1. ¿Cuál es el objetivo de la etapa de análisis en el Diseño y Análisis de un algoritmo?
   a. Determinar el lenguaje y herramientas disponibles para su desarrollo.
   b. Estimar los recursos que consumirá el algoritmo una vez implementado.
   c. Estimar la potencia y características del equipo informático necesarios para el correcto funcionamiento del mismo.

2. ¿El tiempo de ejecución de un algoritmo depende de la talla del problema?

a. Sí, siempre.
b. No, nunca.
c. No necesariamente.

3. El estudio de la complejidad resulta realmente interesante para tamaños grandes de problema por varios motivos:
a. Las diferencias reales en tiempo de compilación de los algoritmos con diferente coste para tamaños pequeños del problema no suelen ser muy significativas.
b. Las diferencias reales en tiempo de ejecución de algoritmos con diferente coste para tamaños grandes del problema no suelen ser muy significativas.
c. Ninguna de las anteriores.

4. ¿Por qué se emplean funciones de coste para expresar el coste de un algoritmo?
a. Para poder expresar el coste de los algoritmos con mayor exactitud.
b. Para que la expresión del coste del algoritmo sea válida para cualquier entrada del mismo.
c. Para poder expresar el coste de un algoritmo mediante una expresión matemática.

5. La complejidad temporal en el mejor de los casos:
a. Es el tiempo que tarda el algoritmo en resolver la talla más pequeña que se le puede presentar.
b. Es una función de la talla que tiene que estar definida para todos los posibles valores de esta.
c. Las demás opciones son verdaderas.

6. El caso base de una ecuación de recurrencia asociada a la complejidad temporal de un algoritmo expresa:
a. El coste de dicho algoritmo en el mejor de los casos.
b. El coste de dicho algoritmo en el peor de los casos.
c. Ninguna de las anteriores.

7. Indica cual es la complejidad de la siguiente función:

```
unsigned sum( const mat &a ){
     unsigned d = A.n_rows();
     unsigned a = 0;
     for(unsigned i = 0; i < d; i++)
          for(unsigned j = 0; j < d; j++)
               a += A(i, j)
     return a;
}
```

a. O(n log n)
b. $O(n^2)$
c. O(n)

8. Si la complejidad temporal de la función F2 $\in \Theta(a^2)$ entonces la complejidad temporal del siguiente algoritmo es:

```
funcion suma(a:entero)
     m := 5*a
     para i:= 1 hasta m
          para j := 1 hasta m
               F2(a)
          fpara
     fpara
ffuncion
```

a. a^4
b. a^2
c. a^3

9. El coste temporal asintótico del fragmento

```
s = 0;
for(i = 0; i < n; i++)
     for(j = i; j < n; j++)
          s += i*j;
```

y del fragmento

```
s = 0;
for(i = 0; i < n; i++)
     for(j = 0; j < n; j++)
          s += i*i*j;
```

a. ... el del primero menor que el del segundo.
b. ... el del segundo menor que el del primero.
c. ... iguales.

10. El coste asintótico de la siguiente función es:

```
funcion factorial(int n){
     if(n == 0) return n;
     else return n*factorial(n-1);
}
```

a. $\Theta(n)$.
b. $\Theta(n^2)$.

c. $\Theta(\log(n))$.

11. Sea f(n) la solución de la relación de recurrencia:

$$f(n) = 2f(n/2) + 1$$
$$f(1) = 1$$

Indica cual de las siguientes expresiones es cierta:
a. f(n) pertenece a $\Theta(n \log(n))$.
b. f(n) pertenece a $\Theta(n^2)$.
c. f(n) pertenece a $\Theta(n)$.

12. La complejidad de la función TB es:


```
int tb(const vector<int> &A, int iz, int de){
        int n, i,;
        n = de - iz + 1;
        if(n < 1) return 0;
        else if (n == 1) return 1;
        else if(A[iz] == A[de]) return tb(A, iz + 1, de - 1) + 1;
        else return tb(A, iz + 1, de - 1);
}
```

a. $\Theta(n)$
b. $\Theta(n \log(n))$
c. $\Theta(n^2 \log(n))$

13. ¿Pertenece $3n^2 + 3$ a $O(n^3)$?
a. Si.
b. Solo para c = 1, n0 = 5.
c. No.

14. Indique cual de las siguientes expresiones es cierta.
a. $O(n^2) \subset O(2^{\log(n)}) \subset O(2^n)$.
b. $O(2^{\log(n)}) \subset O(n^2) \subset O(2^n)$.
c. $O(n^2) \subset O(2^{\log(n)}) \subset O(2^n)$.

15. ¿Cuál de las siguientes jerarquías es correcta?
a. $O(1) \subset O(\log(n)) \subset O(\log(\log(n)))$.
b. $O(n!) \subset O(2^n) \subset O(n^n)$.
c. $O(2^n) \subset O(n!) \subset O(n^n)$.

16. Ordena de menor a mayor las siguiente complejidades.

1. $O(1)$
2. $O(n^2)$
3. $O(n\log(n))$
4. $O(n!)$

   a. 3, 1, 2 y 4.
   b. 1, 3, 2 y 4.
   c. 1, 3, 4 y 2.

17. ¿Cuál de los siguientes algoritmos de ordenación tiene menor complejidad?
a. Burbuja.
b. Inserción directa.
c. Mergesort.

18. Un problema de tamaño n puede transformarse en tiempo O(n) en siete n/7, por otro lado, la solución al problema cuando la talla es 1 requiere un tiempo constante. ¿Cuál de estas clases de coste temporal asintótico es la más ajustada?
a. $O(n^2)$
b. $O(n)$
c. $O(n \log n)$

19. La versión de quicksort que utiliza el pivote el elemento del vector que ocupa la posición central...
a. ... se comporta mejor cuando el vector ya esta ordenado.
b. ... se comporta peor cuando el vector ya esta ordenado.
c. ... no presenta casos mejor y peor distintos para instancias del mismo tamaño.

20. Indica cual es la complejidad, en función de n, del fragmento siguiente:

```
int a = 0;
for(int i = 0; i < n; i++)
    for(int j = i; j > 0; j /= 2)
        a += A[i][j];
```

a. $O(n \log n)$
b. $O(n^2)$
c. $O(n)$

21. ¿Cuál es la complejidad temporal de la siguiente función recursiva?

```
unsigned desperdicio (unsigned n){
    if(n <= 1)
        return 0;
    unsigned sum = desperdicio(n/2) + desperdicio(n/2);
    for(unsigned i = 1; i < n - 1; i++)
        for(unsigned j = 1; j <= i; j++)
            for(unsigned k = 1; k <= j; k++)
                sum += i*j*k;
    return sum;
}
```


a. $\Theta(n^3 \log n)$
b. $\Theta(n^3)$
c. $\Theta(2^n)$

22. Dada la siguiente relación de recurrencia. ¿Qué cota es verdadera?

$f(n) = 1$ si $n = 1$;
$f(n) = n + 3f(n/3)$ si $n > 1$.

a. f(n) pertenece a Θ (n)
b. f(n) pertenece a Θ (n^3)
c. f(n) pertenece a Θ (n log n)

23. Un algoritmo recursivo basado en el esquema de divide y vencerás:
a. Será más eficiente cuantos más equitativa sea la división en subproblemas.
b. Nunca tendrá una complejidad exponencial.
c. Las demás opciones son verdaderas.

24. En cuál de los siguientes casos no se puede aplicar el esquema de Divide y Vencerás:
   a. Cuando los subproblemas son de tamaños muy diferentes.
   b. Cuando el problema no cumple el principio de optimalidad.
   c. Se puede aplicar en ambos casos.