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

Estructuras de Datos y Algoritmos — Examen final 1^{er} cuatrimestre
Grado en Desarrollo de Videojuegos. 2^{o} 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) Indica el coste asintótico de las siguientes dos funciones y compáralo.

```
1  int suma1(int n) {
2    int suma = 0;
3    for (int i = 0; i < n; ++i )
4      for (int j = 0; j < n; ++j )
5        suma += 1;
6    return suma;
7  }
8
9  int suma2(int n) {
10   int suma = 0;
11   for (int i = 0; i < n; ++i )
12     for (int j = 0; j < i; ++j )
13       suma += 1;
14   return suma;
15 }
```

2. (0.3 pt) ¿Qué coste tiene la siguiente función en el caso peor?

```
1  int sumaCuadrados(size_t n) {
2    int ret = 0;
3    if (n > 0)
4      ret = n*n + sumaCuadrados(n-1);
5    return ret;
6  }
```

3. (0.2 pt) ¿Qué coste tiene el algoritmo de ordenación quicksort en el caso peor?

4. (0.2 pt) Consideremos un árbol binario de búsqueda (BST) implementado mediante la clase bst, es decir, con nodos enlazados mediante shared_ptr. ¿Qué coste tiene el método **void** remove(**const** T& e) en el caso peor?

5. (0.5 pt) ¿Ocurre algún error al compilar y ejecutar el siguiente código que define y utiliza una clase Entero? Justifica tu respuesta.

```
1  class Entero {
2    private:
3      int* elem;
4    public:
5      Entero(int n) : elem(new int(n)) {}
6      void incrementa() {*elem = *elem + 1;}
7  };
8  int main() {
9    Entero a(5);
10   Entero b = a;
11   b.incrementa();
12   return 0;
13 }
```

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.

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

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 1^{er} cuatrimestre
Grado en Desarrollo de Videojuegos. 2^{o} A
Facultad de Informática, UCM

Instrucciones:

- Esta segunda parte del examen dura **2 horas**.
- En «Publicación docente de ficheros» del escritorio tenéis disponible 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) Un fábrica dispone de N máquinas y tiene N empleados contratados. Cada máquina está muy automatizada, así que únicamente necesita que un empleado se encargue de ella. Además, al inicio de cada día se negocia qué maquinas puede manejar cada empleado, además de fijar qué sueldo percibirá cada empleado al manejar cada máquina (puede ser diferente de una máquina a otra).

Al empezar cada día, el dueño de la fábrica debe enviar a los empleados a las distintas máquinas para comenzar la producción. Sin embargo, no le interesa cualquier asignación sino que quiere minimizar la suma de sueldos que debe pagar a sus empleados. Además, si detecta que con las restricciones impuestas no es posible asignar un empleado a cada máquina entonces establecerá el día como festivo y cerrará la fábrica.

Implementa una solución a este problema siguiendo una de las dos técnicas algorítmicas tratadas en la asignatura: *divide y vencerás* o *vuelta atrás*.

Entrada
La entrada consta de varios casos de prueba. Cada caso comienza con una línea indicando el número $0 < N \leq 15$ de máquinas y empleados. A continuación le siguen N líneas, cada una mostrando los sueldos de cada empleado i en orden creciente de 1 a N. Para cada empleado i, la entrada contiene una secuencia de N números separados por espacios ($S_{i1}\ S_{i2} \ldots S_{iN}$) donde $0 \leq S_{ij} \leq 100$ indica el sueldo que cobrará el empleado i al manejar la máquina j. Si algún $S_{ij} = 0$ entonces el trabajador i no puede manejar la máquina j ese día. La entrada termina con un caso especial con el valor de $N = 0$ que no debe procesarse.

Salida
Para cada caso de prueba la salida será una línea. Si no es posible encontrar alguna asignación de empleados a máquinas, la línea contendrá la palabra `FESTIVO`. En otro caso, mostrará la mínima suma de sueldos que el dueño tiene que pagar para asignar una máquina a cada empleado.

Ejemplo de entrada

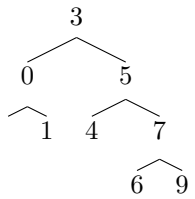
```
1
18
2
3 8
2 10
2
5 0
7 6
3
3 0 0
8 0 0
1 2 3
0
```

Ejemplo de salida

```
18
10
11
FESTIVO
```

2. (3 pt) Sabemos que a partir de único un recorrido `preorden`, `inorden` o `postorden` no es posible reconstruir el árbol binario original. Sin embargo, esto cambia si nos proporcionan el recorrido `inorden` además del recorrido `postorden`. A partir de los dos recorridos sí tenemos información suficiente para reconstruir el árbol original:

```
Inorden:   0 1 3 4 5 6 7 9
Postorden: 1 0 4 6 9 7 5 3
```



En este ejercicio debes implementar y utilizar una función paramétrica que reconstruya un árbol binario a partir de sus recorridos `inorden` y `postorden`. La función debe tener la siguiente cabecera:

```
template <typename T>
bintree<T> reconstruye(const vector<T>& inorden, const vector<T>& postorden);
```

La función `reconstruye` debe implementarse de manera recursiva, aunque puede utilizar funciones auxiliares si lo consideráis necesario (por ejemplo, una versión más general `reconstruye_rec`).

Una vez reconstruido el árbol binario, se debe calcular y mostrar su altura.

Entrada

La entrada consta de varios casos de prueba. Cada caso comienza con una línea indicando la longitud $N \geq 0$ de cada uno de los recorridos. A continuación siguen dos líneas: la primera contiene una secuencia de números enteros en el rango $[-(2^{30}) \dots 2^{30}]$ separados por un espacio representando el recorrido `inorder` del árbol binario, y la segunda línea contiene el recorrido `postorden` siguiendo el mismo formato. Tened en cuenta que un árbol vacío ($N = 0$) presentará dos recorridos vacíos, cada uno en una línea. 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 conteniendo la altura del árbol binario reconstruido.

Ejemplo de entrada

```
1
5
5
2
0 3
0 3
3
0 3 5
0 5 3
8
0 1 3 4 5 6 7 9
1 0 4 6 9 7 5 3
-1
```

Ejemplo de salida

```
1
2
2
4
```