ED2 - Mais um Problema.pdf - Rafael Manfrim

Você pediu uma pizza de mussarela de n fatias para você e seus colegas. Entretanto, vocês possuem gostos diferentes. Alguns gostam de cebola e outros de calabresa. Deste modo, decidiram pedir uma pizza metade cebola e metade calabresa. Infelizmente, a pizza chegou com as coberturas misturadas. Não era possível identificar as metades, sendo que toda a pizza ficou com rodelas de calabresa e tiras de cebola. As tiras grudaram na mussarela e não é possível removê-las. A pizza já veio cortada em 16 pedaços. Você gosta bem mais de calabresa do que de cebolas, portanto, deseja pegar os pedaços que tem mais calabresa. Entretanto, seus colegas não gostaram de suas exigências por pedaços específicos, e definiram que se você escolher um pedaço, terá que pegar os outros logo em sequência (para à direita ou à esquerda). Seu objetivo é selecionar k pedaços, k > 0, onde você obtenha a maior diferença possível entre rodelas de calabresa e tiras de cebola. Você deseja maximizar o total das diferenças entre calabresas e cebolas consumidas, independente do número de fatias que comer. Ou seja, cada calabresa no pedaço conta positivamente para o "valor" do pedaço, e cada cebola conta negativamente. Por exemplo, você pode escolher três pedaços em sequência, onde:

- Pedaço 1: 3 calabresas e 1 cebola.
- Pedaço 2: 1 calabresa e 2 cebolas.
- Pedaço 3: 4 calabresas e 2 cebolas.

Obtendo uma diferença final de 3. Seu objetivo é maximizar essa diferença, independente do número de fatias consumidas, podendo ser apenas uma fatia, ou a pizza inteira. O número de pedaços consumidos não é relevante, portanto, pode tratar situações de empate como preferir. Criem um algoritmo que resolva este problema.

```cpp
#include <iostream>

using namespace std;

int main() {
    // Aqui já estão os resultados da relação calabresa x cebola
    int pizza[] = {-3, 4, -5, 0, 2, -1, 6, -4, 3, 8, -10, 2, -3, 1, 6, 9};
    int tamanho_pizza = 16;
    int rodada_pizza = 1;
    int pedaco = 0;

    int inicio = 0;
    int fim = 0;

    int pontuacao = 0;
    int negativos_acumulados = 0;
    int positivos_acumulados = 0;
    int positivos_aplicar = 0;

    while(true) {
        if(rodada_pizza == 2 && pedaco == inicio) {
            pontuacao += positivos_aplicar;
            break;
        }
        if(pizza[pedaco] < 0) {
            if(pontuacao >= 0) {
                negativos_acumulados += pizza[pedaco];
            }
            if(pontuacao + negativos_acumulados < 0) {
                inicio = 0;
                pontuacao = 0;
```

```
                negativos_acumulados = 0;
                positivos_acumulados = 0;
            }       } else {
            if(inicio == 0) {
                inicio = pedaco;
                positivos_acumulados = 0;
                positivos_aplicar = 0;
            }
            if(pizza[pedaco] + negativos_acumulados >= 0) {
                positivos_aplicar += positivos_acumulados;
                pontuacao += pizza[pedaco] + negativos_acumulados;
                negativos_acumulados = 0;
                positivos_acumulados = 0;
                fim = pedaco;
            } else {
                positivos_acumulados += pizza[pedaco];
            }
        }
        pedaco++;

        if(pedaco == tamanho_pizza) {
            if(rodada_pizza == 2) {
                break;
            }
            rodada_pizza++;
            pedaco = 0;
        }
    }
    cout << "Pizza tem " << tamanho_pizza << " pedaços" << endl;
    cout << "Início: " << inicio + 1 << endl;
    cout << "Fim: " << fim + 1 << endl;
    cout << "Pontuacao: " << pontuacao;

    return 0;
}
```

A solução deve ser ótima, ou seja, a melhor solução possível. Determine a sua complexidade de tempo no pior caso utilizando a notação-O.

```
  Minha solução possui um loop que roda 2 vezes pelo número de pedaços da pizza, ou seja, O(n) pois a constante multiplicativa 2 é descartada.
```