Práctica 6

1. Una expresión dada puede contener paréntesis abiertos, cerrados y algunos otros caracteres opcionales. Ningún otro operador estará presente en la cadena. Se debe generar un algoritmo que, dada una expresión muestre todas las expresiones válidas que se puedan conseguir eliminando el número mínimo de paréntesis para que la cadena de entrada sea válida. Ejemplos:

Entrada: `str = ()())()`
Salida: `()()() (())()`

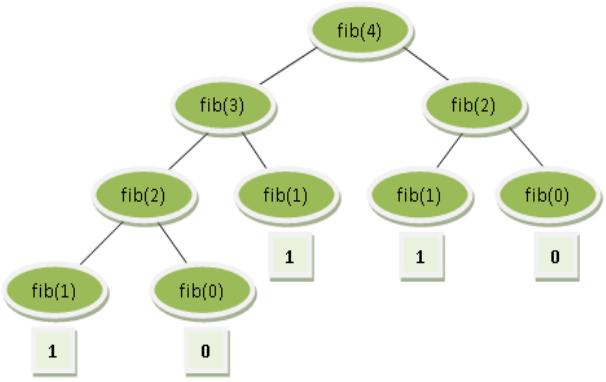
Es decir, existen dos posibles soluciones

Entrada: `str = (v)())()`
Salida: `(v)()() (v())()`

2.
Esta secuencia tiene los dos primeros valores 0 (cero) y 1 (uno), y los siguientes valores serán la suma de los dos valores precedentes. Por definición, la fórmula para encontrar cualquier número de Fibonacci es:

```
fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2);
```

Una forma de encontrar números de Fibonacci es por llamadas recursivas. Esto se ilustra a continuación, presentando el árbol de derivación cuando se calcula $fib(4)$, es decir, el quinto valor de esta secuencia:



En este caso,

- fib(4) = 1+0+1+1+0 = 3
- Se realizaron 8 llamadas recursivas.

Se pide que escriba un algoritmo tal que dado un N, devuelva $fib(N)$ y la cantidad de llamadas recursivas realizadas.

3. Definimos el algoritmo de ordenación *Mergesort* de la siguiente manera:

- Si la longitud de la lista es 0 ó 1, entonces ya está ordenada.

- Si la longitud de la lista es mayor que 1, se divide la lista desordenada en dos sublistas de aproximadamente la mitad (una diferencia de 1 como máximo) del tamaño de la original.
  - **a)** Ordenar cada sublista recursivamente aplicando el *Mergesort*.
  - **b)** Mezclar las dos sublistas en una sola lista ordenada.

Se pide que realice una implementación del *Mergesort*.

4. Supongamos que disponemos de n archivos f_1, f_2 , ... , f_n con tamaños l_1, l_2, ..., l_n, y un pendrive de capacidad $d < l_1 + l_2 + \ldots + l_n$.

a) Queremos maximizar el número de ficheros que ha de contener el pendrive, y para eso ordenamos los ficheros por orden creciente de su tamaño y vamos agregando ficheros en el pendrive hasta que no podamos más. Determinar si este algoritmo greedy encuentra solución óptima en todos los casos.

b) Queremos llenar el pendrive tanto como podamos, y para eso ordenamos los ficheros por orden decreciente de su tamaño, y vamos agregando ficheros en el pendrive hasta que no podamos más. Determinar si este algoritmo greedy encuentra solución óptima en todos los casos.

c) Dé los casos anteriores que satisfacen la forma greedy, programar una rutina que resuelva el problema.

5. El problema de la mochila 0-1 consiste en encontrar la manera óptima de cargar una mochila con objetos. Existen n objetos disponibles. El i-ésimo objeto cuesta v_i pesos, y pesa w_i kilos, donde v_i y w_i son enteros. Se pretende cargar el mayor valor en pesos posibles pero con un máximo W de kilos. Se debe decidir qué objetos tomar. Cada objeto puede ser elegido o no. No se puede tomar una parte de un objeto ni se puede elegir un objeto más de una vez.
Existe una variante, el problema de la mochila fraccional, de descripción similar, salvo que está permitido tomar fracciones de objetos.

a) Programar un algoritmo para resolver el problema de la mochila fraccional.

b) Explicar por qué este algoritmo no se puede extender al problema de la mochila 0-1.

6. El problema del cambio consiste en determinar, dado un número entero C, una combinación de monedas de 1 ctvo., 5 ctvos., 10 ctvos. y 20 ctvos. que sumen C y que use la mínima cantidad de monedas posible.

a) Programar un algoritmo para resolver este problema.

b) ¿Este algoritmo resuelve el problema para cualquier denominación de monedas? Justificar.

7. Escribir una versión divide and conquer de un algoritmo que dado un arreglo busque los dos elementos más grandes de un arreglo.

8. Escribir una función por backtracking que imprima todos los números de 6 cifras que no tengan repeticiones de dígitos.