Análisis de Algoritmos 2017/2018

Práctica 1

Alejandro Santorum & David Cabornero - Pareja 10
Grupo 1201

Código	Gráficas	Memoria	Total

1. Introducción.

El trabajo consta de tres bloques, divididos en completar funciones dentro de permutaciones.c, ordenacion.c y tiempos.c para que nos funcionen los 5 ejercicios diseñados de antemano.

En permutaciones.c hemos creado tres funciones: una para generar números aleatorios, otra para permutar de forma aleatoria un array de números y otra para generar varias permutaciones.

En ordenacion.c, implementamos un algoritmo de ordenación, en este caso el método de la burbuja, tanto con flag como sin flag, aunque sin embargo se nos ha dicho que debemos utilizar en tiempos.c el algoritmo sin flag. En cualquier caso, dejamos la función indicada.

En tiempos.c, tenemos que implementar tres funciones. En la primera se nos pide generar permutaciones y guardar los tiempos que tardan en ser ordenadas, la segunda es la función principal debido a que es la única llamada desde el programa main y esta llama a las otras dos. La segunda se encarga para reservar la suficiente memoria para la/las estructura/estructuras PTIEMPO y de completar el programa llamando a las otras dos funciones de tiempos.c. La tercera función es la encarga de transportar los datos registrados en la estructura PTIEMPO a un fichero de texto.

2. Objetivos

2.1 Apartado 1

En el ejercicio 1, se nos pide elaborar una función de números aleatorios para nuestro programa. En la función implementada primero utilizábamos módulos para conseguir el número, pero este método da más probabilidad a los bits menos significativos.

2.2 Apartado 2

En este apartado se nos pide generar permutaciones aleatorias de números a partir de un array de números ordenados. Esta función tiene por lo tanto dos partes: generar el array ordenado y desordenar el array con la función random.

2.3 Apartado 3

Ahora nos piden generar un array de permutaciones, que no debería tener mayor complicación. Nos insisten mucho en que las permutaciones deben ser equiprobables, así que la mayor complicación exigida en el ejercicio 3 es en mejorar el algoritmo de aleatoriedad del ejercicio 1.

2.4 Apartado 4

Ahora debemos implementar el método de la burbuja, donde introduciendo el número menor de la tabla, el mayor y el array desordenado que queramos ordenar. Recordemos en qué consiste el método de la burbuja: comparamos números de dos a dos desde el principio de la tabla hasta el final, y en caso de que el mayor se encuentre antes que el menor en alguna comparación, se hará un swap. De esta forma, cada vez que recorramos el array se ordenará el último elemento, por lo que en la siguiente vuelta tendremos que hacer una comparación menos. También tenemos que devolver el número de veces que se ha realizado la operación básica. Además, hemos incluido la versión con flags, que termina el bucle si en alguna vuelta no se hace ningún swap. Sin embargo, no hemos utilizado esta función porque así se nos ha indicado.

2.5 Apartado 5

En este apartado se encuentra el grueso del ejercicio. En el primero, le daremos al programa en número de arrays de números desordenados que queremos y el tamaño de dichos arrays, y tendremos que rellenar PTIEMPO, una estructura en la que tendremos que informar de distintos aspectos relativos a el tiempo de ordenación de BubbleSort: el tamaño del array, el número de arrays, el tiempo que tarda en ordenar de media cada array, el número medio de veces que se ejecuta la OB y el máximo y el mínimo de veces que se ejecuta la OB en una ordenación de array.

En el tercero, se nos dará un PTIEMPO rellenado, un fichero y el número de estructuras que tiene PTIEMPO. Nuestro trabajo consistirá en imprimir las estructuras en el fichero indicado.

La segunda función trata de juntar las dos anteriores. Debemos imprimir en un fichero todos los tiempos de PTIEMPO, donde se da un algoritmo de ordenación y ahora, en vez de generar una lista con números consecutivos, la crearemos sabiendo el número máximo, el mínimo y la diferencia que habrá entre los números de la tabla.

3. Herramientas y metodología

Hemos utilizado el sistema operativo Linux, dentro del cual hemos trabajado en el entorno de cloud9, en https://ide.c9.io

3.1 Apartado 1

Debemos devolver un número al azar comprendido entre dos enteros. Hay dos comprobaciones, una para agilizar y otra necesaria. La que pretende agilizar el proceso es el caso concreto de que el mínimo y el máximo sean el mismo número, y en tal caso devolveremos ese número. La otra es el caso de que el máximo sea menor que el mínimo; en tal caso, hemos decidido devolver un error.

Como hemos dicho antes, seleccionar el número aleatorio por módulos no es una buena forma de hacer una función aleatoria, ya que hay una mayor probabilidad de que

salgan los números más pequeños. Para rectificar este error, hemos utilizado el algoritmo de "Numerical recipes in C: the art of scientific computing", capítulo 7.

3.2 Apartado 2

Debemos crear un array de enteros desordenados, para ellos se nos da un solo dato, el tamaño del array. Debemos de tomar dos medidas: la primera es obvia: el tamaño no debe ser menor que cero. Como tampoco puede ser excesivamente grande, también hemos implementado una medida que impida números de cifras desorbitadas.

En esta función primero reservamos memoria para el array, después creamos una lista ordenada de números y por último, realizamos un swap entre cada posición i y una posición aleatoria j, cuya única condición es que j sea mayor que i.

3.3 Apartado 3

En este ejercicio crearemos una lista de arrays de números permutados, y se nos darán dos datos: el tamaño del array y el número de permutaciones que se desean. Realizamos las dos medidas realizadas con anterioridad, con la diferencia de que ahora debemos asegurarnos de que dos datos se encuentran entre cero y un valor máximo establecido.

Reservamos memoria para un doble puntero a int, pp, y creamos un bucle for en el que vayamos guardando distintas permutaciones dentro de pp, todo utilizando dentro del bucle la función creada en el apartado anterior para crear permutaciones aleatorias.

3.4 Apartado 4

Como se nos pidió el BubbleSort sin flags, solo explicaremos este, aunque dejaremos indicado el otro en el código. Recibimos tres datos: el array de enteros desordenado, el menor número del array y mayor. En este caso hemos realizado tres comprobaciones: el array no puede ser NULL, el mínimo no puede ser mayor que el máximo (hemos decidido devolver error, ya que hemos considerado que es más probable que el usuario se haya equivocado al introducir los números a que haya metido los indicados al revés) y uno en el que, si la tabla tiene un elemento (mínimo igual al máximo) entonces se devolverá la tabla sin tocarla y el contador de iteraciones de la operación básica a 0.

Una vez hecho esto, aplicamos un doble bucle for, donde en el bucle más externo se actualiza la parte de la lista que no está ordenada, mientras que en el bucla más interno se va recorriendo la lista desordenada, de tal forma que si cierto elemento es mayor que el siguiente realizamos un swap. Como la operación básica del BubbleSort es la comparación de claves, cada vez que realizamos una de las comprobaciones sumamos al contador de comparaciones 1.

3.5 Apartado 5

La **primera función**, tiempo_medio_ordenacion, nos da cuatro argumentos: una función de ordenación, n_perms, N y *ptiempo. Nuestras primeras comprobaciones consisten en comprobar que los dos punteros no son NULL. Después tenemos que

comprobar que n_perms y N son mayores que 0. Hemos preferido incluir estas comprobaciones en este apartado, aunque se supone que las funciones que vamos a usar ya deberían encargarse de ello.

Hay una parte muy sencilla de la estructura: tanto el tamaño de la entrada como los elementos promediados son N y n_perms, respectivamente. Después, hacemos un bucle donde irá actuando el algoritmo de ordenación sobre las distintas permutaciones. Para calcular el tiempo promedio de una ordenación, medimos el tiempo de cada ordenación y los vamos sumando, para después hacer la media. Lo mismo haremos con las OB medias, iremos sumando los valores que nos devuelve la función para luego hacer la media. El mínimo y el máximo de OB realizadas en una ordenación también se va realizando dentro del bucle, solo hay que actualizarlo cada vez que el mínimo o máximo actual supere al anterior.

Empezaremos explicando la **tercera función**, ya que la segunda requiere de ésta. En primer lugar, comprobaremos que el fichero y el puntero PTIEMPO no sean NULL, además de que el número de estructuras debe ser mayor que 0. La función es simple, abriremos el fichero, imprimimos en una primera fila los parámetros a indicar, y en las siguientes vamos imprimento las estructuras dentro de un bucle for. Para finalizar, cerramos el fichero.

Para finalizar, explicaremos la **segunda función**. Hacemos las comprobaciones necesarias: los dos punteros no deben ser NULL, el número de permutaciones será como mínimo 0, el máximo debe ser mayor que mínimo y el inremento debe ser mayor que 0.

Primero, calculamos el número de tamaños distintos que vamos a tener a través del máximo, el mínimo y el incremento. Después, creamos el puntero a estructura PTIEMPO una vez sabido su tamaño gracias a el número obtenido. Por último, con ayuda de un bucle for y la primera función creada, vamos rellenando los datos de PTIEMPO. Finalmente, escribimos los datos de PTIEMPO en el fichero con la tercera función y liberamos PTIEMPO, ya que lo hemos creado nosotros y no tenemos que devolverlo.

4. Código fuente

4.1 Apartado 1

```c
/************************************************/
/* Funcion: aleat_num Fecha: 22/09/2017         */
/* Autores: David Cabornero Pascual             */
/*          Alejandro Santorum Varela           */
/* Rutina que genera un numero aleatorio        */
/* entre dos numeros dados                      */
/*                                              */
/* Entrada:                                     */
/* int inf: limite inferior                     */
/* int sup: limite superior                     */
/* Salida:                                      */
/* int: numero aleatorio                        */
/************************************************/
int aleat_num(int inf, int sup){
    int result = 0;

    if(inf == sup){
        return sup;
    }

    else if (inf > sup){
        printf("ERROR: Límite inferior mayor que el límite superior.\n");
        exit(-1);
    }

    result = (inf + ((int) ((((double)(sup-inf+1)) * rand())/(RAND_MAX + 1.0))));

    return result;
}
```

Histograma(código)

```c
    num = atoi(argv[++i]);
  } else {
    fprintf(stderr, "Parametro %s es incorrecto\n", argv[i]);
  }
}

/* imprimimos los datos */
histograma = (int*) malloc ((sup-inf+1)*sizeof(int));
if(histograma == NULL){
  printf("Error de memoria en la creación del histograma.\n"); /* Error */
  return -1;
}

for(i = 0; i<(sup-inf+1);i++){
  histograma[i]=0;
}

for(j = 0; j < num; j++) {
  aux = aleat_num(inf, sup);
  printf("%d\n", aux);
  for(k=inf; k<=sup; k++){
    if(k == aux){
      histograma[k-inf]++;
    }
  }
}

for(i = inf; i<=sup;i++){
    printf("Frecuencia del %d: %d/%d\n", i, histograma[i-inf], num);
    printf("Porcentaje del %d: %.2f%%\n\n", i, (((double)histograma[i-inf]/num)*100));
}

out = (FILE *) fopen("data", "w");
if(out == NULL){
  free(histograma);
  return -1;
}

for(i=inf; i<=sup; i++){
  fprintf(out, "%d   %d\n", i, histograma[i-inf]);
}

fclose(out);

free(histograma);

return 0;
}
```

4.2 Apartado 2

```c
/* Funcion: genera_perm Fecha: 22/09/2017          */
/* Autores: David Cabornero Pascual                */
/*          Alejandro Santorum Varela              */
/* Rutina que genera una permutacion               */
/* aleatoria                                       */
/*                                                 */
/* Entrada:                                        */
/* int n: Numero de elementos de la                */
/* permutacion                                     */
/* Salida:                                         */
/* int *: puntero a un array de enteros            */
/* que contiene a la permutacion                   */
/* o NULL en caso de error                         */
/*************************************************/
int* genera_perm(int N){
    int i, aux1, aux2;
    int *perm;

    if(N<=0){
        printf("ERROR: Tamanio del array de la permutación menor o igual a cero.\n");
        return NULL;
    }
    else if(N>MAX_TAM){
        printf("MEDIDA DE SEGURIDAD: Tamanio del array de la permutación excesivo (superior a INT_MAX).\n");
        return NULL;
    }

    perm = (int  *) malloc(N * sizeof(int));
    if(perm == NULL){
        return NULL;
    }

    for(i=0; i<N; i++){
        perm[i] = i;
    }

    for(i=0; i<N; i++){
        aux1 = perm[i];
        aux2 = aleat_num(i, N-1);
        perm[i] = perm[aux2];
        perm[aux2] = aux1;
    }

    return perm;
}
```

4.3 Apartado 3

```c
/* Funcion: genera_permutaciones Fecha: 22/09/2017 */
/* Autores: David Cabornero Pascual                */
/*          Alejandro Santorum Varela              */
/*                                                 */
/* Funcion que genera n_perms permutaciones        */
/* aleatorias de tamanio elementos                 */
/*                                                 */
/* Entrada:                                        */
/* int n_perms: Numero de permutaciones            */
/* int N: Numero de elementos de cada              */
/* permutacion                                     */
/* Salida:                                         */
/* int**: Array de punteros a enteros              */
/* que apuntan a cada una de las                   */
/* permutaciones                                   */
/* NULL en caso de error                           */
/*************************************************/
int** genera_permutaciones(int n_perms, int N){
    int i,j;
    int **pp=NULL;

    if(N<=0 || n_perms<=0){
        printf("ERROR: El número de permutaciones y/o el tamanio de las mismas menor o igual que cero.\n");
        return NULL;
    }
    else if(N>MAX_TAM||n_perms>MAX_TAM){
        printf("MEDIDA DE SEGURIDAD: El número de permutaciones y/o el tamanio de las mismas es demasiado grande.\n");
    }

    pp = (int **) malloc(n_perms * sizeof(int*));
    if(pp == NULL){
        return NULL;
    }

    for(i=0; i<n_perms; i++){
        pp[i] = genera_perm(N);
        if(pp[i] == NULL){
            for(j=0; j<i; j++){
                free(pp[j]);
            }
            free(pp);
            return NULL;
        }
    }
    return pp;
}
```

4.4 Apartado 4

```c
/**************************************************/
/* Funcion: InsertSort    Fecha: 22/09/2017       */
/*                                                */
/*Descripción: Rutina que ordena los elementos    */
/* de una tabla                                   */
/*                                                */
/* Entrada:                                       */
/* tabla: array de enteros a ordenar              */
/* iu: límite superior                            */
/* ip: límite inferior                            */
/*                                                */
/* Salida:                                        */
/* int: OK si ha sido un éxito la rutina          */
/*  o ERR si ha habido algún fallo/problema       */
/**************************************************/
int BubbleSort(int *tabla, int ip, int iu){
    int i, j,aux, cont=0;

    if(tabla == NULL){ /* Error */
        return ERR;
    }

    if(ip == iu){
        return cont;
    }

    if(ip > iu){
        printf("ERROR: índice primero superior a índice último en la tabla a ordenar. Compruebe los parámetros de entrada.\n");
        return ERR;
    }

    for(i=iu; (i>(ip+1)); i--){
        for(j=ip; j<i; j++){
            cont++;
            if(tabla[j]>tabla[j+1]){        /* Si el elemento siguiente es menor que el actual, realizamos un "swap" */
                aux = tabla[j];
                tabla[j] = tabla[j+1];
                tabla[j+1] = aux;
            }
        }
    }
    return cont;
}
```



```c
int BubbleSort_flag(int *tabla, int ip, int iu){
    int i, j,aux, flag=1, cont=0;

    if(tabla == NULL){ /* Error */
        return ERR;
    }

    if(ip == iu){
        return cont;
    }

    if(ip > iu){
        printf("ERROR: índice primero superior a índice último en la tabla a ordenar. Compruebe los parámetros de entrada.\n");
        return ERR;
    }


    for(i=iu; (flag == 1) && (i>(ip+1)); i--){
        flag = 0;
        for(j=ip; j<i; j++){
            cont++;
            if(tabla[j]>tabla[j+1]){        /* Si el elemento siguiente es menor que el actual, realizamos un "swap" */
                aux = tabla[j];
                tabla[j] = tabla[j+1];
                tabla[j+1] = aux;
                flag = 1;
            }
        }
    }

    return cont;
}
```

4.5 Apartado 5

```c
/*****************************************************/
/* Funcion: tiempo_medio_ordenación     Fecha: 30/09/2017 */
/* Autores: David Cabornero Pascual                    */
/*          Alejandro Santorum Varela                  */
/*                                                     */
/* Entrada:                                            */
/* int n_perms: Numero de permutaciones                */
/* int N: Numero de elementos de cada                  */
/* ptiempo: puntero a estructura Tiempo                */
/* permutacion                                         */
/* metodo: puntero a función de ordenación             */
/* Salida:                                             */
/* short: OK en caso de éxito, ERR en caso de ERROR    */
/*****************************************************/
short tiempo_medio_ordenacion(pfunc_ordena metodo, int n_perms, int N, TIEMPO *ptiempo){
  int **perms;
  int i, max=0, min=INT_MAX;
  double media = 0, actual, total=0;
  clock_t start, end;

  /*Comprobaciones de error-------------------------------------------------------*/
  if(metodo == NULL){
    printf("Error. Puntero a función nulo.\n");
    return ERR;
  }
  if(ptiempo == NULL){
    printf("Error. Puntero a la estructura tiempo nulo.\n");
    return ERR;
  }
  if(n_perms<=0){
    printf("Error. Número de permutaciones nulo o negativo.\n");
    return ERR;
  }
  if(N<=0){
    printf("Error. Tamaño de las permutaciones negativo o nulo.\n");
    return ERR;
  }

  /*El grueso del programa--------------------------------------------------------*/
  ptiempo->N = N;
  ptiempo->n_elems = n_perms;

  perms = genera_permutaciones(n_perms, N);
  if(perms == NULL){
    printf("Error en la función tiempo_medio_ordenacion.\n");
    return ERR;
  }

  for(i=0; i<n_perms; i++){
    start = clock();
    actual = metodo(perms[i], 0, N-1);
    end = clock();

    total = total + (end - start);
    media = media + actual;

    if(actual > max){
      max = actual;
    }
    if(actual < min){
      min = actual;
    }
  }

  total = (total/(double)CLOCKS_PER_SEC)/n_perms;
  total = total*1000; /* Pasamos el tiempo a milisegundos */

  media = media/n_perms;

  ptiempo->medio_ob = media;
  ptiempo->max_ob = max;
  ptiempo->min_ob = min;
  ptiempo->tiempo = total;

  for(i=0; i<n_perms; i++){
    free(perms[i]);
  }
  free(perms);

  return OK;
}
```

```c
/***********************************************************/
/* Funcion: genera_tiempos_ordenacion    Fecha: 30/09/2017*/
/* Autores: David Cabornero Pascual                       */
/*          Alejandro Santorum Varela                     */
/*                                                        */
/* Entrada:                                               */
/* int n_perms: Numero de permutaciones                   */
/* int num_min: tamaño mínimo permutación                 */
/* int num_max: tamaño máximo de permutación              */
/* permutacion                                            */
/* int incr: incremento del tamaño.                       */
/* metodo: puntero a función de ordenación                */
/* fichero: puntero a un fichero donde se imprimirán datos*/
/* Salida:                                                */
/* short: OK en caso de éxito, ERR en caso de ERROR       */
/***********************************************************/
short genera_tiempos_ordenacion(pfunc_ordena metodo, char *fichero, int num_min, int num_max, int incr, int n_perms){
  TIEMPO *ptiempo;
  int array_size, i, j;
  double aux;

  /*Comprobaciones*/
  if(num_min<0 || num_max < num_min || incr <= 0){
    printf("Error. Mete unos números decentes por favor.\n");
    return ERR;
  }
  if(n_perms<=0){
    printf("Error. Número de permutaciones nulo o negativo.\n");
    return ERR;
  }
  if(fichero == NULL){
    printf("Error. Puntero a fichero nulo.\n");
    return ERR;
  }
  if(metodo == NULL){
    printf("Error. Puntero a función nulo.\n");
    return ERR;
  }
  /*-----------------------------------------*/
```

```c
  /*Grueso del programa*/
  aux = num_max - num_min;
  aux = aux/incr; /* ya nos hemos asegurado anteriormente que incr != 0 */
  array_size = ceil(aux);
  printf("array = %d\n", array_size);
  /*El anterior printf, intuitivamente, te dice el número
  de filas que se van a imprimir en el fichero de salida*/

  ptiempo = (TIEMPO *) malloc(array_size * sizeof(TIEMPO));
  if(ptiempo == NULL){
    printf("Error. Reserva de memoria en PTIEMPO.\n");
    return ERR;
  }

  for(i=num_min, j=0; j<array_size; i = i + incr, j++){
    if(tiempo_medio_ordenacion(metodo, n_perms, i, &ptiempo[j]) == ERR){
      printf("Error en la función 2 haciendo la 1. del ej5.\n");
      return ERR;
    }
  }


  if(guarda_tabla_tiempos(fichero, ptiempo, array_size) == ERR){
    return ERR;
  }


  free(ptiempo);

  return OK;
}
```

```c
/*****************************************************/
/* Funcion: guarda_tabla_tiempos         Fecha: 30/09/2017*/
/* Autores: David Cabornero Pascual                    */
/*          Alejandro Santorum Varela                  */
/*                                                     */
/* Entrada:                                            */
/* int n_tiempos: tamaño del array                     */
/* fichero: puntero a un fichero donde se imprimirán datos*/
/* ptiempo: array de punteros a estructura tiempo          */
/*                                                     */
/* Salida:                                             */
/* short: OK en caso de éxito, ERR en caso de ERROR       */
/*****************************************************/
short guarda_tabla_tiempos(char *fichero, TIEMPO *ptiempo, int n_tiempos){
  FILE *f=NULL;
  int i;

  /*Comprobaciones*/
  if(fichero == NULL){
    printf("Error. Puntero a fichero nulo.\n");
    return ERR;
  }
  if(ptiempo == NULL){
    printf("Error. Puntero a la estructura tiempo nulo.\n");
    return ERR;
  }
  if(n_tiempos <= 0){
    printf("Error. Tamaño del array tiempo negativo o nulo.\n");
    return ERR;
  }


  /*Grueso del programa*/
  f = (FILE *) fopen(fichero, "a");
  if(f == NULL){
    return ERR;
  }

  fprintf(f, "size  time(ms)  avg_ob  max_ob  min_ob.\n");

  for(i=0; i<n_tiempos; i++){
    fprintf(f, "%d   %f   %.2f   %d   %d\n", ptiempo[i].N, ptiempo[i].tiempo, ptiempo[i].medio_ob, ptiempo[i].max_ob, ptiempo[i].min_ob);
  }


  fclose(f);

  return OK;
}
```


144:26

5. Resultados, Gráficas

5.1 Apartado 1

```
santorum:~/workspace (master) $ valgrind ./ejercicio1 -limInf 1 -limSup 100 -numN 10
==1852== Memcheck, a memory error detector
==1852== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==1852== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==1852== Command: ./ejercicio1 -limInf 1 -limSup 100 -numN 10
==1852==
Practica numero 1, apartado 1
Realizada por: Alejandro Santorum & David Cabornero
Pareja 10
Grupo: 1201
32
92
93
74
79
49
67
75
9
98
==1852==
==1852== HEAP SUMMARY:
==1852==     in use at exit: 0 bytes in 0 blocks
==1852==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==1852==
==1852== All heap blocks were freed -- no leaks are possible
==1852==
==1852== For counts of detected and suppressed errors, rerun with: -v
==1852== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

```
santorum:~/workspace (master) $ valgrind ./ejercicio1 -limInf 1 -limSup 10 -numN 20
==1857== Memcheck, a memory error detector
==1857== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==1857== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==1857== Command: ./ejercicio1 -limInf 1 -limSup 10 -numN 20
==1857==
Practica numero 1, apartado 1
Realizada por: Alejandro Santorum & David Cabornero
Pareja 10
Grupo: 1201
1
6
2
1
2
2
1
1
6
8
4
9
4
3
5
1
4
3
9
10
==1857==
==1857== HEAP SUMMARY:
==1857==     in use at exit: 0 bytes in 0 blocks
==1857==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==1857==
==1857== All heap blocks were freed -- no leaks are possible
==1857==
==1857== For counts of detected and suppressed errors, rerun with: -v
==1857== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Ejercicio 1,Histograma

```
santorum:~/workspace (master) $ valgrind ./ejercicio1b -limInf 1 -limSup 10 -numN 20
==1865== Memcheck, a memory error detector
==1865== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==1865== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==1865== Command: ./ejercicio1b -limInf 1 -limSup 10 -numN 20
==1865==
Practica numero 1, apartado 1.b (HISTOGRAMA)
Realizada por: Alejandro Santorum & David Cabornero
Grupo: 1201
9
8
6
9
7
4
2
6
3
3
9
7
8
8
6
8
4
10
1
3
```

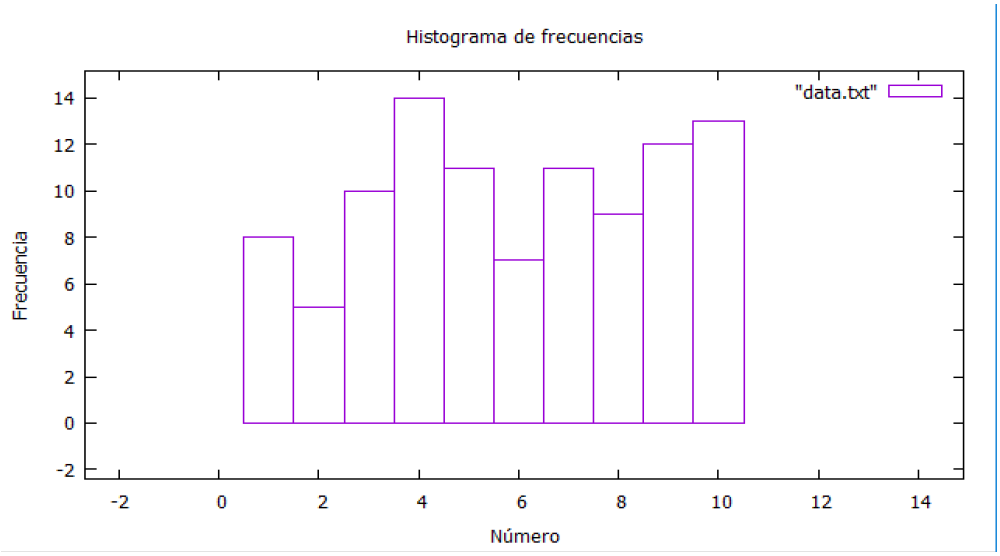
```
Frecuencia del 1: 1/20
Porcentaje del 1: 5.00%

Frecuencia del 2: 1/20
Porcentaje del 2: 5.00%

Frecuencia del 3: 3/20
Porcentaje del 3: 15.00%

Frecuencia del 4: 2/20
Porcentaje del 4: 10.00%

Frecuencia del 5: 0/20
Porcentaje del 5: 0.00%

Frecuencia del 6: 3/20
Porcentaje del 6: 15.00%

Frecuencia del 7: 2/20
Porcentaje del 7: 10.00%

Frecuencia del 8: 4/20
Porcentaje del 8: 20.00%

Frecuencia del 9: 3/20
Porcentaje del 9: 15.00%

Frecuencia del 10: 1/20
Porcentaje del 10: 5.00%

==1865==
==1865== HEAP SUMMARY:
```

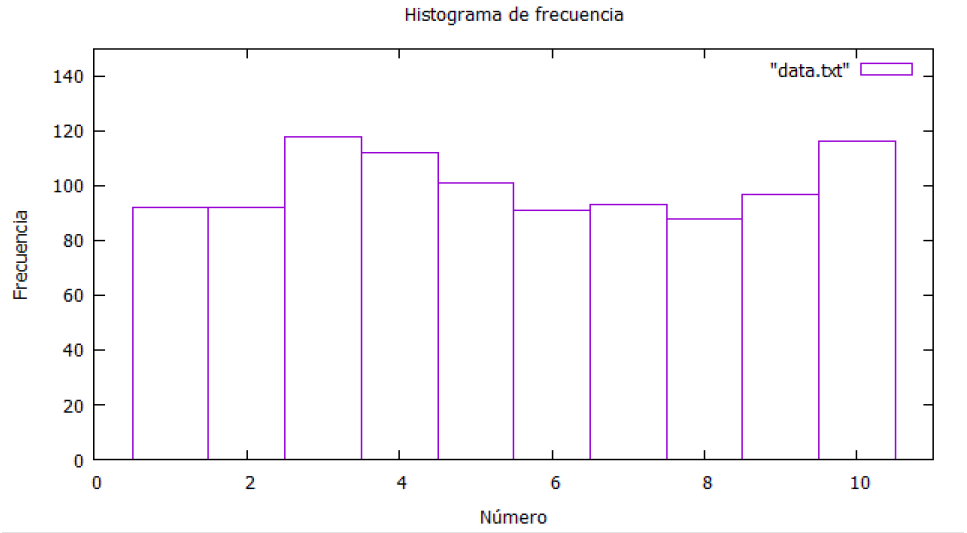
```
==1865==
==1865== HEAP SUMMARY:
==1865==     in use at exit: 0 bytes in 0 blocks
==1865==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==1865==
==1865== All heap blocks were freed -- no leaks are possible
==1865==
==1865== For counts of detected and suppressed errors, rerun with: -v
==1865== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

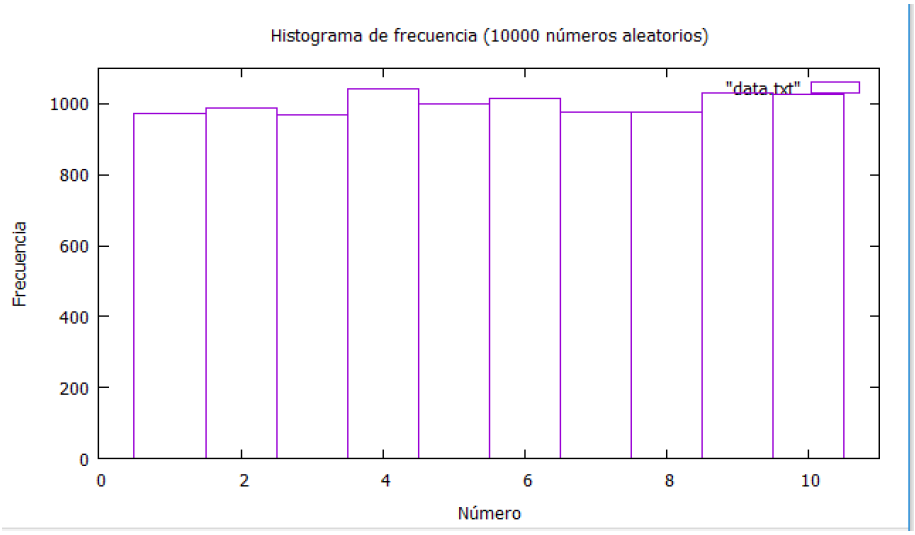
Ahora probamos con números altos, como 2000 o 20000

```
Frecuencia del 1: 209/2000
Porcentaje del 1: 10.45%

Frecuencia del 2: 203/2000
Porcentaje del 2: 10.15%

Frecuencia del 3: 190/2000
Porcentaje del 3: 9.50%

Frecuencia del 4: 218/2000
Porcentaje del 4: 10.90%

Frecuencia del 5: 217/2000
Porcentaje del 5: 10.85%

Frecuencia del 6: 191/2000
Porcentaje del 6: 9.55%

Frecuencia del 7: 203/2000
Porcentaje del 7: 10.15%

Frecuencia del 8: 198/2000
Porcentaje del 8: 9.90%

Frecuencia del 9: 180/2000
Porcentaje del 9: 9.00%

Frecuencia del 10: 191/2000
Porcentaje del 10: 9.55%

==1888==
==1888== HEAP SUMMARY:
==1888==     in use at exit: 0 bytes in 0 blocks
==1888==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==1888==
==1888== All heap blocks were freed -- no leaks are possible
==1888==
==1888== For counts of detected and suppressed errors, rerun with: -v
==1888== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0
santorum:~/workspace (master) $
```

```
Frecuencia del 1: 1994/20000
Porcentaje del 1: 9.97%

Frecuencia del 2: 2078/20000
Porcentaje del 2: 10.39%

Frecuencia del 3: 1960/20000
Porcentaje del 3: 9.80%

Frecuencia del 4: 2071/20000
Porcentaje del 4: 10.36%

Frecuencia del 5: 1998/20000
Porcentaje del 5: 9.99%

Frecuencia del 6: 1993/20000
Porcentaje del 6: 9.96%

Frecuencia del 7: 1973/20000
Porcentaje del 7: 9.87%

Frecuencia del 8: 1961/20000
Porcentaje del 8: 9.80%

Frecuencia del 9: 1994/20000
Porcentaje del 9: 9.97%

Frecuencia del 10: 1978/20000
Porcentaje del 10: 9.89%

==1874==
==1874== HEAP SUMMARY:
==1874==     in use at exit: 0 bytes in 0 blocks
==1874==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==1874==
==1874== All heap blocks were freed -- no leaks are possible
==1874==
==1874== For counts of detected and suppressed errors, rerun with: -v
==1874== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```


Histograma de frecuencias
"data.txt"
Frecuencia
Número
14
12
10
8
6
4
2
0
-2
-2
0
2
4
6
8
10
12
14


Histograma de frecuencia
"data.txt"
Frecuencia
Número
140
120
100
80
60
40
20
0
0
2
4
6
8
10


Histograma de frecuencia (10000 números aleatorios)
"data.txt"
Frecuencia
Número
1000
800
600
400
200
0
0
2
4
6
8
10

5.2 Apartado 2

```
santorum:~/workspace (master) $ valgrind --leak-check=full ./ejercicio2 -tamanio 10 -numP 10
==2496== Memcheck, a memory error detector
==2496== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2496== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2496== Command: ./ejercicio2 -tamanio 10 -numP 10
==2496==
Practica numero 1, apartado 2
Realizada por: Alejandro Santorum & David Cabornero
Pareja 10
Grupo: 1201
8 9 0 2 5 6 7 4 3 1
9 6 4 7 2 3 8 5 1 0
8 0 6 7 2 3 1 9 4 5
8 7 5 9 1 4 3 2 6 0
0 7 1 5 8 6 9 4 2 3
1 9 7 4 8 2 0 3 6 5
2 0 8 1 9 7 6 4 3 5
6 7 5 2 8 3 0 1 9 4
3 9 0 8 4 6 7 2 5 1
0 2 6 3 1 8 9 7 4 5
==2496==
==2496== HEAP SUMMARY:
==2496==     in use at exit: 0 bytes in 0 blocks
==2496==   total heap usage: 10 allocs, 10 frees, 400 bytes allocated
==2496==
==2496== All heap blocks were freed -- no leaks are possible
==2496==
==2496== For counts of detected and suppressed errors, rerun with: -v
==2496== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

```
santorum:~/workspace (master) $ valgrind --leak-check=full ./ejercicio2 -tamanio 25 -numP 15
==2506== Memcheck, a memory error detector
==2506== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2506== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2506== Command: ./ejercicio2 -tamanio 25 -numP 15
==2506==
Practica numero 1, apartado 2
Realizada por: Alejandro Santorum & David Cabornero
Pareja 10
Grupo: 1201
16 19 12 4 14 7 3 10 11 24 20 15 21 9 23 8 2 1 13 18 0 22 5 6 17
14 1 16 4 8 12 6 22 10 2 23 20 11 17 3 0 18 21 19 9 7 13 5 15 24
8 22 2 15 11 6 17 23 10 13 24 3 20 9 4 1 12 5 16 19 14 18 0 21 7
20 4 18 10 8 0 14 17 19 16 11 3 1 21 24 13 9 6 7 22 2 15 23 12 5
6 20 15 18 5 16 13 23 1 10 0 24 7 21 19 8 17 9 4 12 3 22 14 2 11
10 9 24 0 14 12 21 20 11 15 16 4 2 23 1 6 5 7 22 3 17 8 13 18 19
18 23 1 4 12 10 14 21 0 15 3 5 22 24 2 13 8 19 17 20 16 11 9 6 7
21 13 2 24 17 1 10 5 7 11 16 0 4 23 20 15 18 22 19 6 8 9 12 14 3
9 10 8 15 21 0 19 6 20 2 5 16 13 24 22 7 11 18 3 14 1 12 23 17 4
13 17 0 8 19 1 11 3 24 5 4 22 6 9 10 20 16 18 7 15 14 23 21 12 2
20 21 19 22 18 11 5 10 8 1 16 9 7 6 23 4 15 13 2 12 24 14 17 0 3
22 12 4 2 6 19 11 23 18 10 0 14 17 3 5 13 7 24 15 20 16 21 8 1 9
15 18 2 13 12 16 3 4 8 5 0 6 14 21 10 23 7 24 1 20 19 9 17 22 11
14 23 4 8 2 18 10 5 15 12 11 22 6 7 3 1 24 21 13 20 17 19 9 0 16
2 4 7 6 19 23 3 12 22 11 18 5 9 8 1 17 16 13 10 24 20 15 14 0 21
==2506==
==2506== HEAP SUMMARY:
==2506==     in use at exit: 0 bytes in 0 blocks
==2506==   total heap usage: 15 allocs, 15 frees, 1,500 bytes allocated
==2506==
==2506== All heap blocks were freed -- no leaks are possible
==2506==
==2506== For counts of detected and suppressed errors, rerun with: -v
==2506== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

5.3 Apartado 3

```
santorum:~/workspace (master) $ valgrind --leak-check=full ./ejercicio3 -tamanio 10 -numP 10
==2516== Memcheck, a memory error detector
==2516== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2516== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2516== Command: ./ejercicio3 -tamanio 10 -numP 10
==2516==
Practica numero 1, apartado 3
Realizada por: Alejandro Santorum & David Cabornero
Pareja 10.
Grupo: 1201
4 5 8 3 2 7 1 9 0 6
0 9 2 7 5 6 8 1 3 4
2 1 9 5 3 6 8 7 0 4
1 4 5 2 9 3 6 0 7 8
7 4 0 9 6 2 3 5 1 8
1 7 5 3 9 0 4 8 6 2
2 9 8 3 4 5 7 0 6 1
5 7 9 4 6 3 2 8 0 1
5 3 8 0 6 7 9 2 1 4
3 5 8 0 2 4 9 6 7 1
==2516==
==2516== HEAP SUMMARY:
==2516==     in use at exit: 0 bytes in 0 blocks
==2516==   total heap usage: 11 allocs, 11 frees, 480 bytes allocated
==2516==
==2516== All heap blocks were freed -- no leaks are possible
==2516==
==2516== For counts of detected and suppressed errors, rerun with: -v
==2516== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

```
santorum:~/workspace (master) $ valgrind --leak-check=full ./ejercicio3 -tamanio 25 -numP 15
==2521== Memcheck, a memory error detector
==2521== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2521== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2521== Command: ./ejercicio3 -tamanio 25 -numP 15
==2521==
Practica numero 1, apartado 3
Realizada por: Alejandro Santorum & David Cabornero
Pareja 10.
Grupo: 1201
21 8 19 10 12 15 14 9 22 4 6 24 16 17 23 18 20 5 1 13 2 7 11 0 3
24 4 7 1 21 14 23 19 20 2 11 13 5 9 17 15 22 3 18 10 8 0 6 16 12
2 0 24 17 14 23 13 16 9 19 18 5 6 10 21 11 3 22 12 4 8 7 15 1 20
22 10 3 23 1 16 13 14 18 15 4 12 9 8 21 6 17 11 20 5 2 24 0 7 19
16 4 7 23 13 17 3 1 8 24 14 15 18 21 22 0 11 6 20 2 19 9 12 5 10
4 2 16 23 24 19 21 18 22 9 7 15 20 10 6 14 12 17 13 0 3 5 8 11 1
14 11 6 22 12 23 10 17 24 3 7 8 19 16 20 4 15 18 5 13 2 0 1 21 9
13 18 4 9 14 10 7 8 20 0 24 12 2 15 21 16 3 1 19 23 22 5 6 11 17
23 0 1 16 13 18 19 7 14 22 8 2 4 17 24 11 21 10 12 5 20 15 9 3 6
21 5 23 12 24 6 3 2 8 14 17 18 4 15 19 11 16 7 9 13 20 0 1 22 10
2 21 18 8 14 20 15 13 24 16 1 23 19 4 11 5 6 12 0 10 22 7 3 9 17
10 1 17 16 8 2 14 12 15 11 18 24 9 6 5 23 21 13 4 19 3 0 7 22 20
23 20 4 13 12 0 6 22 24 1 17 3 16 8 19 7 10 2 5 14 9 21 11 15 18
2 20 22 9 13 16 24 14 7 10 23 0 12 6 21 4 8 18 19 15 1 3 17 5 11
17 13 9 22 15 11 18 6 2 0 24 7 1 10 12 20 16 23 4 19 8 21 3 5 14
==2521==
==2521== HEAP SUMMARY:
==2521==     in use at exit: 0 bytes in 0 blocks
==2521==   total heap usage: 16 allocs, 16 frees, 1,620 bytes allocated
==2521==
==2521== All heap blocks were freed -- no leaks are possible
==2521==
==2521== For counts of detected and suppressed errors, rerun with: -v
==2521== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

5.4 Apartado 4

```
santorum:~/workspace (master) $ valgrind --leak-check=full ./ejercicio4 -tamanio 5
==2541== Memcheck, a memory error detector
==2541== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2541== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2541== Command: ./ejercicio4 -tamanio 5
==2541==
Practica numero 1, apartado 4
Realizada por: Alejandro Santorum & David Cabornero
Grupo: 1201
0       1       2       3       4
==2541==
==2541== HEAP SUMMARY:
==2541==     in use at exit: 0 bytes in 0 blocks
==2541==   total heap usage: 1 allocs, 1 frees, 20 bytes allocated
==2541==
==2541== All heap blocks were freed -- no leaks are possible
==2541==
==2541== For counts of detected and suppressed errors, rerun with: -v
==2541== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

```
santorum:~/workspace (master) $ valgrind --leak-check=full ./ejercicio4 -tamanio 10
==2536== Memcheck, a memory error detector
==2536== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2536== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2536== Command: ./ejercicio4 -tamanio 10
==2536==
Practica numero 1, apartado 4
Realizada por: Alejandro Santorum & David Cabornero
Grupo: 1201
0       1       2       3       4       5       6       7       8       9
==2536==
==2536== HEAP SUMMARY:
==2536==     in use at exit: 0 bytes in 0 blocks
==2536==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==2536==
==2536== All heap blocks were freed -- no leaks are possible
==2536==
==2536== For counts of detected and suppressed errors, rerun with: -v
==2536== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

5.5 Apartado 5

```
santorum:~/workspace (master) $ valgrind --leak-check=full ./ejercicio5 -num_min 1 -num_max 50 -incr 10 -numP 10 -fichSalida ficherosalida
==2551== Memcheck, a memory error detector
==2551== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2551== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2551== Command: ./ejercicio5 -num_min 1 -num_max 50 -incr 10 -numP 10 -fichSalida ficherosalida
==2551==
Practica numero 1, apartado 5
Realizada por: Alejandro Santorum & David Cabornero
Pareja 10.
Grupo: 1201
array = 5
Salida correcta
==2551==
==2551== HEAP SUMMARY:
==2551==     in use at exit: 0 bytes in 0 blocks
==2551==   total heap usage: 57 allocs, 57 frees, 5,328 bytes allocated
==2551==
==2551== All heap blocks were freed -- no leaks are possible
==2551==
==2551== For counts of detected and suppressed errors, rerun with: -v
==2551== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

bash - "santorum × | ficherosalida

	size	time(ms)	avg_ob	max_ob	min_ob.
1	1	0.082300	0.00	0	0
2	11	0.110800	54.00	54	54
3	21	0.013400	209.00	209	209
4	31	0.028500	464.00	464	464
5	41	0.047600	819.00	819	819

```
santorum:~/workspace (master) $ valgrind --leak-check=full ./ejercicio5 -num_min 10 -num_max 80 -incr 4 -numP 1000 -fichSalida ficherosalida
==2562== Memcheck, a memory error detector
==2562== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2562== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2562== Command: ./ejercicio5 -num_min 10 -num_max 80 -incr 4 -numP 1000 -fichSalida ficherosalida
==2562==
Practica numero 1, apartado 5
Realizada por: Alejandro Santorum & David Cabornero
Pareja 10.
Grupo: 1201
array = 18
Salida correcta
==2562==
==2562== HEAP SUMMARY:
==2562==     in use at exit: 0 bytes in 0 blocks
==2562==   total heap usage: 18,020 allocs, 18,020 frees, 3,313,144 bytes allocated
==2562==
==2562== All heap blocks were freed -- no leaks are possible
==2562==
==2562== For counts of detected and suppressed errors, rerun with: -v
==2562== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
santorum:~/workspace (master) $
```

bash - "santorum × | ficherosalida

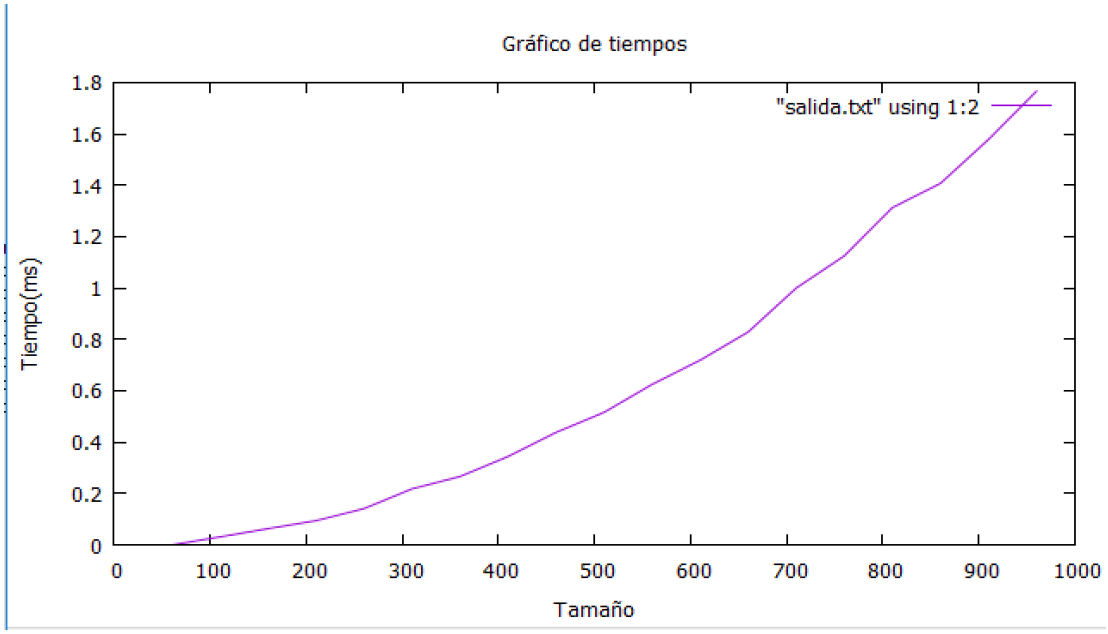
	size	time(ms)	avg_ob	max_ob	min_ob.
1					
2	size	time(ms)	avg_ob	max_ob	min_ob.
3	10	0.005689	44.00	44	44
4	14	0.006685	90.00	90	90
5	18	0.010352	152.00	152	152
6	22	0.014950	230.00	230	230
7	26	0.020071	324.00	324	324
8	30	0.028436	434.00	434	434
9	34	0.033716	560.00	560	560
10	38	0.041803	702.00	702	702
11	42	0.053264	860.00	860	860
12	46	0.061160	1034.00	1034	1034
13	50	0.072414	1224.00	1224	1224
14	54	0.086460	1430.00	1430	1430
15	58	0.099134	1652.00	1652	1652
16	62	0.110684	1890.00	1890	1890
17	66	0.125735	2144.00	2144	2144
18	70	0.145888	2414.00	2414	2414
19	74	0.163652	2700.00	2700	2700
20	78	0.205504	3002.00	3002	3002

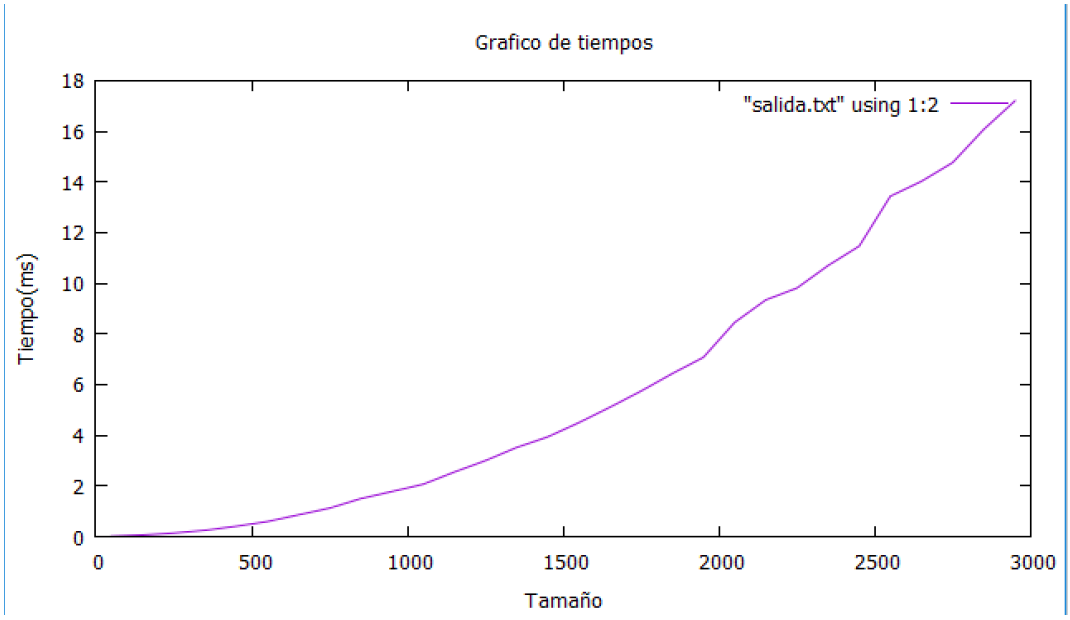
Gráfica comparando los tiempos mejor peor y medio en OBs para BubbleSort, comentarios a la gráfica.

TAMAÑO	TIEMPO(MS)	OB_MEDIO	OB_MAX	OB_MIN
50	0.015000	1224.00	1224	1224
150	0.063000	11174.00	11174	11174
250	0.140000	31124.00	31124	31124
350	0.250000	61074.00	61074	61074
450	0.406000	101024.00	101024	101024
550	0.594000	150974.00	150974	150974
650	0.858000	210924.00	210924	210924
750	1.125000	280874.00	280874	280874
850	1.500000	360824.00	360824	360824
950	1.782000	450774.00	450774	450774
1050	2.063000	550724.00	550724	550724
1150	2.547000	660674.00	660674	660674
1250	3.000000	780624.00	780624	780624
1350	3.516000	910574.00	910574	910574
1450	3.938000	1050524.00	1050524	1050524
1550	4.500000	1200474.00	1200474	1200474
1650	5.109000	1360424.00	1360424	1360424
1750	5.750000	1530374.00	1530374	1530374
1850	6.438000	1710324.00	1710324	1710324
1950	7.079000	1900274.00	1900274	1900274
2050	8.453000	2100224.00	2100224	2100224
2150	9.343000	2310174.00	2310174	2310174
2250	9.812000	2530124.00	2530124	2530124
2350	10.703000	2760074.00	2760074	2760074
2450	11.469000	3000024.00	3000024	3000024
2550	13.438000	3249974.00	3249974	3249974
2650	14.030000	3509924.00	3509924	3509924
2750	14.766000	3779874.00	3779874	3779874
2850	16.078000	4059824.00	4059824	4059824
2950	17.203000	4349774.00	4349774	4349774

Como no hemos utilizado flags, el tiempo medio, el caso mejor y el caso peor coinciden.

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





5. Respuesta a las preguntas teóricas.

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

5.1 Pregunta 1

Nuestra implementación de aleat_num se basa en la información obtenida en el capítulo 7 del libro “Numerical recipes in C: the art of scientific computing”.

El inconveniente fundamental de la función rand() junto con la aritmética modular era que priorizaba los números con bits de menor peso, por lo que no era estrictamente aleatoria. La nueva función implementada gracias al libro mencionado amortigua dicha tendencia por los bits de menor peso, priorizando los bits de mayor peso; además de que no utiliza aritmética modular lo que potencia la aleatoriedad.

5.2 Pregunta 2

Después de realizar comprobaciones (la tabla no puede ser NULL, el primer elemento no puede ser mayor que el último y si el primero y el último son iguales la tabla ya está ordenada, ya que solo tiene un elemento) procedemos al bucle for, que comprende el grueso del programa.

Se nos ha recomendado no usar flags en el BubbleSort, para poder contabilizar el número de iteraciones del bucle de manera más sistemática, sin embargo en el código lo dejamos indicado.

El bucle externo recorrerá la tabla desde el último elemento hasta el primero. Este bucle nos sirve para acotar la parte de la tabla que ya está ordenada. Según vaya disminuyendo i, la parte de la tabla no ordenada será la comprendida entre el primer elemento y i, mientras que la parte que comprende entre i y el último elemento estará ordenada.

En el bucle más interno, recorremos la tabla desde el primer elemento hasta i, es decir, la parte de la tabla no ordenada. Dentro del bucle, vamos comparando cada elemento de la tabla con el siguiente, de tal forma que si el elemento anterior es mayor que el siguiente realizaremos un swap. De esta manera, nos aseguramos de que el elemento mayor de la parte de la tabla no ordenada siempre quede en el último lugar, pasando así a ser parte de la tabla ordenada y pudiendo reducir i una unidad.

5.3 Pregunta 3

Como se ha dicho en el ejercicio 2, el bucle externo sirve para seleccionar la parte de la tabla que no está ordenada, y en cada iteración queda ordenado el último elemento de la tabla no ordenada, pudiendo reducir i una unidad.

Esto significa que, si aplicáramos el bucle sobre el primer número (lo cual sería la última iteración del bucle) estaríamos ordenando la lista desordenada que comprende desde el primer número hasta el primer número. La tabla desordenada a ordenar solo tendría un elemento, por lo que resulta absurdo realizar esta última iteración del bucle.

5.4 Pregunta 4

Para ser la OB, es necesario cumplir un serie de requisitos:

La OB ha de estar en el bucle más interno, ha de ser una operación representativa y ha de ejecutarse siempre que se ejecute el bucle más interno.

Solo existe una candidata que cumpla estos tres requisitos: la comparación de clave, donde se compara un elemento de la tabla con el siguiente. No podríamos poner,

por ejemplo, el swap, ya que solo se ejecuta cuando se cumple la condición impuesta por la comparación de clave.

5.5 Pregunta 5

Como el BubbleSort que se nos ha pedido no tiene flags, siempre va a hacer el mismo número de iteraciones, ya que el número de comparaciones de clave no depende de cómo esté ordenada la tabla sino del tamaño, solo los Swap dependen de cómo esté ordenada la tabla.

$$W_{BSort}(N) = B_{BSort}(N) = \sum_{i=2}^{N} n_{BSort}(T, i) = \sum_{i=2}^{N} \sum_{j=1}^{i} 1 = \sum_{i=2}^{N} i = (N-1) * N/2 = \frac{N^2}{2} - \frac{N}{2} = \frac{N^2}{2} + O(N)$$

6. Conclusiones finales.

En lo que respecta a las fugas de memoria, la práctica está perfecta. Hemos intentado ser lo más rigurosos posibles en comprobaciones, haciendo solo las más necesarias durante el grueso del código, ya que somos conscientes de que dificulta mucho su lectura.

Aparte de lo exigido, hemos añadido el BubbleSort_flag, aunque solo haya sido el código, ya que no nos hemos detenido a hablar en los apartados del informe.

En nuestra opinión la práctica está acorde con lo visto en clase, no como en asignaturas anteriores en las cuales las prácticas eran otro mundo diferente a la teoría.

Finalmente, debemos aclarar que hemos intentado acotar el informe al mínimo exigido. Si se considera que este tamaño es excesivo intentaremos mejorar para las futuras prácticas.