1º)

```
sub_maximo(L, n):
    incluir_atual = 0
    excluir_atual = 0
    para i de 0 até n-1:
        novo_excluir = max(incluir_atual, excluir_atual)
        incluir_atual = excluir_atual + L[i]
        excluir_atual = novo_excluir
    retorne max(incluir_atual, excluir_atual)
```

O algoritmo itera uma vez por cada elemento da lista, ou seja, O(n).

2º)

```
sub_maximo_2xn(M, n):
    incluir_atual = [0,0]
    excluir_atual = [0,0]
    novo_excluir = [0,0]

    para i de 0 até n-1:
        novo_excluir[0] = max(incluir_atual[0], excluir_atual[0])
        novo_excluir[1] = max(incluir_atual[1], excluir_atual[1])

        incluir_atual[0] = excluir_atual[0] + M[0][i]
        incluir_atual[1] = excluir_atual[1] + M[1][i]

        excluir_atual[0] = novo_excluir[0]
        excluir_atual[1] = novo_excluir[1]

    retorne max(incluir_atual[0] + incluir_atual[1], excluir_atual[0] + excluir_atual[1])
```

Também itera uma vez por cada coluna. O(n).

3º)

```
sub_maximo_somazero(L):
    mapa = novo mapa // chave e valor
    max_tamanho = 0
    fim_subconjunto = -1

    para i de 0 até n-1:
        soma = soma + L[i]
        se soma == 0:
            max_tamanho = i+1
            fim_subconjunto = i
        se soma não está no mapa:
            mapa[soma] = i
        se soma - 0 está no mapa:
            tamanho_subconjunto = i - mapa[soma]
        se tamanho_subconjunto > max_tamanho:
            max_tamanho = tamanho_subconjunto
            fim_subconjunto = i
```

O algoritmo itera uma vez por cada elemento da lista,
e as operações no mapeamento é O(1), totalizando O(n).