Análisis de algoritmos
Escuela Politécnica Superior, UAM, 2016–2017

Conjunto de Prácticas no. 1

Fecha de entrega de la práctica

- Grupos del Lunes: 10 de Octubre.
- Grupos del Martes: 11 de Octubre.
- Grupos del Viernes: 14 de Octubre.

La documentación impresa se entregará al comienzo de la clase de prácticas.

Esta primera práctica servirá como asentamiento para futuras prácticas. Se generarán varios ficheros .c **permutaciones.c ordenacion.c tiempos.c** donde se irán guardando a modo de librería los códigos de las distintas funciones C; igualmente se incluirán en ficheros .h **permutaciones.h ordenacion.h tiempos.h** los prototipos de las funciones implementadas en los ficheros .c y los # **define** genéricos, tales como:

```
#define ERR  -1
#define OK   (!(ERR))
```

Será obligatorio ajustarse al esquema de prácticas que se da, es decir, los prototipos de las funciones no deben modificarse (excepto si lo indica el profesor de prácticas).
Consejos en la implementación de las rutinas:

1. Comprobad que los argumentos de entrada de la rutina son correctos. Independencia de las funciones.
2. Antes de salir de una rutina o del programa liberad toda la memoria y cerrad todos los ficheros que ya no se utilicen.
3. Comprobad que los ficheros se pueden abrir, en caso negativo actuad como en el punto 2.
4. Comprobad en cada caso que al reservar memoria, ésta se ha realizado eficientemente y en el caso de que no se hubiera podido reservar, antes de salir de la rutina o el programa proceded como en el punto 2.
5. Comprobad los programas en ejecuciones extremas para verificar el buen comportamiento del programa (p.e. tablas muy grandes, entradas ilegales, ...).
6. Utilizad entornos de ejecución en modo protegidos (Windows 98, Windows NT, 2000, XP, Vista, Windows 7, Windows 8, Unix, Linux, OSX ... Nunca DOS puro).
7. **Las prácticas serán corregidas en un sistema Unix o bien Linux, con lo cual los programas deberán desarrollarse en ANSI C para que sean portables.**

La práctica comenzará desarrollando en un fichero **permutaciones.c** un conjunto de rutinas C que generen permutaciones aleatorias de un cierto número de elementos. Dichas rutinas se usarán más adelante para obtener las entradas de los algoritmos de ordenación.
Cada práctica se realizará en un período de tres o cuatro semanas.

Primer Bloque

1. La rutina C **rand** de la librería **stdlib** genera números aleatorios equiprobables entre 0 y el valor de **RAND_MAX**. Usar esta función para construir una rutina **int aleat_num (int inf, int sup)** que genere un número aleatorio equiprobable entre los enteros **inf, sup**, ambos inclusive.

   El programa C de prueba de la rutina (a entregar) debe tener el siguiente formato:

```
/*********************************************/
/* Programa: P1_1_???        Fecha:           */
/* Autores:                                   */
/*                                            */
/*  Programa que genera numeros aleatorios    */
/*  entre dos numeros dados                   */
/*                                            */
/*  Entrada: Linea de comandos                */
/*             -limInf: limite inferior       */
/*             -limSup: limite superior       */
/*             -numN:    cantidad de numeros  */
/*  Salida:  0: OK, -1: ERR                   */
/*********************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "ordenar.h"

int main(int argc, char** argv)
{
  int i;
  unsigned int inf, sup, num, j;

  srand(time(NULL));

  if (argc != 7) {
    fprintf(stderr, "Error en los parametros de entrada:\n\n");
    fprintf(stderr, "%s -limInf <int> -limSup <int> -numN <int>\n", argv[0]);
    fprintf(stderr, "Donde:\n");
    fprintf(stderr, "   -limInf : Limite inferior.\n");
    fprintf(stderr, "   -limSup : Limite superior.\n");
    fprintf(stderr, "   -numN    : Cantidad de numeros a generar.\n");
    exit(-1);
  }

  printf("Practica numero 1, apartado 1\n");
  printf("Realizada por: Vuestros nombres\n");
  printf("Grupo: Vuestro grupo\n");
  /* comprueba la linea de comandos */
  for(i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-limInf") == 0) {
      inf = atoi(argv[++i]);
    } else if (strcmp(argv[i], "-limSup") == 0) {
      sup = atoi(argv[++i]);
    } else if (strcmp(argv[i], "-numN") == 0) {
      num = atoi(argv[++i]);
    } else {
      fprintf(stderr, "Parametro %s es incorrecto\n", argv[i]);
    }
  }

  for(j = 0; j < num; j++)  /* imprimimos los datos */
    printf("%d\n", aleat_num(inf, sup));

  return 0;
}

/*****************************************/
/* Funcion: aleat_num              Fecha:  */
/* Autores:                                */
/*                                         */
/*  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)
{
  /* vuestro codigo */
}
```

Comprobad mediante la elaboración de un histograma si la generación aleatoria es equiprobable para cada dígito.

Obs: Se valorará el nivel de aleatoriedad de la función implementada, por lo que se recomienda pensar detenidamente sobre su diseño en lugar de implementar la primera idea que se tenga. Como ayuda se recomienda consultar el capítulo 7 (Random Numbers) del libro "Numerical recipes in C: the art of scientific computing" del cual hay varias copias disponibles en la biblioteca.

2. El siguiente pseudocódigo proporciona un método de generación de permutaciones aleatorias

```
para i de 1 a N:
   perm[i] = i;

para i de 1 a N:
   intercambiar perm[i] con perm[aleat_num(i, N)];
```

donde **aleat_num** es la rutina del ejercicio anterior.

Escribid una rutina C **int *genera_perm(int n)** para dicho algoritmo.

El programa C de prueba de la rutina debe tener el siguiente formato:

```
/*****************************************************/
/* Programa: P1_2_???        Fecha:                  */
/* Autores:                                          */
/*                                                   */
/*  Programa que genera permutaciones                */
/*  aleatorias                                       */
/*                                                   */
/*  Entrada: Linea de comandos                       */
/*              -tamanio: numero elementos permutacion  */
/*              -numP:     numero de permutaciones     */
/*  Salida:  0: OK, -1: ERR                          */
/*****************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "permutaciones.h"

int main(int argc, char** argv)
{
  int i;
  unsigned int num, tamanio, j, k;
  int* perm = NULL;

  srand(time(NULL));
  if (argc != 5) {
    fprintf(stderr, "Error en los parametros de entrada:\n\n");
    fprintf(stderr, "%s -tamanio <int> -numP <int>\n", argv[0]);
    fprintf(stderr, "Donde:\n");
    fprintf(stderr, "   -tamanio  : numero elementos permutacion.\n");
    fprintf(stderr, "   -numP     : numero de permutaciones.\n");
    exit(-1);
  }
  printf("Practica numero 1, apartado 2\n");
  printf("Realizada por: Vuestros nombres\n");
  printf("Grupo: Vuestro grupo\n");
  /* comprueba la linea de comandos */
  for(i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-tamanio") == 0) {
      tamanio = atoi(argv[++i]);
    } else if (strcmp(argv[i], "-numP") == 0) {
      num = atoi(argv[++i]);
    } else {
      fprintf(stderr, "Parametro %s es incorrecto\n", argv[i]);
    }
  }

  for(j = 0; j < num; j++) { /* imprimimos el resultado */
    perm = genera_perm(tamanio);
    if (perm == NULL) {                 /* error */
      printf("Error: No hay memoria\n");
      exit(-1);
    } else {
      for(k = 0; k < tamanio; k++) /* imprimimos cada elemento */
    printf("%d ", perm[k]);
      printf("\n");
      free(perm);           /* liberamos la permutacion */
    }
  }

  return 0;
}

/*****************************************/
/* Funcion: genera_perm            Fecha:   */
/* Autores:                                 */
/*                                          */
/*  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)
{
  /* vuestro codigo */
}
```

Segundo bloque

3. Construid la rutina **int **genera_permutaciones(int n_perms, int tamanio)** que genera mediante la función **genera_perm** del ejercicio anterior **n_perms** permutaciones equiprobables de **tamanio** elementos cada una. El programa C de prueba de la rutina **genera_permutaciones** debe tener el siguiente formato:

```
/*************************************************/
/* Programa: P1_3_???        Fecha:              */
/* Autores:                                      */
/*                                               */
/*  Programa que genera permutaciones N          */
/*  aleatorias de M elementos                    */
/*                                               */
/*  Entrada: Linea de comandos                   */
/*          -tamanio: numero elementos permutacion*/
/*          -numP:     numero de permutaciones   */
/*  Salida:  0: OK, -1: ERR                      */
/*************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "ordenar.h"

int main(int argc, char** argv)
{
  int i;
  unsigned int n_perms, tamanio, j, k;
  int** perms = NULL;

  srand(time(NULL));
  if (argc != 5) {
    fprintf(stderr, "Error en los parametros de entrada:\n\n");
    fprintf(stderr, "%s -tamanio <int> -numP <int>\n", argv[0]);
    fprintf(stderr, "Donde:\n");
    fprintf(stderr, "   -tamanio  : numero elementos permutacion.\n");
    fprintf(stderr, "   -numP     : numero de permutaciones.\n");
    exit(-1);
  }
  printf("Practica numero 1, apartado 3\n");
  printf("Realizada por: Vuestros nombres\n");
  printf("Grupo: Vuestro grupo\n");
  /* comprueba la linea de comandos */
  for(i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-tamanio") == 0) {
      tamanio = atoi(argv[++i]);
    } else if (strcmp(argv[i], "-numP") == 0) {
      n_perms = atoi(argv[++i]);
    } else {
      fprintf(stderr, "Parametro %s es incorrecto\n", argv[i]);
    }
  }

  perms = genera_permutaciones(n_perms, tamanio);
  if (perms == NULL) {    /* error */
    printf("Error: No hay memoria\n");
    exit(-1);
  } else {
    for(j = 0; j < n_perms; j++) {   /* para cada permutacion */
      for(k = 0; k < tamanio; k++)
printf("%d ", perms[j][k]); /* imprimimos cada elemento */
      printf("\n");
      free(perms[j]);   /* liberamos la permutacion */
    }
    free(perms);  /* liberamos el array de permutaciones */
  }

  return 0;
}

/*************************************************/
```

```
/* Funcion: genera_permutaciones   Fecha:         */
/* Autores:                                       */
/*                                                */
/*  Funcion que genera n_perms permutaciones      */
/*  aleatorias de tamanio elementos               */
/*                                                */
/*  Entrada:                                      */
/*   int n_perms: Numero de permutaciones         */
/*   int tamanio: 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 tamanio)
{
  /* vuestro codigo */
}
```

Algoritmos de ordenación

En esta sección se determinará experimentalmente el tiempo medio de ejecución y el número medio, mejor y peor de veces que se ejecuta la Operación Básica (OB) del algoritmo de ordenación por inserción sobre tablas de diferentes tamaños, y se comparará con el análisis teórico.

4. Implementar el fichero **ordenacion.c** una función **int InsertSort(int *tabla, int ip, int iu)** para el método de ordenación por inserción (InsertSort), esta función devuelve ERR en caso de error o el número de veces que se ha ejecutado la OB en el caso de que la tabla se ordene correctamente, **tabla** es la tabla a ordenar, **ip** es el primer elemento de la tabla e **iu** es el último elemento de la tabla.

   El programa C de prueba de la rutina (a entregar) debe tener el siguiente formato:

```
/************************************************/
/* Programa: P1_4_???        Fecha:             */
/* Autores:                                     */
/*                                              */
/*  Programa que Comprueba InsertSort           */
/*                                              */
/*  Entrada: Linea de comandos                  */
/*          -tamanio: numero elementos permutacion*/
/*  Salida:  0: OK, -1: ERR                     */
/************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "permutaciones.h"
#include "ordenacion.h"

int main(int argc, char** argv)
{
  int tamano, i, j, ret;
  int* perm = NULL;

  srand(time(NULL));
  if (argc != 3) {
    fprintf(stderr, "Error en los parametros de entrada:\n\n");
    fprintf(stderr, "%s -tamanio <int> -numP <int>\n", argv[0]);
    fprintf(stderr, "Donde:\n");
    fprintf(stderr, "   -tamanio  : numero elementos permutacion.\n");
    return 0;
  }
  printf("Practica numero 1, apartado 4\n");
  printf("Realizada por: Vuestros nombres\n");
  printf("Grupo: Vuestro grupo\n");
  /* comprueba la linea de comandos */
  for(i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-tamanio") == 0) {
      tamano = atoi(argv[++i]);
    } else {
      fprintf(stderr, "Parametro %s es incorrecto\n", argv[i]);
    }
  }

  perm = genera_perm(tamano);
  if (perm == NULL) {                /* error */
    printf("Error: No hay memoria\n");
    exit(-1);
```

```
  }
  ret = InsertSort(perm, 0, tamano-1);
  if (ret == ERR) {
    printf("Error: Error en InsertSort\n");
    free(perm);
    exit(-1);
  }
  for(j = 0; j < tamano; j++)
    printf("%d \t", perm[j]);
  printf("\n");

  free(perm);

  return 0;
}
```

Tercer Bloque

5. Definir en **tiempos.h** la siguiente estructura que servirá para almacenar los tiempos de ejecución de un algoritmo sobre un conjunto de permutaciones:

```
typedef struct tiempo {
   int n_perms;       // numero de permutaciones
   int tamanio;       // tamanio de las permutaciones
   int n_veces;       // numero de veces que se evalua cada permutacion
   double tiempo;     // tiempo promedio de reloj
   double medio_ob;   // numero promedio de veces que se ejecuta la OB
   int min_ob;        // minimo de ejecuciones de la OB
   int max_ob;        // maximo de ejecuciones de la OB
} TIEMPO, *PTIEMPO;
```

Implementad en el fichero **tiempos.c** la función:

short tiempo_medio_ordenacion(pfunc_ordena metodo,

int n_perms, int tamanio, PTIEMPO ptiempo),

donde **pfunc_ordena** es un puntero a la función de ordenación, definido como:

typedef int (* pfunc_ordena)(int*, int, int);

este typedef deberá incluirse en `ordenacion.h`, **n_perms** representa el número de permutaciones a generar y ordenar por el método que se use (en este caso método de inserciń), **tamanio** es el tamaño de cada permutación y **ptiempo** es un puntero a una estructura de tipo **TIEMPO** que a la salida de la función contendrá el número de permutaciones promediadas en el campo **n_perms**, el tamaño de las permutaciones en el campo **tamanio**, el número de veces que se ordena cada permutación en el campo **n_veces**, el tiempo medio de ejecución (en segundos) en el campo **tiempo**, el número promedio de veces que se ejecutó la OB en el campo **medio_ob**, el número mínimo de veces que se ejecutó la OB **min_ob** y el número máximo de veces que se ejecutó la OB en el campo **max_ob** para el algoritmo de ordenación **ordena_metodo** sobre las permutaciones generadas por la función **genera_permutaciones**.

La rutina **tiempo_medio_ordenacion** devuelve devuelve ERR en caso de error y OK en el caso de que las tablas se ordenen correctamente.

Implementad además la función:

short genera_tiempos_ordenacion(pfunc_ordena metodo, char * fichero,

int num_min,int num_max,int incr, int n_perms)

que escribe en el fichero **fichero** los tiempos medios, y los números promedio, mínimo y máximo de veces que se ejecuta la OB en la ejecución del algoritmo de ordenación **metodo** con **n_perms** permutaciones de tamaños en el rango desde **num_min** hasta **num_max**, ambos incluidos, usando incrementos de tamaño **incr**. La rutina devolverá el valor ERR en caso de error y OK en caso contrario.

genera_tiempos_ordenacion llamará a una función

short guarda_tabla_tiempos(char *fichero, PTIEMPO tiempo, int N)

con la que se imprime en un fichero una tabla con cinco columnas correspondientes al tamaño de la permutación, al tiempo de ejecución y al número promedio, máximo y mínimo de veces que se ejecuta la OB; el array **tiempo** guarda los tiempos de ejecución y **N** es el tamaño del array **tiempo**.

El programa C de prueba (a entregar) debe tener el siguiente formato:

```
/*****************************************************************/
/* Programa: P1_5_???        Fecha:                              */
/* Autores:                                                      */
/*                                                               */
/*  Programa que escribe en un fichero                           */
/*  los tiempos medios del algoritmo de                          */
/*  ordenacion por Insercion                                    */
/*                                                               */
/*  Entrada: Linea de comandos                                   */
/*          -num_min: numero minimo de elementos de la tabla     */
/*          -num_max: numero minimo de elementos de la tabla     */
/*          -incr:    incremento\n                               */
/*          -numP:    Introduce el numero de permutaciones a promediar*/
/*          -fichSalida: Nombre del fichero de salida            */
/*  Salida:  0 si hubo error                                     */
/*           -1 en caso contrario                                */
/*****************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "ordenacion.h"
#include "tiempos.h"

int main(int argc, char** argv)
{
  int i, num_min, num_max, incr, n_perms;
  char nombre[256];
  short ret;

  srand(time(NULL));
  if (argc != 11) {
    fprintf(stderr, "Error en los parametros de entrada:\n\n");
    fprintf(stderr, "%s -num_min <int> -num_max <int> -incr <int>\n", argv[0]);
    fprintf(stderr, "\t\t -numP <int> -fichSalida <string> \n");
    fprintf(stderr, "Donde:\n");
    fprintf(stderr, "-num_min: numero minimo de elementos de la tabla\n");
    fprintf(stderr, "-num_max: numero minimo de elementos de la tabla\n");
    fprintf(stderr, "-incr:    incremento\n");
    fprintf(stderr, "-numP:    Introduce el numero de permutaciones a promediar\n");
    fprintf(stderr, "-fichSalida: Nombre del fichero de salida\n");

    exit(-1);
  }
  printf("Practica numero 1, apartado 5\n");
  printf("Realizada por: Vuestros nombres\n");
  printf("Grupo: Vuestro grupo\n");
  /* comprueba la linea de comandos */
  for(i = 1; i < argc ; i++) {
    if (strcmp(argv[i], "-num_min") == 0) {
      num_min = atoi(argv[++i]);
    } else if (strcmp(argv[i], "-num_max") == 0) {
      num_max = atoi(argv[++i]);
    } else if (strcmp(argv[i], "-incr") == 0) {
      incr = atoi(argv[++i]);
    } else if (strcmp(argv[i], "-numP") == 0) {
      n_perms = atoi(argv[++i]);
    } else if (strcmp(argv[i], "-fichSalida") == 0) {
      strcpy(nombre, argv[++i]);
    } else {
      fprintf(stderr, "Parametro %s es incorrecto\n", argv[i]);
    }
  }

  /* calculamos los tiempos */
  ret = genera_tiempos_ordenacion(InsertSort, nombre,
  num_min, num_max,
  incr, n_perms);

  if (ret == ERR) { /* ERR_TIME debera ser un numero negativo */
    printf("Error en la funcion Time_Ordena\n");
    exit(-1);
  }

  printf("Salida correcta \n");

  return 0;
}


/*************************************************/
/* Funcion: tiempo_medio_ordenacion       Fecha:     */
/*                                                   */
/*  Vuestra documentacion (formato igual             */
/*  que en el primer apartado):                      */
```

```
/*************************************************/
short tiempo_medio_ordenacion_dist(pfunc_ordena metodo, int n_perms,
      int tamanio, PTIEMPO ptiempo)
{
  /* vuestro codigo */
}

/*******************************************************/
/* Funcion: genera_tiempos_ordenacion     Fecha:       */
/*                                                     */
/*  Vuestra documentacion                              */
/*******************************************************/
short genera_tiempos_ordenacion_dist(pfunc_ordena metodo, char* fichero, int num_min, int num_max, int incr, int n_perms)
{
  /* vuestro codigo */
}
```

Comentario: No sería extraño que al ejecutar vuestro programa obtengáis que el tiempo de ejecución es cero, eso es debido a que la velocidad del procesador es tan alta que no llega ni siquiera a consumir un TIC del reloj medido mediante la función **clock** durante la llamada a **tiempo_medio_ordenacion**. Luego, para ver el tiempo de ejecución será necesario introducir algún mecanismo de retardo o utilizar otra función más precisa para la medición de tiempos.

A modo de sugerencia, se podría modificar **int tiempo_medio_ordenacion(pfunc_ordena metodo, int n_perms, int tamanio, PTIEMPO ptiempo, int n_veces)**, de tal forma que se ordenará **n_veces** cada permutación de tamaño **tamanio** según el método de ordenación **metodo**.

Otra opción es utilizar funciones más precisas que la función de librería estándar **clock** para la medición de tiempos. Por ejemplo en UNIX y LINUX existe la función del sistema **clock_gettime** que da una precisión de nanosegundos (10^{-9}). Esta función no esta definida en ANSI C y no existe en Windows (aunque sí existen llamadas equivalentes) ni OSX.

6. En ocasiones es interesante ordenar una tabla en valores de mayor a menor. Implementar una rutina **int InsertSortInv(int* tabla, int ip, int iu)** con los mismos argumentos y retorno que **InsertSort** pero que ordene la tabla en orden inverso (de mayor a menor), obtener los tiempos de ejecución comparar los resultados obtenidos con los de la rutina **InsertSort**.

   Importante:

   - Considerad sólo el tiempo de ordenación. Es recomendable añadir en **tiempo_medio_ordenacion** el argumento necesario para incorporar el retardo e igualmente modificar el programa principal con la previsión de poder cambiar el tiempo de retardo si vais a utilizar la función de librería estándar **clock**.
   - Tened cuidado de no re-ordenar una permutación que ya ha sido ordenada. Una manera de hacer esto es realizar una copia de la permutación en un array auxiliar y ordenar el array auxiliar tantas veces como indique el retardo.

Cuestiones sobre la práctica

1. Justifica tu implementación de **aleat_num** ¿en qué ideas se basa? ¿de qué libro/artículo, si alguno, has tomado la idea? Propón un método alternativo de generación de números aleatorios y justifica sus ventajas/desventajas respecto a tu elección.

2. Justifica lo más formalmente que puedas la corrección (o dicho de otra manera, el porqué ordena bien) del algoritmo **InsertSort**.

3. ¿Por qué el bucle de **InsertSort** no actúa sobre el primer elemento de la tabla?

4. ¿Cuál es la operación básica de **InsertSort**?

5. Dar tiempos de ejecución en función del tamaño de entrada n para el caso peor $W_{SS}(n)$ y el caso mejor $B_{SS}(n)$ de **InsertSort**. Utilizar la notación asintótica (O, Θ, o, Ω,etc) siempre que se pueda. ¿Son estos valores los mismos que los del algoritmo **InsertSortInv** ?

6. Compara los tiempos obtenidos para **InsertSort** e **InsertSortInv**, justifica las similitudes o diferencias entre ambos (es decir, indicad si las gráficas son iguales o distintas y por qué).

Material a entregar en cada uno de los apartados

Documentación: La documentación constará de los siguientes apartados:

1. **Introducción:** Consiste en una descripción técnica del trabajo que se va a realizar, qué objetivos se pretenden alcanzar, qué datos de entrada requiere vuestro programa y qué datos se obtienen de salida, así como cualquier tipo de comentario sobre la práctica.
2. **Código impreso:** El código de la rutina según el apartado. Como código también va incluida la cabecera de la rutina.
3. **Resultados:** Descripción de los resultados obtenidos, gráficas comparativas de los resultados obtenidos con los teóricos y comentarios sobre los mismos.
4. **Cuestiones:** Respuestas en papel a las cuestiones teóricas.

Esta documentación se le entregará al profesor de prácticas, en el correspondiente día de entrega de la práctica. En portada deberá incluirse los nombres de los alumnos.
Todos los ficheros necesarios para compilar la práctica y la documentación se guardarán en un único fichero comprimido, en formato zip, o tgz (tgz representa un fichero tar comprimido con gzip). El nombre de dicho fichero será **nombre_apellido.zip** o **nombre_apellido.tgz**. Donde **nombre_apellido** es el nombre y apellido de uno de los miembros de la pareja.
Adicionalmente, las prácticas deberán ser guardadas en algún medio de almacenamiento (lápiz usb, CD o DVD, disco duro, disco virtual remoto, etc) por el alumno para el día del examen de prácticas en Enero.

Ojo: Se recalca la importancia de llevar un lápiz usb **además de otros medios de almacenamiento como discos usb, cd, disco virtual remoto, email a dirección propia, etc**, ya que no se garantiza que puedan montarse y accederse todos y cada uno de ellos durante el examen, lo cual supondría la calificación de suspenso en prácticas.

Asimismo, una copia de los fuentes y la documentación deberán enviarse por el sistema electrónico de entrega de prácticas, como se indica a continuación.

Instrucciones para la entrega de los códigos de prácticas

La entrega de los códigos fuentes correspondientes a las prácticas de la asignatura AA se realizará por medio de la página web **https://moodle.uam.es/**.

Makefile

Con el objetivo de normalizar la compilación y ejecución de los diversos programas se sugiere utilizar la herramienta make junto al fichero Makefile incluido a continuación. En este fichero Makefile se implementarán las opciones ejercicio1, ejercicio2, ..., ejercicio5 que compilarán los programas de los diferentes apartados. Igualmente se implementarán las opciones ejercicio1_test, ejercicio2_test,..., ejercicio5_test que ejecutarán los programas.

```
#-----------------------
# IMPORTANTE: Los valores de los parametros de los ejercicio?_test deben ajustarse.
# Se asume que los ficheros fuente se llaman ejercicio1.c, ejercicio2.c,...,ordenar.h
#-----------------------

CC = gcc -ansi -pedantic
CFLAGS = -Wall
EXE = ejercicio1 ejercicio2 ejercicio3 ejercicio4 ejercicio5

all : $(EXE)

.PHONY : clean
clean :
rm -f *.o core $(EXE)

$(EXE) : % : %.o ordenacion.o tiempos.o permutaciones.o
@echo "#--------------------------"
```

```
@echo "# Generando $@ "
@echo "# Depende de $^"
@echo "# Ha cambiado $<"
$(CC) $(CFLAGS) -o $@ $@.o ordenacion.o tiempos.o permutaciones.o

permutaciones.o : permutaciones.c permutaciones.h
@echo "#--------------------------"
@echo "# Generando $@"
@echo "# Depende de $^"
@echo "# Ha cambiado $<"
$(CC) $(CFLAGS) -c $<

 ordenacion.o : ordenacion.c ordenacion.h
@echo "#--------------------------"
@echo "# Generando $@"
@echo "# Depende de $^"
@echo "# Ha cambiado $<"
$(CC) $(CFLAGS) -c $<

 tiempos.o : tiempos.c tiempos.h
@echo "#--------------------------"
@echo "# Generando $@"
@echo "# Depende de $^"
@echo "# Ha cambiado $<"
$(CC) $(CFLAGS) -c $<

ejercicio1_test:
@echo Ejecutando ejercicio1
@./ejercicio1 -limInf 1 -limSup 5 -numN 10

ejercicio2_test:
@echo Ejecutando ejercicio2
@./ejercicio2 -tamanio 1 -numP 5

ejercicio3_test:
@echo Ejecutando ejercicio3
@./ejercicio3 -tamanio 1 -numP 5

ejercicio4_test:
@echo Ejecutando ejercicio4
@./ejercicio4 -tamanio 1

ejercicio5_test:
@echo Ejecutando ejercicio5
@./ejercicio5 -num_min 1 -num_max 5 -incr 1 -numP 5 -fichSalida ejercicio5.log
```