Задача 9

Всесторонний анализ алгоритма

MAX_SUM.cpp

```cpp
#include <iostream>
#include <vector>
#include <climits>

int long_find_max_sum(const std::vector<int>& arr,
                      int k) {
    int n = arr.size();
    int max_sum = INT_MIN;

    for (int i = 0; i <= n - k; ++i) {
        int current_sum = 0;
        for (int j = i; j < i + k; ++j) {
            current_sum += arr[j];
        }

        max_sum = std::max(max_sum, current_sum);
    }

    return max_sum;
}
```

1. Оцените и обоснуйте асимптотическую верхнюю границу сложности для этой функции.
2. Как можно **оптимизировать** этот алгоритм? Разработайте оптимизированный алгоритм и обоснуйте его сложность.

1) Оценим сложность:

```
for i = 0 ... n-k :                                   | n-k+2
    current_sum = 0                                   | n-k+1
    for j = i ... i+k-1 :                             | (n-k+1)(k+1)      T(n,k) = O(nk)
        current_sum += arr[j]                         | (n-k+1) k
    max_sum = max(max_sum, current_sum)               | n-k+1
```

$\sum_{i=0}^{n-k} (i+k-i+1) = \sum_{i=0}^{n-k} (2i+1)$

2) Можно реализовать следующий алгоритм:

```cpp
int long_find_max_sum(const std::vector<int>& arr, int k) {
    int n = arr.size();
    int current_sum = 0;

    // Считаем сумму первых k элементов
    for (int i = 0; i < k; i++) {
        current_sum += arr[i];
    }

    int max_sum = current_sum;

    // Двигаем окно
    for (int i = k; i < n; i++) {
        current_sum += arr[i] - arr[i - k];
        max_sum = std::max(max_sum, current_sum);
    }

    return max_sum;
}
```

1. Посчитаем сумму первых k эл-ов ($T_1(n,k) = O(k)$)
2. Будем пересчитывать новую сумму в окне с концом в k за $O(1)$ по формуле, для этого достаточно из предыдущей суммы в окне отнять первый эл-т и добавить следующий, т.к. все остальные эл-ты в окне сохраняются при сдвиге окна на 1.

В итоге алгоритм будет работать за $O(n)$, т.к. 1. цикл работает за $O(k)$, а 2. цикл работает за $O(n)$, при этом $k \le n$, значит $T(n,k) = O(n)$.