Asignatura: Fundamentos de algoritmia
Evaluación continua. Cuestionario.

Grupo A. Azul
Profesor: Isabel Pita.

Nombre del alumno:

1. Tenemos una máquina capaz de realizar 10^8 operaciones por segundo en la que ejecutamos programas con distinto orden de complejidad. Los programas deben ejecutarse en un tiempo lo más cercano posible a un segundo. Relaciona cada orden de complejidad con uno de los tamaños de los datos de entrada dados en la siguiente tabla de forma que todos ellos se ejecuten lo más cerca posible de un segundo.

Complejidad	Tamaño entrada
$\mathcal{O}(n)$	30
$\mathcal{O}(n^2)$	8
$\mathcal{O}(3^n)$	10.000
$\mathcal{O}(n^n)$	100.000.000
$\mathcal{O}(2^n)$	15

Respuesta:

b) Si el algoritmo tiene complejidad $\mathcal{O}(n)$, se utiliza una entrada entre 100.000.000 datos. Aunque normalmente no son aceptables entradas de más de 1.000.000 por problemas de espacio.

a) Si el algoritmo tiene complejidad $\mathcal{O}(n^2)$, se utiliza una entrada entre 1.000 y 10.000 datos, porque $(10^4)^2 = 10^8$.

e) Si el algoritmo tiene una complejidad $\mathcal{O}(3^n)$ se puede llegar hasta los 15 datos.

d) Si el algoritmo tiene una complejidad $\mathcal{O}(n^n)$, no admite más de 8 o 10 datos. En algunos problemas de vuelta atrás se puede llegar a las 15.

c) Si el algoritmo tiene una complejidad $\mathcal{O}(2^n)$, se utilizan entradas de unos 30 datos, aunque en los algoritmos de vuelta atrás se pueden conseguir entradas de hasta 100 datos con un tiempo aceptable.

2. Dadas las siguientes funciones de coste, indica su orden de complejidad más ajustado (representado por la función más sencilla) y ordena los órdenes de complejidad obtenidos utilizando $\subset$.

a) $f_1(n) = 2^n + 3^n + 4^n + n^n$

b) $f_2(n) = \frac{1}{10}n + \log^2 n$

c) $f_3(n) = n\log_2^2 n + n\log_5 n + \log_{10} n$

d) $f_4(n) = n\log n + n$

e) $f_5(n) = \log_3 5n + \log_{10} 10n + \log_2 n^2$

f) $f_6(n) = \log_3 10$

Respuesta:

a) $f_1(n) = 2^n + 3^n + 4^n + n^n \in \mathcal{O}(n^n)$.

b) $f_2(n) = \frac{1}{10}n + \log^2 n \in \mathcal{O}(n)$.

c) $f_3(n) = n\log_2^2 n + n\log_5 n + \log_{10} n \in \mathcal{O}(n\log^2 n)$.

d) $f_4(n) = n\log n + n \in \mathcal{O}(n\log n)$.

e) $f_5(n) = \log_3 5n + \log_{10} 10n + \log_2 n^2 \in \mathcal{O}(\log n)$.

f) $f_6(n) = \log_3 10 \in \mathcal{O}(1)$.

$\mathcal{O}(1) \subset \mathcal{O}(\log n) \subset \mathcal{O}(n) \subset \mathcal{O}(n\log n) \subset \mathcal{O}(n\log^2 n) \subset \mathcal{O}(n^n)$.

3. Indica la complejidad en tiempo más ajustada del siguiente algoritmo y justifícala indicando el número de vueltas que da cada bucle y el coste de cada vuelta.

```
int n, m; std::cin >> n >> m;
for (int i = 0, j = 0; i < n && j < m; ++i, ++j)
        x += v[i][j];
}
```

Respuesta:

El coste del cuerpo del bucle es constante. El número de vueltas del bucle es el mínimo entre los valores de n y m. Por lo tanto el coste es $\mathcal{O}(min(n, m))$.

4. Indica la complejidad en tiempo más ajustada del siguiente algoritmo y justifícala indicando el número de vueltas que da cada bucle y el coste de cada vuelta.

```
int n, m; std::cin >> n >> m;
for (int i = 0, j = m-1; i < n || j >= 0; ++i, --j)
        x += i + j;
}
```

Respuesta:

El coste del cuerpo del bucle es constante. El número de vueltas del bucle es el máximo entre los valores de n y m. Por lo tanto el coste es $\mathcal{O}(max(n, m))$.

5. Indica la complejidad en tiempo más ajustada del siguiente algoritmo, y justifícala indicando el número de vueltas que da el bucle y el coste de esa vuelta.

```
int n; std::cin >> n;
for (int i = n; i > 0; i = i/2)  ++x;
```

Respuesta:

La variable de control del bucle se divide entre dos en cada vuelta del bucle, por lo que el bucle alcanza el valor n en $\log n$ vueltas. El coste de cada vuelta del bucle es constante porque consiste en incrementar una variable. Por lo tanto el coste del bucle completo pertenece al orden $\mathcal{O}(\log n)$.

6. Indica la complejidad en tiempo más ajustada del siguiente algoritmo, y justifícala indicando el número de vueltas que da el bucle y el coste de cada vuelta.

```
int n; std::cin >> n;
int i = 1; int k = 3; int cont = 0;
while (i < n) {
  ++cont;
  i *= k;
}
```

Respuesta:

La variable de control del bucle se multiplica por k en cada vuelta del bucle, por lo que el bucle alcanza el valor n en $\log_3 n$ vueltas. El coste de cada vuelta del bucle es constante porque consiste en incrementar una variable. Por lo tanto el coste del bucle completo pertenece al orden $\mathcal{O}(\log n)$.

7. Dado el problema de contar el número de veces que aparece una palabra en una array de cadenas de caracteres, resuelto con la siguiente función:

```
int contar (std::vector<std::string> const& v, std::string b) {
     int cont = 0;
     for (std::string const& str : v)  {
            if (str == b) ++cont;
   }
   return cont;
}
```

a) Indica qué complejidad tiene la operación == sobre las cadenas de caracteres en el caso peor.

b) Indica qué complejidad tiene el programa en el caso peor en función de los valores de entrada.

Respuesta:

a) `==` tiene complejidad lineal en el número de caracteres de la cadena.

b) $\mathcal{O}(n * m)$ siendo n el número de cadenas de caracteres del vector y m el tamaño de la cadena de caracteres mas larga.

8. Dado el problema de buscar un elemento en una matriz, resuelto con la siguiente función genérica:

```
template <class T>
bool buscar (std::vector<std::vector<T>> const& m, T const& b) {
     for (int i = 0; i < m.size(); ++i)
           for (int j = 0; j < m[i].size(); ++j)
                if (m[i][j] == b) return true;
   return false;
}
```

a) Indica cómo puede afectar el tipo al que se instancie la matriz a la complejidad del algoritmo.

b) Indica qué complejidad tiene la función `buscar` en función del número de elementos de la matriz.

c) Indica qué complejidad tiene la función `buscar` en función del número de filas y columnas de la matriz.

Respuesta:

a) Debe tenerse en cuenta la complejidad de la comparación entre los dos valores del tipo genérico. Por ejemplo si la matriz se instancia a una cadena de caracteres la comparación tendrá coste lineal en el número de caracteres.

b) $\mathcal{O}(n * k)$, siendo n el número de elementos de la matriz y k el coste de comparar los elementos.

c) $\mathcal{O}(n * m * k)$ siendo n el número de filas de la matriz y m el número de columnas. k es el coste de comparar los elementos.