ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Название лабораторной работы (Задача о разделении множества)»

Выполнила работу

Смирнова Анна

Академическая группа №J3112

Принято

Преподаватель: Дунаев Максим Владимирович

Санкт-Петербург

2024

1. Введение

Задача о разделении множества.

Цель: Дан массив, необходимо разделить его на две части с равной суммой.

Задачи:

* Задача должна быть решена комбинаторным способом.
* Сложность алгоритма должна быть не менее чем О(2^N).
* В массиве содержатся только положительные числа.
* Максимум 25 элементов и сумма до 200.

1. Теоретическая подготовка

* Знание языка программирования C++
* Понимание алгоритма для решения задачи

1. Реализация

Процесс этапа выполнения работы:

1. Постановка задачи:

Необходимо выяснить, как разделить заданный массив на две части с равной суммой.

1. Анализ поставленной задачи:

Для решения можно использовать рекурсию для перебора всех возможных комбинаций элементов массива.

1. Проектирование алгоритма:

Использование библиотек:

* #include <iostream>
* #include <vector>

Основные шаги алгоритма:

* Подсчет общей суммы элементов массива.
* int totalSum = 0;
* for (int num : arr) {
* totalSum += num;
* }
* Проверка, является ли общая сумма четным числом (если нечетное, разделение невозможно).

if (totalSum % 2 != 0) {

        cout << "Невозможно разделить массив на две части с равной суммой." << endl;

        return 0;

    }

* Реализация рекурсивной функции для нахождения двух подмножеств с равными суммами с выводом их в терминал.
* bool split(const vector<int>& arr, int n, int sum1, int sum2, vector<int>& subset1, vector<int>& subset2) {
* if (n == 0) {
* return sum1 == sum2;
* }
* subset1.push\_back(arr[n - 1]);
* if (split(arr, n - 1, sum1 + arr[n - 1], sum2, subset1, subset2)) {
* return true;
* }
* subset1.pop\_back();
* subset2.push\_back(arr[n - 1]);
* if (split(arr, n - 1, sum1, sum2 + arr[n - 1], subset1, subset2)) {
* return true;
* }
* subset2.pop\_back();
* return false;
* }

Тестирование:

* Проверка кода на массивах с чётным и нечётным количеством элементов, а также с различными значениями.

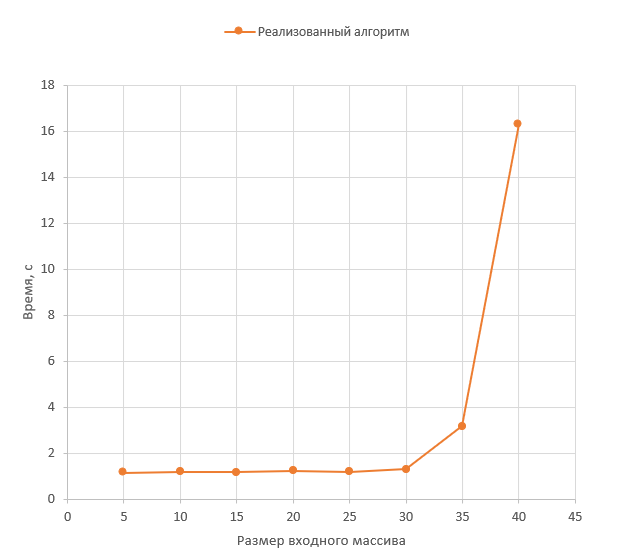
4. Экспериментальная часть

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически заданная сложность задачи составляет O(2^N) и более. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 |
| Время выполнения программы, с | 1,152 | 1,207 | 1,173 | 1,253 | 1,214 | 1,324 | 3,185 | 16,319 |

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №1.



Изображение №1 - График работы алгоритма

Подсчёт по памяти:

Для массива arr: sizeof(int) \* N

Для разделённых массивов: 2 \* sizeof(int) \* N

Так как int = 4 байт, то для переменных: 4 + 4 + 4 + 8 + 8 = 28 байт

Для разделённых массивов: 2 \* sizeof(int) \* N

Для рекурсивных вызовов: 28 байт \* N.

Следовательно, общая память равна: 4\*N + 8\*N + 28\*N = 40\*N байт

Подсчёт асимптотики (сложности):

Для цикла подсчёта суммы: O(N)

Для рекурсивной функции: O(2^N)

Общая сложность: O(2^N)

Анализ графика и таблицы:

По полученным значениям в результате изменения размера входного набора видно, что время выполнения работы алгоритма для массивов, содержащих до 25 элементов, значительно не изменяется. Поэтому, я увеличила количество, сохранив при этом сумму до 200. Изменение времени работы для 35 и 40 элементов заметно на графике: для 30 – почти не поменялось, для 35 – увеличилось в почти 2,5 раза, а для 40 – увеличилось ещё в 5 раз.

1. Заключение

В ходе выполнения работы мною был реализован алгоритм, позволяющий определить, можно ли разделить массив целых чисел на два подмножества с равной суммой. Цель работы была достигнута путём тестирования кода на массивах с чётным и нечётным количеством элементов, а также с различными значениями.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат использования памяти.

1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла lab4.cpp

#include <iostream>

#include <vector>

using namespace std;

bool split(const vector<int>& arr, int n, int sum1, int sum2, vector<int>& subset1, vector<int>& subset2) {

    if (n == 0) {

        return sum1 == sum2;

    }

    subset1.push\_back(arr[n - 1]);

    if (split(arr, n - 1, sum1 + arr[n - 1], sum2, subset1, subset2)) {

        return true;

    }

    subset1.pop\_back();

    subset2.push\_back(arr[n - 1]);

    if (split(arr, n - 1, sum1, sum2 + arr[n - 1], subset1, subset2)) {

        return true;

    }

    subset2.pop\_back();

    return false;

}

int main() {

    vector<int> arr = {1, 1, 1, 1, 2};

    vector<int> subset1, subset2;

    const int N = size(arr);

    int totalSum = 0;

    for (int num : arr) {

        totalSum += num;

    }

    if (totalSum % 2 != 0) {

        cout << "Невозможно разделить массив на две части с равной суммой." << endl;

        return 0;

    }

    if (split(arr, N, 0, 0, subset1, subset2)) {

        cout << "Первый массив: ";

        for (int num : subset1) {

            cout << num << " ";

        }

        cout << endl;

        cout << "Второй массив: ";

        for (int num : subset2) {

            cout << num << " ";

        }

        cout << endl;

    }

    else {

        cout << "Невозможно разделить массив на две части с равной суммой." << endl;

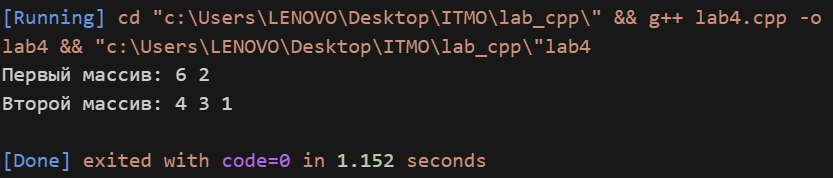
    }

    return 0;

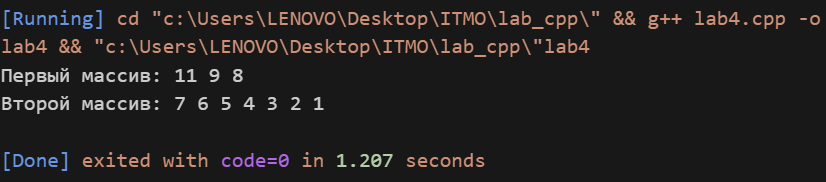
}

ПРИЛОЖЕНИЕ В

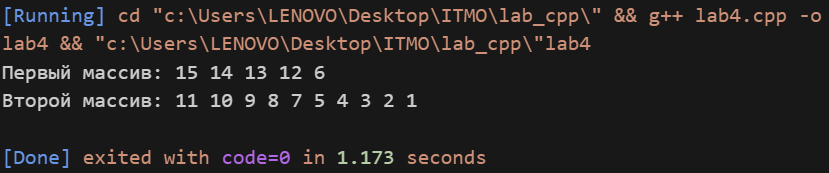
Результаты работы кода для массивов с различным количеством элементов и суммой до 200 предоставлены на изображениях №2-9.



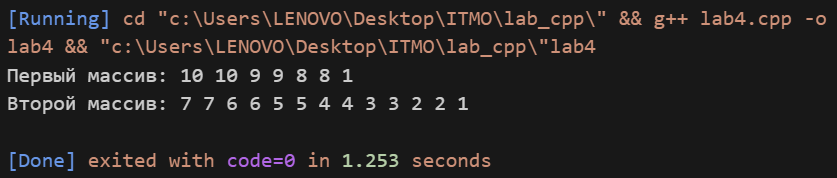
Изображение №2 - Результат работы алгоритма для 5 элементов



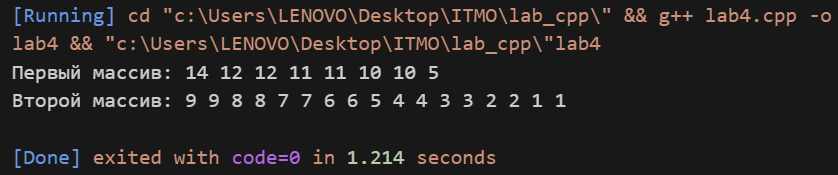
Изображение №3 - Результат работы алгоритма для 10 элементов



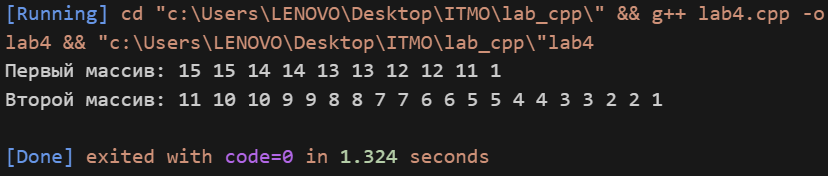
Изображение №4 - Результат работы алгоритма для 15 элементов



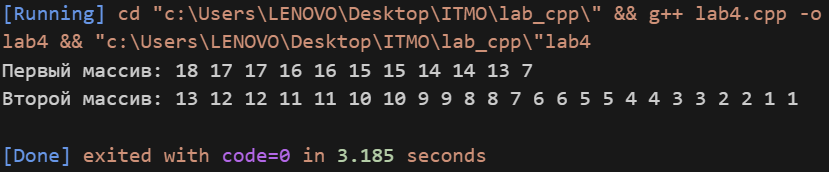
Изображение №5 - Результат работы алгоритма для 20 элементов



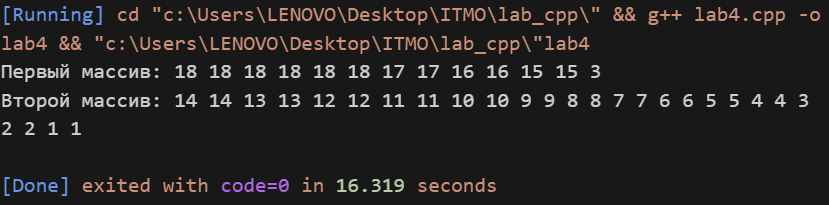
Изображение №6 - Результат работы алгоритма для 25 элементов



Изображение №7 - Результат работы алгоритма для 30 элементов

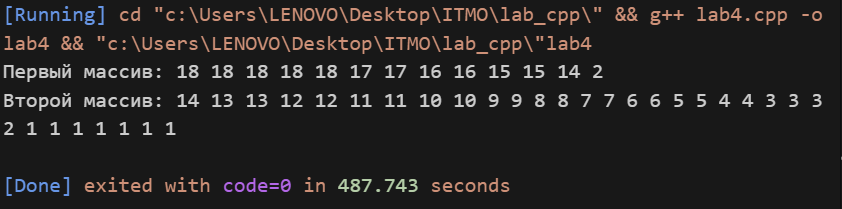


Изображение №8 - Результат работы алгоритма для 35 элементов



Изображение №9 - Результат работы алгоритма для 40 элементов

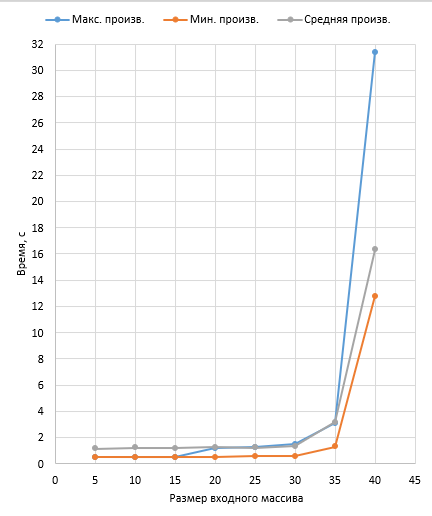
В рамках интереса я исследовала время работы для 45 элементов, результат показан на Изображении №10.



Изображение №10 - Результат работы алгоритма для 45 элементов

ПРИЛОЖЕНИЕ С

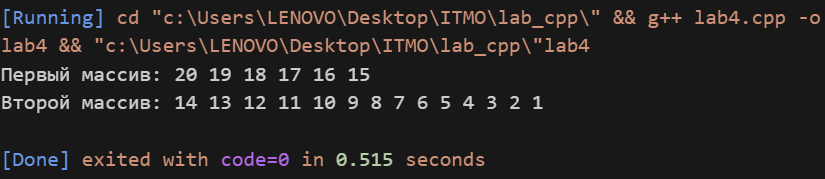
В ходе выполнения лабораторной работы мною было замечено, что при высокой производительности компьютера, эффективность работы алгоритмов значительно увеличивается. Особенно это заметно при увеличении количества элементов. График представлен на изображении №11.

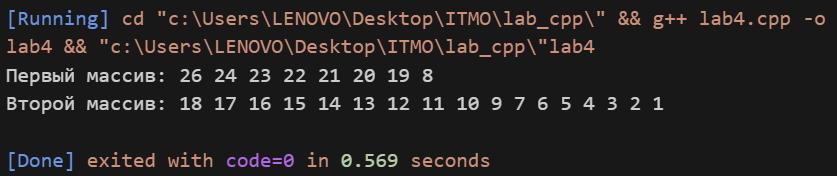


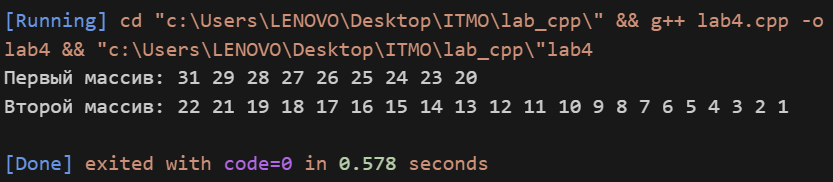
Изображение №11 – График с различными производительностями

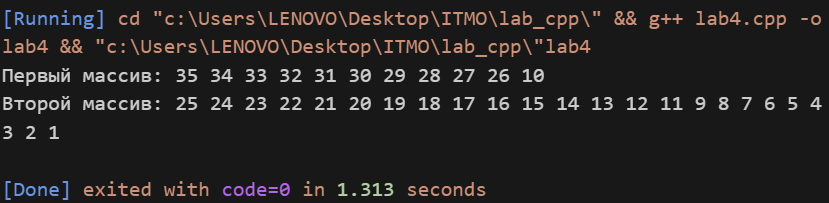
ПРИЛОЖЕНИЕ D

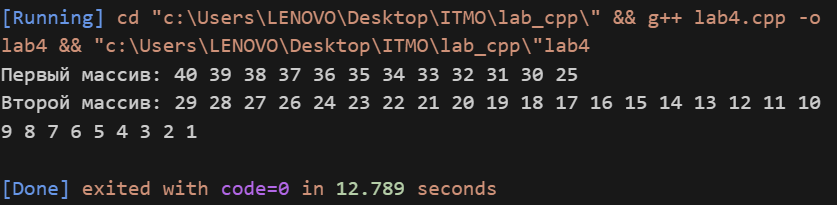
Максимальная производительность, питание от сети.

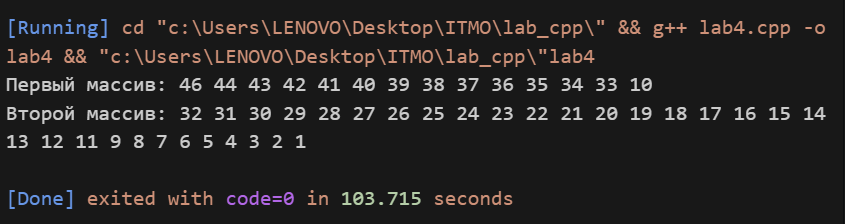












Минимальная производительность, нет питания от сети.

