Você é um detetive em uma pequena cidade e está investigando um roubo que aconteceu na região.

Durante a investigação, você descobre que uma carteira com 150 *fulampos* (o dinheiro da região) foram roubados. A vítima não lembra exatamente o que havia em sua carteira, mas sabe que existiam apenas duas notas.

Em pouquíssimo tempo, a polícia encontrou alguns suspeitos e acredita-se que o ladrão não tenha conseguido usar ou trocar o dinheiro roubado. Agora, você precisa descobrir quais deles poderiam ter roubado os 150 *fulampos* verificando a carteira de cada um deles.

Tarefa

Escreva uma função que recebe como parâmetro um array de **números inteiros ordenados** (referente às notas na carteira de um suspeito) e retorne, no menor tempo possível, se ele pode ter roubado os 150 *fulampos* da nossa vítima.

Lembre-se: A carteira do ladrão e das demais pessoas podem ter mais do que duas notas.

Casos de Teste

Teste sua função com TODOS estes casos antes de enviar:

Caso 1:
const carteira1 = [10, 20, 50, 70, 80, 100, 150]

Caso 2:
const carteira2 = [5, 25, 50, 100, 200]

Caso 3:
const carteira3 = [1, 2, 3, 147, 148, 149]

Caso 4:
const carteira4 = [75]

Caso 5:
const carteira5 = [150]

Caso 6:
const carteira6 = []

2. Marque os casos de teste em que sua solução retornou TRUE: *

- [x] Caso 1: [10, 20, 50, 70, 80, 100, 150]
- [x] Caso 2: [5, 25, 50, 100, 200]
- [x] Caso 3: [1, 2, 3, 147, 148, 149]
- [ ] Caso 4: [75]
- [ ] Caso 5: [150]
- [ ] Caso 6: [] (array vazio)
- [ ] Nenhum dos casos retornou true

3. Qual é a complexidade de tempo da solução que foi desenvolvida?

A notação Big-O é uma forma de descrever como o tempo de execução de uma solução cresce conforme aumenta a quantidade de dados.
• O(1) → tempo constante: não importa se são 10 ou 1 milhão de itens, o tempo é praticamente o mesmo.
• O(log n) → cresce devagar: mesmo com muitos dados, o tempo aumenta pouco (ex.: procurar em uma lista ordenada).
• O(n) → cresce proporcionalmente: dobrar a quantidade de dados dobra o tempo de execução.
• O(n log n) → cresce um pouco mais rápido que proporcional, mas ainda é eficiente (ex.: bons algoritmos de ordenação).
• O(n^2) → cresce muito rápido: se dobrar os dados, o tempo pode quadruplicar (ex.: comparar cada item com todos os outros).

O(n)

4. Explique sua solução:

- Como seu algoritmo funciona?
- Por que você escolheu essa abordagem?
- Por que a complexidade que você selecionou acima está correta? *