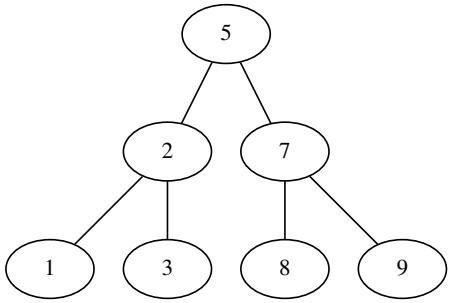
Nombre: .....
Apellidos: .....
DNI: .....

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

Instrucciones:
Esta primera parte del examen dura **1 hora** y tiene una puntuación total de **3.5pt**.
No se puede encender el ordenador ni utilizar calculadora.

1. (0.3pt) ¿El siguiente árbol binario es un BST? Justifica tu respuesta.



2. (0.3 pt) Compara el coste de los algoritmos de ordenación `mergesort()` y `quicksort()`.

3. (0.3 pt) Disponemos de dos algoritmos de ordenación: A con coste $O(n^2)$ y B con coste $O(n.log\ n)$. ¿Qué algoritmo es asintóticamente mejor? ¿Será más rápido para ordenar cualquier vector? Justifica tu respuesta.

4. (0.3 pt) ¿Qué funcionalidad tiene un objeto de la clase `shared_ptr<T>`? ¿Qué problemas resuelve en una estructura de datos? Pon un ejemplo.

5. (0.3 pt) Obtén la recurrencia de coste $T(n)$ de la siguiente función recursiva.

```
1 int f(int n) {
2   if (n <= 0)
3     return 0;
4   else {
5     return f(n-1) + n;
6   }
7 }
```

6. (2 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.

$$T(n) = \begin{cases} 3 & \text{si } n = 0, 1 \\ 2T(\frac{n}{2}) + 6 & \text{si } n > 1 \end{cases}$$

Recordatorio

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

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

Instrucciones:

- Esta segunda parte del examen dura **2 horas**.
- En «Publicación docente de ficheros» del escritorio tenéis disponibles las transparencias de la asignatura, el código de los TADs, las plantillas de solución del juez, la documentación de la STL, etc.
- Se valorará la calidad del código, la eficiencia, la inclusión del coste de las funciones/métodos involucrados y la corrección con respecto a los casos de prueba.

1. (3.5 pt) Consideremos un vector `v` de tamaño n. Diremos que un elemento de ese vector es *mayoritario* si aparece más de $\frac{n}{2}$ veces. Si los elementos tienen un orden `<`, encontrar el elemento mayoritario (o comprobar que no existe) es sencillo: basta con ordenar los elementos y realizar un recorrido contando secuencias de elementos consecutivos repetidos. Pero si no podemos utilizar ese orden la situación cambia.

En este ejercicio debes implementar una **plantilla de función** como esta:

```
template < typename T, typename Equal=equal_to<T> >
pair<bool,T> mayoria(const vector<T>& v)
```

La plantilla recibe el tipo `T` y el tipo functor `Equal` para comparar elementos de tipo `T` mediante igualdad. Por su parte, la función acepta como parámetro un vector de elementos de tipo `T` y devuelve una pareja `pair<bool,T>`: el primer valor indica si existe elemento mayoritario, y el segundo valor almacena dicho elemento mayoritario. Además de cumplir esta cabecera, la función `mayoria` debe:

- Tener un **coste inferior a** $O(n^2)$.
- No utilizar **ningún operador de orden** entre elementos de tipo `T`, únicamente el functor `Equal`.

Para resolver este problema debes aplicar la técnica algorítmica tratada en la asignatura («*divide y vencerás*» o «*vuelta atrás*») que mejor se adapte al problema. Para obtener la máxima nota es imprescindible implementar la plantilla y usar el functor `Equal`, aunque soluciones específicas para `vector<int>` que utilicen el operador `==` también recibirán puntos.

Entrada
La entrada consta de varios casos de prueba. Cada caso es una línea que comienza con un número $0 \leq N \leq 100{,}000$ indicando la longitud del vector. A continuación aparecen N números naturales en el rango $[0..2^{30}]$ separados por un espacio. La entrada termina con un caso especial con el valor de $N = -1$ que no debe procesarse.

Salida
Para cada caso de prueba la salida será una línea. Si no existe elemento mayoritario, se mostrará la palabra `NADA`. En otro caso, se mostrará dicho elemento mayoritario.

Ejemplo de entrada

```
4 1 2 3 4
1 6
8 3 0 3 3 3 2 2 3
8 3 0 3 3 3 2 2 2
0
-1
```

Ejemplo de salida

```
NADA
6
3
NADA
NADA
```

2. (3 pt) Utiliza **de manera adecuada** un **diccionario de la STL** para implementar el inventario de un videojuego. Los elementos del inventario tiene un nombre, y por cada elemento podemos almacenar únicamente un número máximo de unidades M (ese valor es el mismo para todos los elementos). El jugador puede interaccionar de 3 maneras con el inventario:

- Añadir una unidad del elemento cuando el personaje lo encuentra.
- Gastar una unidad del elemento cuando el personaje lo utiliza durante el juego.
- Aplicar el truco con las teclas «abajo-atrás → abajo → abajo-delante → puño + patada». Este truco introducido por los programadores aumenta en 5 unidades **el último elemento añadido**. Sin embargo, debido a un *bug*, el contador de ese elemento se pone a 0 si al realizar ese aumento las unidades sobrepasan el máximo M.

El objetivo de este ejercicio es procesar una secuencia de interacciones con el inventario y mostrar el número de unidades de cada elemento en el inventario.

Entrada
La entrada consta de varios casos de prueba. Cada caso comienza con una línea indicando el número $1 \leq N \leq 10000$ de interacciones a procesar y el máximo $1 \leq M \leq 10000$ de unidades de cada elemento que el inventario puede amacenar. N y M están separados por un espacio. A continuación le siguen N líneas representando las interacciones. Estas líneas tienen el siguiente formato:

- `ADD elemento` para añadir una unidad de `elemento`. No produce ningún cambio si ya se disponían de M unidades de dicho elemento.
- `USE elemento` para gastar una unidad de `elemento`. Se garantiza que siempre que se use un elemento se dispondrá de al menos una unidad.
- `APP CHEAT` para aplicar el truco. Se garantiza que en algún momento anterior se ha añadido un elemento al inventario.

El nombre `elemento` será una cadena de letras minúsculas de hasta 100 caracteres. La entrada termina con un caso especial con el valor de $N = M = 0$ que no debe procesarse.

Salida
Para cada caso de prueba la salida será una serie de líneas `elemento unidades` (separado con exactamente un espacio) mostrando el estado final del inventario. Se deben mostrar todos los elementos añadidos en algún momento, aunque en el estado final se tengan 0 unidades, y los elementos se deben mostrar en **orden alfabético**. Al final de cada caso se debe insertar una línea en blanco.

Ejemplo de entrada

```
5 10
ADD botella
ADD botella
ADD casco
USE botella
APP CHEAT
4 8
ADD zirconio
APP CHEAT
APP CHEAT
ADD pluma
0 0
```

Ejemplo de salida

```
botella 1
casco 6

pluma 1
zirconio 0
```