# dz3

## 5.1

# Упражнение 5.1 **Сравнение алгоритмов**

1. Время работы алгоритма **A** (в микросекундах) зависит от размера входных данных следующим образом:
   $\boldsymbol{T_A(n) = 0,1n^2 \log_{10} n}$.
2. Время работы алгоритма **B** (в микросекундах) зависит от размера входных данных следующим образом:
   $\boldsymbol{T_B(n) = 2,5n^2}$.

- Какой из этих алгоритмов более эффективен с точки зрения $O$? Начиная с какого размера входных данных?
- Если практический размер входных данных не превышает $10^9$, какому алгоритму следует отдать предпочтение?

второй алгоритм эффективнее

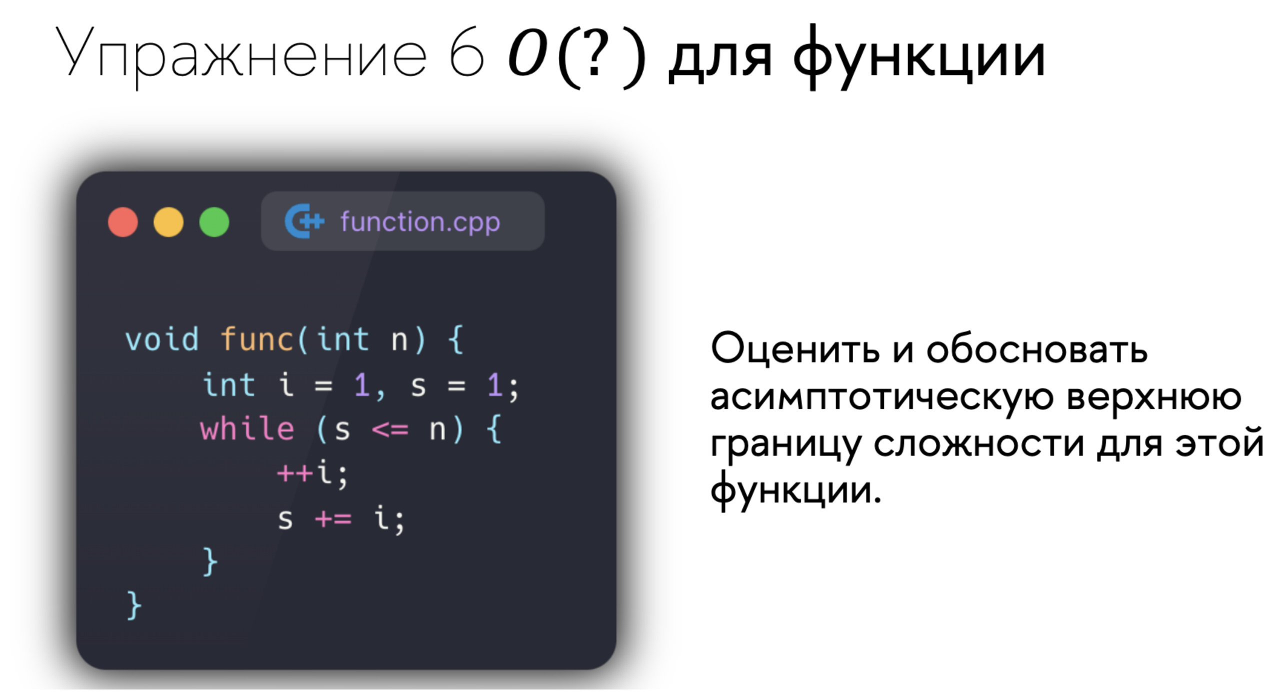
$0.1n^2 log_{10} n \geq 2.5n^2$

$log_{10} n \geq 25$

выполняется начиная с $n = 10^{25}$

если размер входных данных меньше $10^9$ лучше пользоваться 1м алгоритмом

## 6




мы будем складывать числа пока сумма $1+..+k$ не станет больше чем $n$

тогда получится, что

$$\frac{k(k+1)}{2} \geq n$$

$$k^2 + k \geq 2n$$

$$k^2 + k - 2n \geq 0$$

$$k \geq \frac{1+\sqrt{1+4n}}{2}$$

именно $k$ раз будет выполнятся цикл

тогда верхняя граница сложности:

$$O(N) = \sqrt{N}$$

## Последний слайд

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. Оценить и обосновать асимптотическую верхнюю границу сложности для этой функции.

Этот алгоритм ищет максимальную сумму чисел на подотрезке длины $k$

алгоритм перебирает элементы начала отрезка, и для каждого начала считает сумму этого отрезка

$$T(N, K) = 2 + C(N - K)K$$

поэтому верхняя граница сложности для этого алгоритма:

$$O(N, K) = NK$$

действительно, $2 + C(N - K)K < NK$

например при $N > K > 10$

## 2. Как можно улучшить/оптимизировать этот алгоритм? Разработайте оптимизированный алгоритмов и обоснуйте его сложность.

можно воспользоваться методом скользящего окна, то есть не пересчитывать сумму для каждого отрезка, а считать используя уже подсчитанные данные.

```cpp
#include <vector>

int fast_find_max_sum(const std::vector<int>& arr, int k) {
    int n = arr.size();
    int max_sum = INT_MIN;
    int cur_sum = 0;

    for(size_t i = 0; i < k; ++i){
        cur_sum+=arr[i];
    }

    max_sum = std::max(max_sum,cur_sum);

    for(size_t i = k; i< n; ++i){
        cur_sum-=arr[i-k];
        cur_sum+=arr[i];
        max_sum = std::max(max_sum,cur_sum);
    }

    return max_sum;
}
```

в этом алгоритме я избавился от вложенного цикла, тем самым привел его к линейному виду

$$T(N) = 4 + K + N - K = N + 4$$

тогда верхняя граница

$$O(N) = N$$