Pregunta 3

1. Se espera que todas las listas sean de largo mayor > 0. En el caso de que la lista tenga 1 elemento solo tenemos una opción para elegir, si se tienen 2 elementos, como buscamos los elementos disjuntos, elegimos el mayor, si la lista tiene largo mayor a 2 seleccionamos la mayor suma entre la lista sin el ultimo elemento, la lista con el ultimo elemento y la lista que solo contiene al ultimo elemento:

$$F(n) = \begin{cases} 0 & \text{si } n = 0 \\ \max(a_1, 0) & \text{si } n = 1 \\ \max(F(n-1),\ F(n-2) + a_n) & \text{si } n > 2 \end{cases} \tag{1}$$

Demostración. Para probar la correctitud se procederá por el principio de inducción:

$\boxed{\text{Caso Base:}}$

$n = 0$

$$F(0) = 0$$

En el caso de que se entregue una lista vacía, la máxima satisfacción posible es 0, por lo que $F(0)$ debe devolver 0.

$n = 1$

$$F(1) = \max(a_1, 0)$$

Como la lista solo contiene un valor solo existen 2 sublistas posibles, la lista $[a_n]$ o la lista vacía. Si la lista $[a_n]$ es mayor o igual a 0 entonces se retorna su único valor, en caso contrario se entrega el vacío, en ambos casos se obtiene la sublista con la máxima satisfacción.

$\boxed{\text{Caso Inductivo:}}$

PDQ: $\forall n : F(0) \wedge F(1) \wedge \cdots \wedge F(n-1) \wedge F(n) \Rightarrow F(n+1)$

$\Rightarrow$ Debemos mostrar que el algoritmo calcula correctamente $F(n+1)$. Cuando el algoritmo calcula $F(n+1)$, este establece que:

$$F(n+1) = \max(F(n), F(n-1) + a_{n+1}\ , a_{n+1})$$

Si se tiene una lista de largo n $[a_1, ..., a_n]$ y le agregamos un elemento a_{n+1}, tenemos que F(n) y F(n-1) existen y nos devuelven los respectivos máximos, si queremos agregar a a_{n+1} a las opciones tendremos que sumárselo a F(n-1), de esta forma nos aseguramos que la sublista que se genera solo contiene elementos no adyacentes. En el caso de que agregar el elemento a_{n+1} y quedarnos con la máxima sublista en el conjunto $[a_1, ..., a_{n-1}]$ no nos entregue la máxima suma también se tiene la opción de que la máxima suma sea F(n). Al buscar el máximo entre F(n) y F(n-1)+a_{n+1} nos aseguramos que obtendremos la máxima satisfacción posible de todas las sublistas y que ninguna sublista contiene elementos adyacentes a otros en la lista original.

$\square$

2. Algoritmo que calcula F de forma recursiva:

```
def F(n, lista): #n = largo
    if n == 0:
        return 0
    else if n == 1:
        return max(a[0],0)
    else:
        return max(F(n - 1), F(n - 2) + a[n-1])
```

La complejidad de este algoritmo es $\Theta(2^n)$, como en cada llamada se vuelve a llamar otras 2 veces entonces por cada llamada duplica el tamaño del problema.

3. Algoritmo que calcula F de forma iterativa

```
def F(n, lista):
    Fn_1 = 0
    Fn_2 = 0
    for i in range(n):
        ( Fn_1, Fn_2 ) = ( max(Fn_1, Fn_2+lista[i]),  Fn_1 )

    return Fn_1
```

El algoritmos anterior tiene complejidad $\Theta(n)$, ya que su complejidad esta en el loop, que pasa por todos los elementos de la lista original.