1. Функция fibonachi(int num)

Описание:

Эта функция вычисляет n-е число Фибоначчи рекурсивно. Числа Фибоначчи определяются следующим образом:

• F(0) = 0

• F(1) = 1

• F(n) = F(n-1) + F(n-2) для n > 1

Алгоритмическая сложность:

• O(2^n): Рекурсивный подход создает два вызова для каждого n, что приводит к экспоненциальному количеству вызовов.

Оптимизация:

• Можно использовать мемоизацию (хранение уже вычисленных значений) или итеративный подход, чтобы снизить сложность до O(n). Также можно использовать формулу Бине или матрицы, чтобы достичь O(log n).

▎2. Функция akkerman(int m, int n)

Описание:

Функция вычисляет значение функции Аккермана, которая является одной из классических примеров рекурсивных функций. Она определяется следующим образом:

• A(0, n) = n + 1

• A(m, 0) = A(m - 1, 1) для m > 0

• A(m, n) = A(m - 1, A(m, n - 1)) для m > 0 и n > 0

Алгоритмическая сложность:

• О(2^(m+n)): Эта функция растет очень быстро и не имеет простого выражения для сложности. На практике, для небольших значений m и n она может быть вычислена, но для больших значений это становится непрактично.

Оптимизация:

• Для небольших значений m и n функции Аккермана можно использовать табличное хранение результатов, но это может потребовать значительных ресурсов памяти. В общем случае оптимизация затруднительна из-за экспоненциального роста.

▎3. Функция search(const vector<vector<int>>& array, int target, int row, int col)

Описание:

Эта функция ищет заданное число (target) в двумерном массиве (векторе векторов). Она проходит по всем элементам массива рекурсивно.

Алгоритмическая сложность:

• O(m \* n): где m — количество строк, а n — количество столбцов в массиве. В худшем случае функция проверяет каждый элемент массива.

Оптимизация:

• Если массив отсортирован (например, по строкам и столбцам), можно использовать алгоритм бинарного поиска или другие методы, чтобы сократить время поиска до O(log(m \* n)). Если массив не отсортирован, можно рассмотреть использование других структур данных (например, хэш-таблиц), чтобы достичь O(1) в среднем случае для поиска.  
