Pregunta 1:   
  
Escribe la especificación formal de un TAD Natural que, además de las operaciones constructoras, disponga de las operaciones serPar, serImpar, suma, producto y factorial.

Pregunta 2:   
  
Dada una función de Fibonacci implementada recursivamente, i) escribe su complejidad computacional temporal en el peor de los casos, en el mejor de los casos, y el caso promedio, ii) explica qué estrategia algorítmica sigue esa implementación y iii) si invocamos Fibonacci(5), ¿cuántas veces se llama la función a sí misma (di el número total y luego sepáralo por casos en función del parámetro de entrada de la función). Nota: no escribas la función, simplemente responde a los apartados anteriores.

Pregunta 3:   
  
El siguiente código muestra un TAD Lista con implementación doblemente enlazada.  
  
typedef int TIPOELEM;  
  
typedef struct celda {TIPOELEM elemento;  
  
 struct celda \* ant,sig;} TCELDA;  
  
typedef TCELDA \* POSICION;  
  
typedef struct lista {  
  
 int longitud;  
  
 POSICION fin;} \* TLISTA;  
  
i) Escribe la implementación en C de la función recupera  
  
void recupera(TLISTA l, POSICION p, TIPOELEM \*e)  
  
{  
  
}  
  
ii) Escribe la implementación en C de la función crea  
  
void crea(TLISTA \*l)  
  
{  
  
}  
  
iii) Escribe la complejidad computacional temporal en el peor de los casos de la función recupera  
  
iv) Escribe la complejidad computacional temporal en el peor de los casos de una función que destruyese toda la lista

Pregunta 4:   
  
En una lista queremos buscar 4 personas que cumplan los requisitos para configurar un tribunal. Cada persona tiene que tener al menos 10 años de experiencia profesional y, además, tiene que haber equilibrio de género en el tribunal generado (50% hombres y 50% mujeres). Explica qué complejidad temporal en el mejor y en el peor de los casos tendría una función que implementase ese proceso de la forma más eficiente posible. No escribas la función en C, simplemente explica qué complejidad temporal tendría en el mejor y en el peor de los casos.

Pregunta 5:   
  
Shin Chan es un díscolo e irreverente niño japonés interesado en las construcciones con piezas Lego. Quiere hacer una pirámide con piezas Lego 2x2. Cada planta de la pirámide va creciendo en tamaño, en el tope de la pirámide hay una única pieza, en el segundo nivel hay 4, en el tercer nivel hay 9 y así sucesivamente.  
  
La cuestión es que Shin Chan quiere saber cuántas piezas necesitaría para construir una pirámide de cualquier número de alturas (en cada planta el interior de la pirámide está totalmente lleno de piezas, no hay huecos).  
  
Shin Chan no está motivado para trabajar en resolver este problema, su único interés se centra en sus soeces actividades diarias. Por ello, contrata a Boo, una avezada programadora de aplicaciones, que escribe una función recursiva que, dado el número de alturas, determina el número total de piezas que Shin Chan necesita. Escribe la función que Boo crearía para ayudar a Shin Chan.

Pregunta 6:   
  
Una institución está organizando un curso de formación para el que tiene sólo un determinado cupo de plazas. Recibe peticiones por orden de llegada pero sólo las personas que cumplen ciertos criterios (edad >18 y <65 y nota en un test previo de idoneidad >5) son admisibles. Escribe una función en C que reciba el cupo máximo de admisibles (número entero) y una cola donde se han almacenado las peticiones y escriba por pantalla las personas que pueden incorporarse al curso. Cada persona se representa por su nombre, DNI, edad y nota en el test de idoneidad. Antes de escribir la función escribe el TIPOELEMENTO que tendría el TAD Cola. Abajo figuran las especificaciones de las operaciones del TAD Cola que puedes usar.  
  
typedef void \* TCOLA;  
  
void ColaVacia (TCOLA \*q);  
  
void AnadirCola (TCOLA \*q, TELEMENTO e);  
  
int EsColaVacia (TCOLA q);  
  
void EliminarCola (TCOLA \*q);  
  
void PrimeroCola (TCOLA q , TELEMENTO \*e);

Pregunta 7:   
  
El gráfico de debajo muestra la complejidad computacional de una solución divide y vencerás en función de los parámetros k, c e i.  
  
-Si k<c^i entonces t(n) ∈ O(n^i)  
  
-Si k=c^i entonces t(n) ∈ O(n^i log n)  
  
-Si k>c^i entonces t(n) ∈ O(n^(log\_c k))  
  
a) Explica el significado de k, c e i (indicando también si queremos que cada uno de ellos tenga un valor alto o bajo y por qué)  
  
b) En las dos variantes de multiplicación de enteros grandes (x multiplicado por y) descritas debajo, indica cual es más eficiente, justificando porqué en referencia a sus parámetros divide y vencerás.   
  
Variante 1  
  
xy=aci^n+(ad+bc)i^(n/2)+bd  
  
Variante 2  
  
xy=aci^n+((a-b)(d-c)+ac+bd)i^(n/2)+bd  
  
c) Compara esta complejidad computacional con la que tendríamos con una búsqueda secuencial con y sin centinela.