# Estructuras de datos y algoritmos

Grados en Informática (UCM)

Final de junio — Curso 2016/2017

---

**Ejercicio 4 (2.25 puntos)** Se tienen dos vectores de enteros ordenados $a$ y $b$ con $n$ y $m$ elementos respectivamente. Se desea calcular el *beneficio* del vector $a$ con respecto a $b$. El beneficio del vector se calcula sumando los beneficios obtenidos por cada uno de sus elementos. El beneficio de un elemento $a[i]$ es el producto de la cantidad de elementos de $b$ que son estrictamente menores que $a[i]$ y la cantidad de elementos de $b$ que son estrictamente mayores que $a[i]$. Se pide:

1. (0,5 puntos) Especificar una función que dados dos vectores $a$ y $b$ como se indica en el párrafo anterior calcule el *beneficio* del vector $a$ con respecto a $b$.
2. (1 punto) Implementar como cuerpo de dicha función un algoritmo iterativo eficiente que resuelva el problema.
3. (0,5 puntos) Escribir el invariante que permite demostrar la corrección del algoritmo propuesto y una función de cota.
4. (0,25 puntos) Indicar y justificar adecuadamente el coste asintótico en el caso peor del algoritmo.

Los apartados 1, 3 y 4 pueden hacerse en papel o en el fichero con el código. Para el apartado 2, se completará el código del fichero `main4.cpp`, en el que ya está implementada la entrada/salida.

**Ejercicio 5 (1.5 puntos)** Se tienen dos vectores de enteros ordenados y distintos entre sí $a$ y $b$ con $n$ y $n-1$ elementos respectivamente. Los elementos de $b$ son los mismos que tiene $a$ excepto uno que falta. Se pide implementar un algoritmo recursivo eficiente que encuentre ese valor que falta. Se debe indicar la recurrencia y el coste asintótico en el caso peor del algoritmo.

Para resolver este ejercicio, se completará el código del fichero `main5.cpp`, en el que ya está implementada la entrada/salida.

**Ejercicio 6 (2 puntos)** Queremos añadir a la clase `set` un nuevo método

```
std::pair<bool,T> lower_bound(T const& e) const;
```

que dado un elemento `e` devuelva si existen en el conjunto elementos mayores o iguales que `e` y en caso afirmativo cuál es el menor de ellos.

Copia el fichero `set_eda.h` en `set_modificado.h` y haz las modificaciones en este fichero. Escribe al principio un comentario indicando cuáles han sido las modificaciones, justificando tu solución. Para probar el nuevo método, el programa principal resolverá varios casos. En cada caso, se leerá una serie de valores que se añadirán a un conjunto. Después se leerá otra serie de valores (preguntas) y para cada uno de ellos se escribirá su *lower bound*, si existe. Este código se encuentra en el fichero `main6.cpp`, que **no** puede ser modificado.

**Ejercicio 7 (4.25 puntos)** Se desea diseñar un TAD para gestionar los alumnos de los distintos profesores de una autoescuela (tanto alumnos como profesores se identifican por su nombre, que es un `string`). Para ello se desea disponer de las siguientes operaciones:

- constructora: al comienzo ningún profesor tiene asignados alumnos.
- `alta(A, P)`: sirve tanto para dar de alta a un alumno como para cambiarle de profesor. Si el alumno no estaba matriculado en la autoescuela se le da una puntuación de cero. Si ha cambiado de profesor, se le da de alta con el nuevo, con la puntuación que tuviera, y se le da de baja con el anterior. La puntuación determinará quién se puede examinar.
- `es_alumno(A, P)`: comprueba si el alumno `A` está matriculado actualmente con el profesor `P`.

- `actualizar(A, N)`: aumenta en una cantidad `N` la puntuación del alumno `A`. Si el alumno no está dado de alta con ningún profesor, entonces se lanza una excepción `domain_error` con mensaje `El alumno A no esta matriculado`.

- `examen(P, N)`: obtiene una lista con los alumnos del profesor `P`, ordenados alfabéticamente, que se presentarán a examen por tener una puntuación mayor o igual a `N` puntos.

Implementar de forma eficiente el TAD `autoescuela`, justificando la representación elegida e indicando la complejidad de las operaciones implementadas.

El programa principal resolverá varios casos de prueba. Para cada caso leerá una serie de operaciones (con sus argumentos) y las irá aplicando a una autoescuela inicialmente vacía. Todo el código aparece en el fichero `main7.cpp`, que **no** puede ser modificado.

## Normas de realización del examen

1. Debes programar soluciones para cada uno de los ejercicios, probarlas y entregarlas en el juez automático accesible en la dirección `http://exacrc/domjudge/team`.

2. Escribe comentarios que expliquen tu solución, justifiquen por qué se ha hecho así y ayuden a entenderla. Calcula la complejidad de todas las funciones que implementes.

3. En el juez te identificarás con el nombre de usuario y contraseña que has recibido al comienzo del examen. El nombre de usuario y contraseña que has estado utilizando durante la evaluación continua **no** son válidos.

4. Escribe tu **nombre y apellidos** en un comentario en la primera línea de cada fichero que subas al juez.

5. Descarga el fichero `http://exacrc/documEDG.zip` que contiene material que debes utilizar para la realización del examen (implementación de las estructuras de datos, ficheros con código fuente para completar y ficheros de texto con algunos casos de prueba de cada ejercicio del enunciado).

6. Tus soluciones serán evaluadas por el profesor independientemente del veredicto del juez automático. Para ello, el profesor tendrá en cuenta **exclusivamente** el último envío que hayas realizado de cada ejercicio.