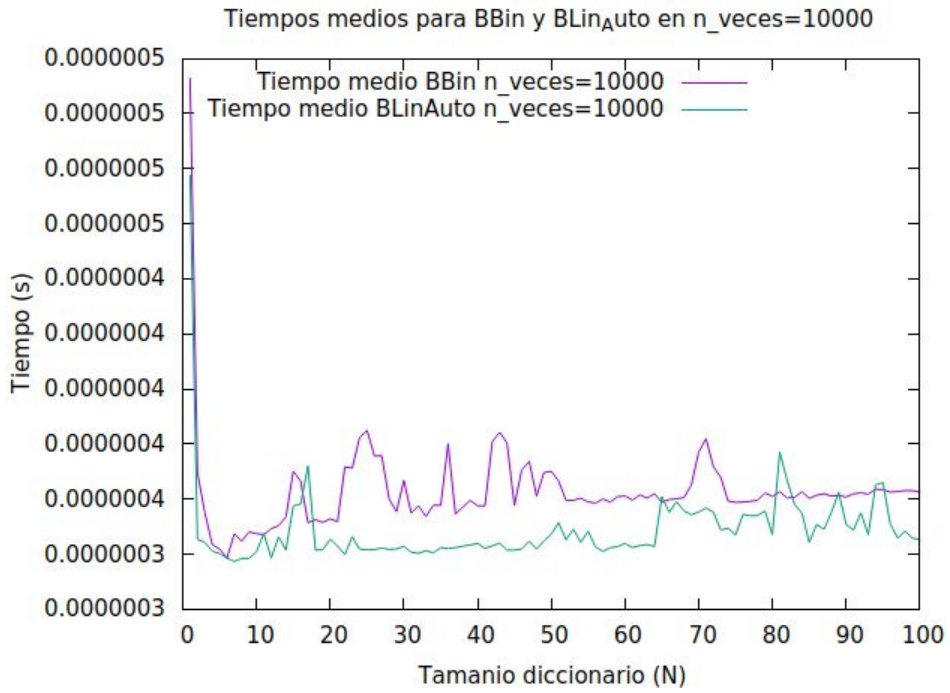


En estas gráficas observamos el número mínimo de OBs, que para n_veces=1 se aprecia datos inestables debido a que solo se ejecuta una sola vez por lo que se pueden dar casos en los que el mínimo no sea uno. Pero con n_veces tomando valores altos se estabiliza dando siempre valores mínimos de 1 OB.

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), comentarios a la gráfica.







En estas gráficas observamos que al tratarse de tiempos muy pequeños las gráficas no son estables aunque ambos algoritmos tienen tiempos promedios parecidos.

5. Respuesta a las preguntas teóricas.

5.1 Pregunta 1

La operación básica es la comparación de la clave que se pasa como argumento con los elementos de la tabla.

5.2 Pregunta 2

En cuanto a BBin el Bss(n) = 1 y el Wss(n)= log(n)+O(1).

En cuanto a Blin el Bss(n)=1 y el Wss(n)= N.

5.3 Pregunta 3

Con blin_auto la posición de los elementos que más veces se han buscado se mueven a las primeras posiciones con el paso de las búsquedas. Logrando así que el algoritmo realice menos comparaciones de clave para el elemento usualmente buscado.

5.4 Pregunta 4

Al ser el Blin_auto cuando ya se han realizado numerosas búsquedas, se espera que el elemento esté en las primeras posiciones por lo que se espera que el orden de ejecución medio sea Ablin(n)= O(1).

5.5 Pregunta 5

Bbin busca bien y es más efectivo porque primero se sitúa en la mitad de la tabla y compara la clave a buscar con el elemento de la mitad de la tabla, si no coincide, compara si es menor o mayor desplazándose al lado indicado por lo que se elimina la mitad de la tabla acotando enormemente la región de búsqueda hasta que la clave coincide con el elemento de la tabla comparado.

6. Conclusiones finales.

En esta práctica hemos implementado y analizado diferentes algoritmos de búsqueda. Llegamos a la conclusión de que si usamos tablas desordenadas la búsqueda binaria no se puede utilizar, por lo tanto son útiles tanto la búsqueda lineal como la búsqueda lineal autoorganizada. Pero es más efectiva BLin_auto cuando se utiliza el generador de claves potencial y si se utiliza el generador de claves uniforme el rendimiento es equivalente, por lo que el mejor algoritmo de búsqueda en este tipo de tablas es la Búsqueda Lineal Autoorganizada.

En cuanto a las tablas ordenadas se ve claramente que la Búsqueda Binaria es la más efectiva puesto que su caso peor es log(n) + O(1) mientras que las búsquedas lineales es N.