Содержание

Введение

Рекурсия – это метод программирования, при котором функция вызывает саму себя. Основная идея рекурсии – разбиение залачи на минимальную подзадачу. Каждый рекурсивный вызов должен приближаться к условию завершения во избежание бесконечного цикла.

Плюсы: простота реализации, читаемость кода в задачах с естественной рекурсивной структурой.

Минусы: риск переполнения стека при глубоком погружении, дополнительные накладные траты ресурсов при вызове функции.

Цель работы – провести сравнительный анализ нерекурсивного и рекурсивного алгоритмов для нахаждения второго максимального по величине элемента в последовательности чисел.

Задачи:

1) разработка рекурсивного и нерекурсивного алгоритмов;
2) реализовать алгоритмы;
3) описать тесты для реализации алгоритмов;
4) выполнить тестирование;
5) оценка затрат трудоёмкости и памяти;
6) провести сравнительный анализ реализаций.

1. Описание алгоритмов

1.1 Нерекурсивный алгоритм

В процессе работы последовательно обрабатываются числа, считываемые с клавиатуры, до встречи с нулём. На каждом шаге поддерживается два значения: max1 (наибольший элемент) и max2 (второй по величине элемент). При обработке нового числа n выполняются следующие проверки:

1) если n > max1, то max2 обновляется значением max1, а max1 – а значением n;
2) если n == max1, то max2 устанавливается равным max1 (для учёта повторяющихся максимальных значений);
3) если n < max1 и n > max2, то max2 обновляется значением n.

Листинг 1. Реализация нерекурсивного алгоритма

```
void secondMaxIterative(int &max1, int &max2) {
    int n;
    cin >> n;
    while (n != 0) {
        if (n > max1) {
            max2 = max1;
            max1 = n;
        } else if (n == max1) {
            max2 = max1;
        } else if (n < max1 && n > max2) {
            max2 = n;
        }
        cin >> n;
    }
}
```

Линейная временная сложность O(n) (фиксированный набор операций сравнения и присваивания, их количество итераций линейно зависит от длины входной последовательности до первого нуля).

Постоянный объём используемой памяти O(1) (только три переменные).

1.2 Рекурсивный алгоритм

В процессе работы используется рекурсия. Функция secondMax вызывает саму себя, передавая по ссылке значения max1 и max2. На каждом шаге рекурсии:

1) считывается число n;
2) если n == 0, то выход из рекурсии;
3) в другом случае проверки числа n по тем же правилам.

Листинг 2. Реализация рекурсивного алгоритма

```
void secondMaxRecursive(int &max1, int &max2) {
    int n;
    cin >> n;
    if (n == 0) {
        return;
    }

    if (n > max1) {
        max2 = max1;
        max1 = n;
    } else if (n == max1) {
        max2 = max1;
    } else if (n > max2) {
        max2 = n;
    }

    secondMaxRecursive(max1, max2);
}
```

Линейная временная сложность O(n).

Линейный объём используемой памяти O(n) (из-за стека рекурсии).

2. Средства реализации и архитектура

Технические характеристики устройства, на котором выполнялись замеры:

1. Операционная система Windows 11.
2. Оперативная память 32 ГБ.
3. Процессор Intel(R) Core(TM) i7-7600U @ 2.80 ГГц , архитектура x64.
4. Компилятор MinGW.
5. Среда разработки Clion.
6. Язык программирования: C++.

Условия тестирования:

1. Ноутбук подключен к сети электропитания;
2. Система нагружена только средой разработки CLion и тестируемой программой;

3. Тестирование программы

Корректность реализаций алгоритмов проводятся на тестовых сценариях:

1) второй максимум равен первому – корректность определения второго максимума, если он совпадает с первым;

2) убывающая последовательность – обычный случай, первый и второй максимумы изначально передаются в функцию;
3) все элементы равны – последовательность состоит их повторяющихся элементов, исключительный случай для проверки работы реализации алгоритма;
4) первый максимум в main, второй в функции – проверка на то, что передающиеся значения в функцию сохраняются;
5) возрастающая последовательность – проверка, что функция правильно найдет первый и второй максимумы.

В таблице 1 обобщены все ранее описанные выкладки.

Таблица 1. Описание проведенных тестов

№	Входные данные	Ожидаемый результат	Критерий
1	3 5 2 5 0	5	Второй максимум равен первому
2	5 4 3 2 1 0	4	Убывающая последовательность
3	2 2 2 0	2	Все элементы равны
4	10 1 10 5 0	10	Первый максимум в main, второй в функции
5	1 2 3 4 5 6 0	5	Возрастающая последовательность

4. Оценка трудоёмкости

Определим зависимость времени выполнения от размера входных данных N, а также выявим предельное значений, при котором применение рекурсивного подхода становится неэффективным или невозможным. Измерения производились с использованием библиотеки chrono (C++17), которая позволяет оценить время выполнения с микросекундной точностью. Каждая итерация запускалась на тех же данных. Для каждого значения N измерялось время выполнения обоих алгоритмов в микросекундах с помощью std::chrono::high_resolution_clock [2]. Изначально вектор значений N был от 3 до 100000. Функция расчета реализована так, что мы имеем записи выполнения функции до тех пор, пока программа не «упадет» с ошибкой 0xC00000FD (Stack Overflow). На рисунке 1 наглядно показано, как это происходит.

Рис.1. Работа программы до переполнение стека

Из рисунка видно, что до размера последовательности в 10000 программа работает, но при 20000 уже происходит ошибка, и операционная система аварийно завершает программу. В таблице 2 представлены имеющиеся данные.

Таблица 2. Время работы реализаций

N	рекурсивно, мс	последовательно, мс
3	26	2
30	23	17
100	25	26
200	90	84
500	219	130
1000	361	306
2000	395	390
5000	1614	1397
10000	2140	1877

Исходя из данных таблицы можно сделать выводы:

1) при N <= 200 обе реализации работают без существенных различий;
2) с увеличением N, итеративный алгоритм начинает выигрывать по времени, особенно начиная с N > 1000.

На основе полученных экспериментальных данных построен график зависимости времени выполнения обеих реализаций алгоритма от размера входных данных N (рисунок 2).

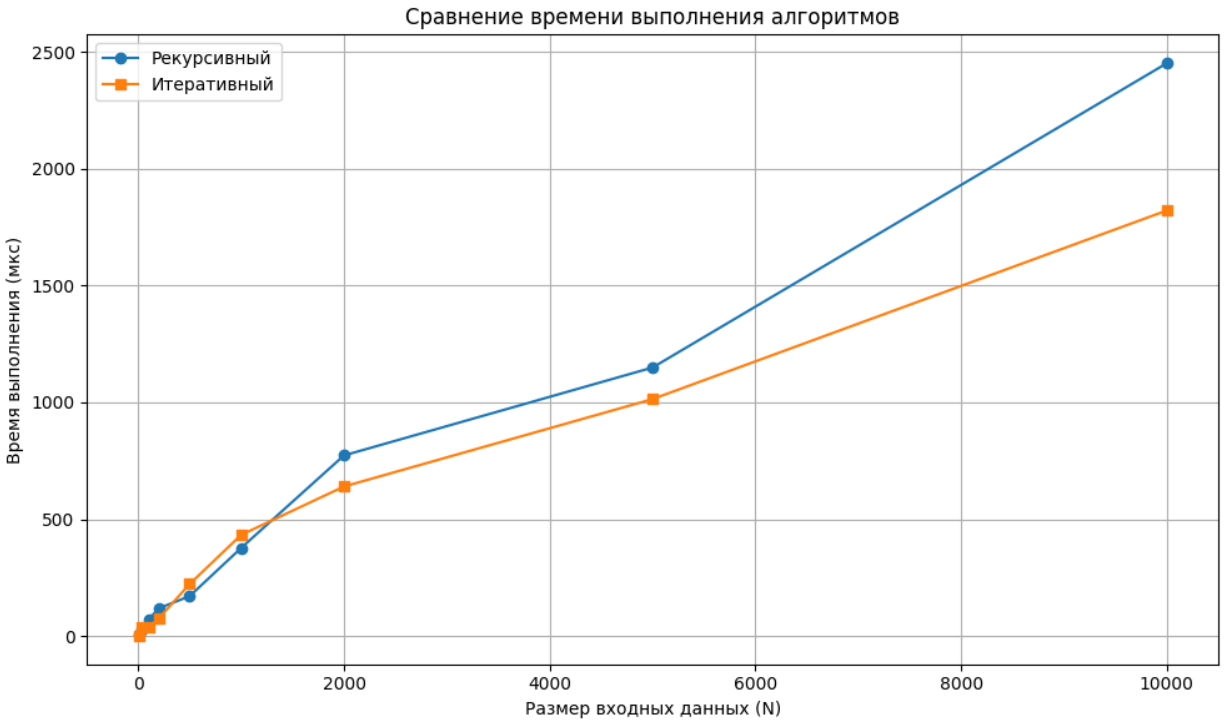

Рис. 2. Зависимость времени выполнения алгоритмов от размера входных данных

Заключение

В ходе выполнения работы были разработаны и реализованы два алгоритма для нахождения второго по величине максимального элемента в последовательности: нерекурсивный (с использованием цикла) и рекурсивный. Оба алгоритма показали корректную работу на всех тестовых сценариях, подтверждая их устойчивость к различным типам входных данных.

Оба рассмотренных алгоритма – обладают одинаковой временной сложностью O(n), поскольку каждый из них обрабатывает входную последовательность чисел построчно, выполняя константное число операций для каждого элемента. Однако на практике рекурсивный подход может быть менее эффективным при больших объёмах данных из-за накладных расходов на вызовы функций и управление стеком вызовов. Это делает его менее предпочтительным в условиях, где критична производительность.

С точки зрения использования памяти, нерекурсивная реализация выигрывает: она требует O(1) памяти, так как работает с фиксированным числом переменных. Рекурсивная же реализация использует O(n) памяти, что связано с формированием стека вызовов — каждый новый шаг рекурсии создаёт новый стек-фрейм. Это ограничивает её применимость: при длинных последовательностях возможен риск переполнения стека.

Таким образом, рекурсивный алгоритм может быть предпочтителен при работе с небольшими входными данными, когда приоритетом являются простота и читаемость кода. Нерекурсивный же подход рекомендуется

использовать в задачах, связанных с обработкой больших объёмов данных, где особенно важны надёжность, эффективность и экономное использование ресурсов.

Список использованных источников

1. Анисимов А.Е. Практикум по программированию на C++. - Ижевск: Удмуртский университет, 2022. - 199 с.
2. QueryPerformanceCounter function (Windows) // документация Microsoft по Win32 API [Электронный ресурс] URL: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter (дата обращения: 11.04.2025)