

Nombre:
 Apellidos:
 DNI:

Estructuras de Datos y Algoritmos — Examen final extraordinario 2018/19
Grado en Desarrollo de Videojuegos. 2º V
Facultad de Informática, UCM

Instrucciones:

Esta primera parte del examen dura **50 minutos** y tiene una puntuación total de **3pt.**

No se puede usar el ordenador, móvil, calculadora o cualquier otro dispositivo electrónico.

Recordatorio

$\sum_{i=a}^n (s_i \pm t_i) = \sum_{i=a}^n s_i \pm \sum_{i=a}^n t_i$	$\sum_{i=a}^n k \cdot s_i = k \cdot \sum_{i=a}^n s_i$	$\sum_{i=a}^n i = \frac{(a+n)(n-a+1)}{2}$	$\sum_{i=0}^{n-1} k^i = \frac{1-k^n}{1-k}$
$(a+b)^2 = a^2 + b^2 + 2ab$	$(a-b)^2 = a^2 + b^2 - 2ab$		

1. (0.5 pt) Calcula el coste asintótico de la función f. Indica el coste de cada instrucción básica y cómo se combina para generar el coste de los bucles. Puedes usar el espacio libre a la derecha del código.

```

1 void f(vector<int> &v){
2   const int n = v.size();
3   int j, elem;
4
5   for (int i = 1; i < n; ++i) {
6     elem = v[i];
7     j = i - 1;
8     while (j >= 0 && v[j] > elem) {
9       v[j+1] = v[j];
10      j--;
11    }
12    v[j+1] = elem;
13  }
14 }
```

2. (0.25 pt) Obtén la recurrencia de coste $T(n)$ de la siguiente función recursiva. Puedes usar el espacio libre a la derecha del código. **No hace falta expandir la recurrencia y obtener el coste asintótico.**

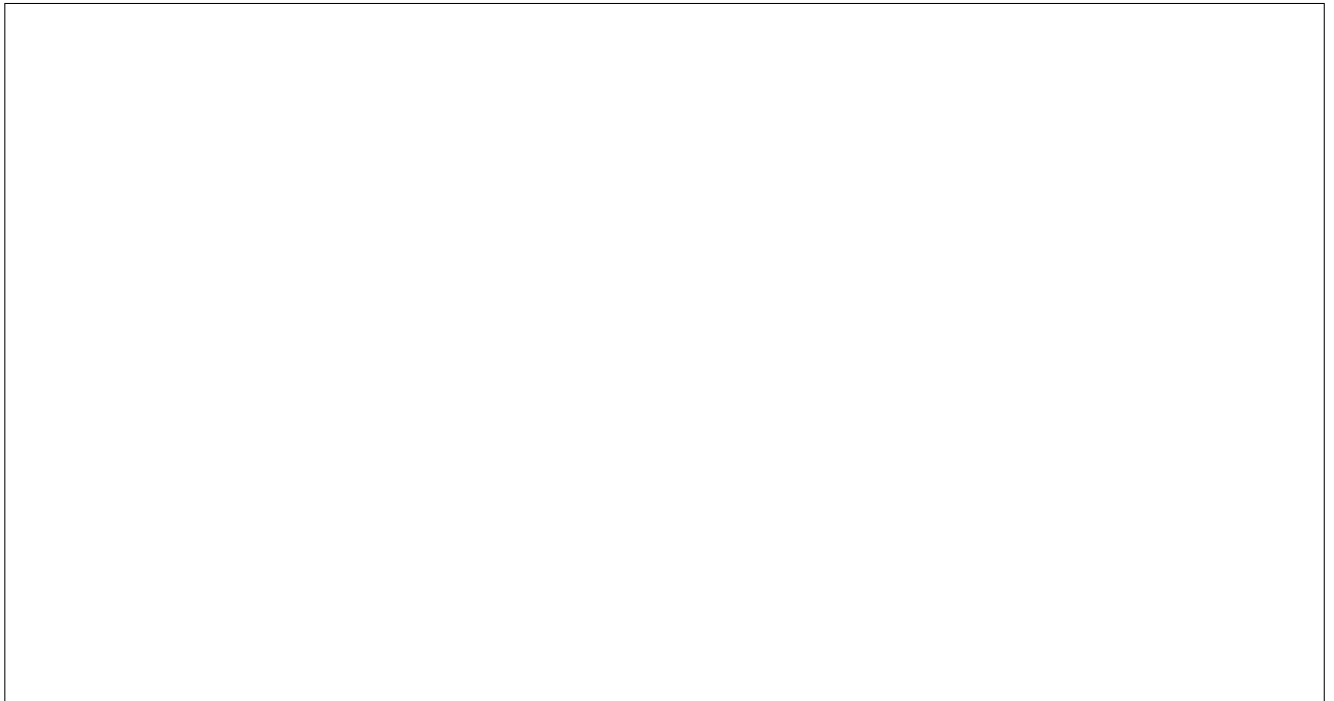
```

1 int numCifras(int n) {
2   int ret;
3   if (n < 10) {
4     ret = 1;
5   } else {
6     ret = 1 + numCifras(n / 10);
7   }
8   return ret;
9 }
```

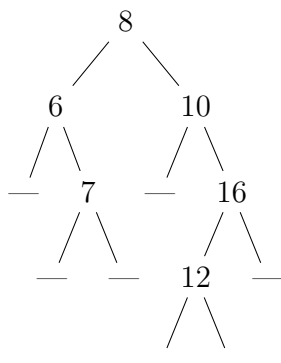
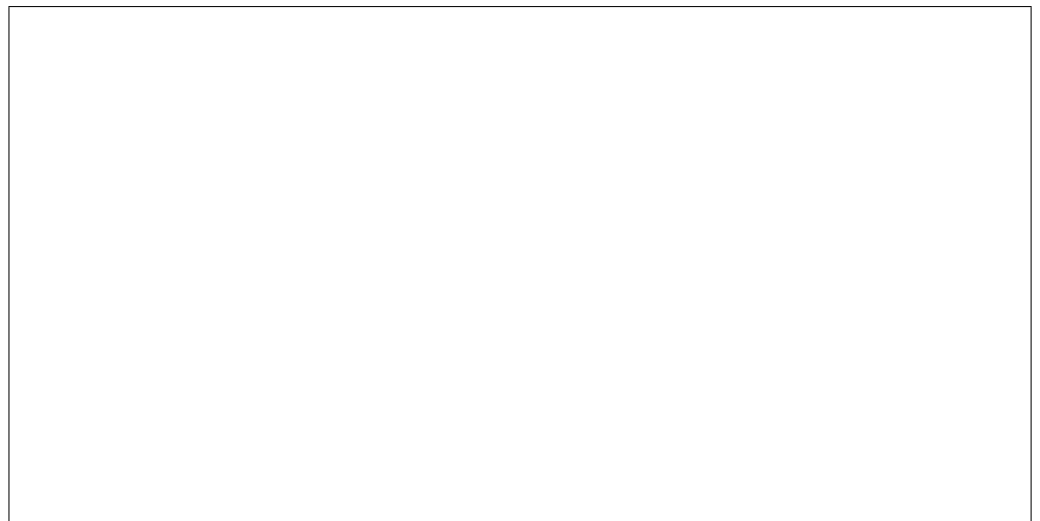
3. (1 pt) A partir de la siguiente recurrencia $T(n)$ que representa el coste de un algoritmo recursivo, utiliza el método de las expansiones para calcular en qué orden de complejidad está incluida $T(n)$. Se deben realizar **todos los pasos** e indicar claramente el resultado obtenido en cada uno de ellos.

$$T(n) = \begin{cases} 3 & \text{si } n = 0 \\ T(n-1) + 5n + 7 & \text{si } n > 0 \end{cases}$$

4. (0.75 pt) Explica la utilidad y funcionamiento de una *tabla de dispersión abierta*, incluyendo un dibujo que represente su organización de manera gráfica. Indica el coste en el caso promedio y peor de las funciones `get`, `contains`, `insert` y `erase`.



5. (0.5 pt) Indica las propiedades que debe cumplir un árbol para ser un *árbol binario de búsqueda* (BST). ¿El árbol de la figura es un árbol binario de búsqueda? Justifica tu respuesta.

Estructuras de Datos y Algoritmos — Examen final extraordinario 2018/19
Grado en Desarrollo de Videojuegos. 2º V
Facultad de Informática, UCM

Instrucciones:

- El fichero enviado debe contener una cabecera indicando vuestro nombre completo.
- Esta segunda parte del examen dura **2 horas y 10 minutos**.
- En el enlace «**Material para descargar**» dentro del juez tenéis un ZIP con las transparencias de la asignatura, el código de los TADs, las plantillas de solución del juez y otros documentos.
- En el enlace «**C++ reference**» dentro del juez (botón en la parte superior) tenéis disponible la documentación completa de la STL.
- Se valorará la calidad del código, la eficiencia, la inclusión del coste de las funciones y métodos involucrados y la corrección con respecto a los casos de prueba.
- Si no os da tiempo a terminar el ejercicio incluid un comentario lo más completo posible describiendo la idea de la solución y su coste. Aunque no haya nada de código, este comentario puede ser evaluable.

1. (3.5 pt) Consideremos un vector no vacío de números enteros. Se dice que un elemento es *pico* si es **mayor o igual** que sus vecinos anterior y posterior. En el caso de los elementos en los extremos únicamente existe un vecino, y el elemento de un vector unitario es el pico de manera trivial. Aplica la técnica algorítmica que mejor se adapte al problema («*divide y vencerás*» o «*vuelta atrás*») para desarrollar un programa que encuentre un elemento pico de un vector con coste $\mathcal{O}(\log n)$, donde n es el número de elementos en el vector. Será necesario incluir un comentario en el código con la **recurrencia** de coste completa.

Todo vector contiene al menos un elemento pico, pero además para este problema garantizamos que los vectores sobre los que se aplica tienen **exactamente un pico**, así que la solución es única.

Desarrollar una solución con coste mayor que $\mathcal{O}(\log n)$ podrá obtener un máximo de 1 punto, siempre y cuando se haya aplicado la técnica algorítmica adecuada y la recurrencia incluida sea correcta.

Entrada

La entrada consta de varios casos de prueba. Cada caso está compuesto por una línea que comienza con el número $N > 0$ de elementos en el vector seguido de N números separados por espacios que representan los elementos del vector. La entrada termina con un caso especial con el valor $N = 0$ que no debe procesarse.

Salida

Para cada caso de prueba la salida será una línea conteniendo el elemento pico del vector (que será único).

Ejemplo de entrada

3	0	2	4
3	4	2	0

1	-6							
7	8	10	20	15	12	11	10	
0								

Ejemplo de salida

4
4
-6
20

Estructuras de Datos y Algoritmos — Examen final extraordinario 2018/19
Grado en Desarrollo de Videojuegos. 2º V
Facultad de Informática, UCM

Instrucciones:

- El fichero enviado debe contener una cabecera indicando vuestro nombre completo.
- Esta segunda parte del examen dura **2 horas y 10 minutos**.
- En el enlace «**Material para descargar**» dentro del juez tenéis un ZIP con las transparencias de la asignatura, el código de los TADs, las plantillas de solución del juez y otros documentos.
- En el enlace «**C++ reference**» dentro del juez (botón en la parte superior) tenéis disponible la documentación completa de la STL.
- Se valorará la calidad del código, la eficiencia, la inclusión del coste de las funciones y métodos involucrados y la corrección con respecto a los casos de prueba.
- Si no os da tiempo a terminar el ejercicio incluid un comentario lo más completo posible describiendo la idea de la solución y su coste. Aunque no haya nada de código, este comentario puede ser evaluable.

2. (3.5 pt) Una biblioteca necesita un programa para gestionar sus fondos y los préstamos realizados. Para ello debe manejar dos tipos de acciones:

- Un **usuario saca en préstamo** un **libro** de los disponibles en la biblioteca.
- Un **usuario devuelve** un **libro** que tenía prestado.

Para simplificar, vamos a identificar cada libro de la biblioteca mediante un número natural y consideraremos que si hay varios ejemplares del mismo libro cada uno tendrá un identificador diferente. También identificaremos a los usuarios de la biblioteca mediante números naturales. El objetivo final de este programa será mostrar por pantalla cuántos libros hay disponibles en la biblioteca y cuántos tiene prestados cada usuario después de haber procesado una serie de acciones. Para ellos supondremos que partimos de un estado inicial donde todos los libros están en la biblioteca y ningún usuario tiene libros prestados.

En este problema es necesario utilizar **únicamente estructuras de la STL de C++**.

Entrada

La entrada consta de varios casos de prueba. Cada caso comienza con una línea de dos números naturales « $L\ A$ » ($1 \leq L \leq 10000, 1 \leq A \leq 1000$) indicando la cantidad de libros (L) en la biblioteca, además del número de acciones (A) a procesar. A continuación sigue una línea con L números naturales separados por un espacio representando los identificadores de los libros disponibles en la biblioteca (en cualquier orden). Finalmente, le siguen A líneas, cada una representando una acción:

- «**PRESTAR usuario libro**»: donde **usuario** es el identificador numérico del usuario y **libro** el identificador numérico del libro a prestar. Si el libro no está disponible en la biblioteca se debe reportar el error (ver el apartado de **Salida**).
- «**DEVOLVER usuario libro**»: donde **usuario** es el identificador numérico del usuario y **libro** el identificador numérico del libro. Si un usuario devuelve un libro que no tenía prestado se debe reportar el error (ver el apartado de **Salida**).

La entrada termina con un caso especial con valor «0 0» que no se debe procesar.

Salida

Para cada caso de prueba la salida será el recuento de los libros disponibles en la biblioteca y los que tiene prestado cada usuario tras procesar todas las acciones. Primero habrá una línea «**Biblioteca**

-> B», siendo B el número de libros disponibles en la biblioteca. Luego se mostrará la información de los usuarios, cada uno en una línea «usuario -> U», donde **usuario** es el identificador del usuario y U el número de libros que tiene prestados. Los usuarios se mostrarán en orden creciente de identificador, y se deben mostrar todos aquellos usuarios que hayan sacado algún libro, aunque en el momento final tengan 0 libros prestados. Tened en cuenta que antes y después del símbolo «->» hay un espacio.

En caso de que haya ocurrido algún error al prestar (el libro no estaba disponible) o al devolver (el usuario no tenía prestado el libro devuelto) únicamente se debe mostrar una línea con la palabra «ERROR».

Después de cada caso de prueba se debe dejar una línea en blanco.

Ejemplo de entrada

```
4 4
4 1 0 2
PRESTAR 500 0
PRESTAR 501 4
PRESTAR 500 2
DEVOLVER 501 4
4 2
0 1 2 4
PRESTAR 500 4
PRESTAR 501 4
4 1
0 1 2 4
PRESTAR 500 5
4 1
4 2 1 0
DEVOLVER 501 4
4 2
1 2 0 4
PRESTAR 500 4
DEVOLVER 500 0
0 0
```

Ejemplo de salida

```
Biblioteca -> 2
500 -> 2
501 -> 0

ERROR

ERROR

ERROR

ERROR
```