ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Реализовать 3 алгоритма сортировки разной сложности»

Выполнил работу

Кащеев Максим Николаевич

Академическая группа

J3113

Принято

Иван Владимирович Ходненко

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Целью работы является анализ алгоритмов сортировки, чтобы узнать, как, в зависимости от различной сложности и различной space complexity меняется время работы алгоритма.

1. Теоретическая подготовка

Для выполнения данной лабораторной нужно уметь писать тесты, знать типы данных int, vector, bool. Также необходимо знать библиотеки, такие как iostream, fstream, vector, chrono, string, random, знать основные алгоритмы сортировки, уметь работать с вводом и выводом.

1. Реализация

По ходу выполнения данной лабораторной работы я реализовывал каждую сортировку в отдельном файле (Cocktail sort – являющийся модифицированной версией сортировки bubble, но с тем отличием, что массив сортируется с конца и с начала одновременно. Heap sort – разложение массива на бинарное дерево с целью дальнейшей сортировки. Bucket sort – разложение неотсортированного массива на отдельные подмассивы, по заданной метрике и дальнейшей сортировке каждого из подмассивов с целью последовательной конкатенации получившихся массивов в один отсортированный массив.) Далее я реализовывал тесты для каждого из алгоритмов сортировки для проверки их сложности в худшем/среднем/лучшем случаях. В файле main реализовал вызов тестов для проверки сложности алгоритма, а также тесты, для построения графиков зависимости времени от количества элементов и создания box plot для каждого из алгоритмов.

1. Экспериментальная часть

В этом разделе вам необходимо привести результаты работы вашего алгоритма, с таблицами и графиками, демонстрирующими выполнения алгоритма с различными условиями и наборами данных. Оценивается производительность и сравниваются результаты с теоретическими оценками.

Подсчёт по памяти (только для циклов и сложных структур) – как в лабораторной работе №2.

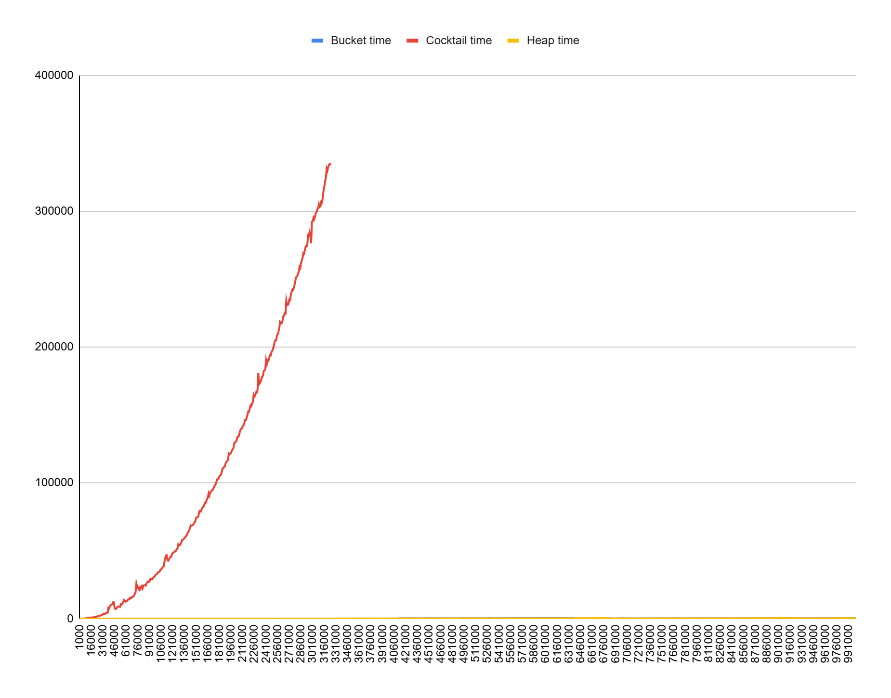
Теоретическое использование памяти алгоритмом cocktail sort составляет O(1) -–так как работа ведется с исходным переданным массивом в памяти. Теоретическое использование памяти для алгоритма heap sort составляет O(1) так как алгоритм сортирует исходный массив, не выделяя память. Теоретическое использование памяти для алгоритма bucket sort составляет O(N + B) где B – количество корзин, при этом при приближении количества корзин к N это значение приближается к O(N).

Подсчёт асимптотики (только для циклов и сложных структур) – как в лабораторной работе №3.

График зависимости времени от числа элементов. Пример выполнения:

Согласно требованиям моего варианта, на вход к моему алгоритмам подаётся до 100000 элементов с шагом в тысячу. Теоретически заданная сложность для алгоритма cocktail sort составляет O(N^2). Теоретически заданная сложность для алгоритма heap sort составляет O(Nlog(N)). Теоретически заданная сложность для алгоритма bucket sort составляет O(Nlog(k)). Для тестирования алгоритма была собрана статистика, приведенная в графике №1.

График, представляющий зависимость времени от количества элементов для каждой из сортировок представлен на изображении №1.



Изображение №1 - График работы алгоритма

Более удобное логарифмическое представление графика алгоритмов, представленного на изображении №1

Изображение выглядит как текст, линия, диаграмма, снимок экрана

Автоматически созданное описание

Изображение №2 – Логарифмический график работы алгоритма

Box plot график для алгоритмов heap sort, bucket sort и cocktail sort.

Изображение выглядит как снимок экрана, текст, диаграмма, линия

Автоматически созданное описание

Изображение №3 – box plot для алгоритмов сортировки

1. Заключение

В ходе выполнения лабораторной работы мною были реализованы три вида алгоритмов сортировки. Цель работы была достигнута путём реализации алгоритмов, соответствующих тестов для них и анализа данных на графике, получившимся в результате выполнения тестов. Полученные результаты демонстрируют, насколько сильно влияет сложность алгоритма на время его реализации.

Фактическое время исполнения хорошо соотносится с теоретической сложностью алгоритмов, значительных выбросов нет. Однако поскольку тестовый набор данных случаен, и количество прогонов составляет 1000, то выбросы есть, поскольку в случайных данных находятся как данные приближенные к лучшему, так и к худшему случаям. Также для быстрых алгоритмов, есть более продолжительные по времени, но меньшие по абсолютным значениям выбросы, связанные с параллельной загрузкой процессора другими задачами.

В качестве дальнейших исследований можно более оптимально реализовать алгоритмы сортировки и уменьшить затраты памяти (к примеру, применить другую сортировку для подмассивов для алгоритма bucket).

1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла cocktail.cpp

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include "tests.cpp"

void coctail(std::vector<int> &matrix)

{

    int end = size(matrix);

    int start = 0;

    bool swapped\_elements = true;

    int middle = 0;

    while (swapped\_elements)

    {

        swapped\_elements = false;

        for (int i=start; i<end; i++) {

            if (matrix[i] > matrix[i+1] && i+1<size(matrix))

            {

                middle = matrix[i];

                matrix[i] = matrix[i+1];

                matrix[i+1] = middle;

                swapped\_elements = true;

            }

        }

        if (swapped\_elements == false)

        {

            break;

        }

        end = end - 1;

        for (int i=end; start<i; i--)

        {

            if (matrix[i] < matrix[i-1])

            {

                middle = matrix[i];

                matrix[i] = matrix[i-1];

                matrix[i-1] = middle;

                swapped\_elements = true;

            }

        }

        start = start + 1;

    }

}

void test\_cocktail\_best()

{

    std::vector<int> src = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    std::vector<int> expected = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    coctail(src);

    check\_results(src, expected, "Cocktail best case");

}

void test\_cocktail\_common()

{

    std::vector<int> src = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    std::shuffle(src.begin(), src.end(), \_tests\_random);

    std::vector<int> expected = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    coctail(src);

    check\_results(src, expected, "Cocktail common case");

}

void test\_cocktail\_worst()

{

    std::vector<int> src = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    std::reverse(src.begin(), src.end());

    std::vector<int> expected = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    coctail(src);

    check\_results(src, expected, "Cocktail worst case");

}

void test\_cocktail\_large\_stat(int iterations = 50)

{

    std::cout << "[test\_cocktail\_large\_stat] STARTED" << std::endl;

    auto start = std::chrono::high\_resolution\_clock::now();

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double, std::milli> duration = end - start;

    for(int i = 0; i < iterations; i++)

    {

        std::vector<int> src = get\_data\_for\_sort();

        start = std::chrono::high\_resolution\_clock::now();

        coctail(src);

        end = std::chrono::high\_resolution\_clock::now();

        std::chrono::duration<double, std::milli> duration = end - start;

        std::cout << std::to\_string(duration.count()) << std::endl;

    }

    std::cout << "[test\_cocktail\_large\_stat] ENDED" << std::endl;

}

ПРИЛОЖЕНИЕ B

Листинг кода файла cocktail.cpp

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include "tests.cpp"

#ifndef LAB\_HEAPSORT\_H

#define LAB\_HEAPSORT\_H

void heap\_sort(std::vector<int>& arr, int n, int i) {

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    if (left < n && arr[left] > arr[largest])

        largest = left;

    if (right < n && arr[right] > arr[largest])

        largest = right;

    if (largest != i) {

        std::swap(arr[i], arr[largest]);

        heap\_sort(arr, n, largest);

    }

}

std::vector<int> heap\_sort\_array(std::vector<int> arr) {

    int n = size(arr);

    for (int i = n / 2 - 1; i >= 0; i--)

        heap\_sort(arr, n, i);

    for (int i = n - 1; i > 0; i--) {

        std::swap(arr[0], arr[i]);

        heap\_sort(arr, i, 0);

    }

    return arr;

}

void test\_heap\_best()

{

    std::vector<int> src = {6000,5009,4000,5005,2000,2052,2055,4008,1000,1512,1900,2050,2051,2054,150,4002,4003,252,12,1001,13,6,16,2001,20,8,100,2053,102,9,3,4001,250,10,251,4,2,1};

    std::vector<int> expected = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    src = heap\_sort\_array(src);

    check\_results(src, expected, "Heap best case");

}

void test\_heap\_common()

{

    std::vector<int> src = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    std::shuffle(src.begin(), src.end(), \_tests\_random);

    std::vector<int> expected = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    src = heap\_sort\_array(src);

    check\_results(src, expected, "Heap common case");

}

void test\_heap\_worst()

{

    std::vector<int> src = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    std::vector<int> expected = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    src = heap\_sort\_array(src);

    check\_results(src, expected, "Heap worst case");

}

void test\_heap\_large\_stat(int iterations = 50)

{

    std::cout << "[test\_heap\_large\_stat] STARTED" << std::endl;

    auto start = std::chrono::high\_resolution\_clock::now();

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double, std::milli> duration = end - start;

    for(int i = 0; i < iterations; i++)

    {

        std::vector<int> src = get\_data\_for\_sort();

        start = std::chrono::high\_resolution\_clock::now();

        heap\_sort\_array(src);

        end = std::chrono::high\_resolution\_clock::now();

        std::chrono::duration<double, std::milli> duration = end - start;

        std::cout << std::to\_string(duration.count()) << std::endl;

    }

    std::cout << "[test\_heap\_large\_stat] ENDED" << std::endl;

}

#endif

ПРИЛОЖЕНИЕ C

Листинг кода файла bucket.cpp

#include <iostream>

#include <vector>

#include <cmath>

#include <chrono>

#include <algorithm>

#include "heapsort.cpp"

#include "tests.cpp"

void bucket\_sort(std::vector<int> &arr, int fixed\_length = 0)

{

    std::vector<std::vector<int>> baskets = {};

    int length = size(arr);

    int min = INT\_MAX;

    int max = INT\_MIN;

    int k = fixed\_length > 1 ? fixed\_length : pow(length, 0.5);

    int counter = 0;

    for (int i = 0; i < length; i++)

    {

        if(arr[i] > max)

        {

            max = arr[i];

            continue;

        }

        if(arr[i] < min)

        {

            min = arr[i];

            continue;

        }

        if (counter < k)

        {

            counter += 1;

            baskets.push\_back({});

        }

    }

    int metrics = max - min;

    if (k >= metrics || metrics == 0 || metrics == 1)

    {

        baskets = {};

    }

    for (int i = 0; i < length; i++)

    {

        if (size(baskets) != 0)

        {

            int position = arr[i] / metrics;

            baskets[position].push\_back(arr[i]);

        }

        else

        {

            baskets.push\_back(arr);

            break;

        }

    }

    int size\_buckets = size(baskets);

    for (int i = 0; i < size\_buckets; i++)

    {

        if (size(baskets[i]) > 1)

        {

            baskets[i] = heap\_sort\_array(baskets[i]);

        }

    }

    for (int i =0; i < size(baskets); i++)

    {

        if (size(baskets[i]) == 0)

        {

            baskets.erase(baskets.begin() + i);

        }

    }

    arr.clear();

    for (int i =0; i < size(baskets); i++)

    {

        if (size(baskets[i]) == 0)

        {

            baskets.erase(baskets.begin() + i);

            continue;

        }

        for (int j=0; j < size(baskets[i]); j++)

        {

            arr.push\_back(baskets[i][j]);

        }

    }

}

void test\_bucket\_best()

{

    std::vector<int> src = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    std::shuffle(src.begin(), src.end(), \_tests\_random);

    std::vector<int> expected = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    bucket\_sort(src, size(src)); // Uses bucket size 1

    check\_results(src, expected, "Bucket best case");

}

void test\_bucket\_common()

{

    std::vector<int> src = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    std::shuffle(src.begin(), src.end(), \_tests\_random);

    std::vector<int> expected = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    bucket\_sort(src, 0); // 0 - switches to dynamic bucket size calculation

    check\_results(src, expected, "Bucket common case");

}

void test\_bucket\_worst()

{

    std::vector<int> src = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    std::shuffle(src.begin(), src.end(), \_tests\_random);

    std::vector<int> expected = {1,2,3,4,6,8,9, 10, 12, 13,16,20, 100, 102, 150, 250, 251, 252, 1000, 1001, 1512, 1900, 2000, 2001, 2050, 2051, 2052, 2053, 2054, 2055, 4000, 4001, 4002, 4003, 4008, 5005, 5009, 6000};

    bucket\_sort(src, 2); // Worst nontrivial case, 1 - would be reduced to underlying sort complexity

    check\_results(src, expected, "Bucket worst case");

}

void test\_bucket\_large\_stat(int iterations = 50)

{

    std::cout << "[test\_bucket\_large\_stat] STARTED" << std::endl;

    auto start = std::chrono::high\_resolution\_clock::now();

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double, std::milli> duration = end - start;

    for(int i = 0; i < iterations; i++)

    {

        std::vector<int> src = get\_data\_for\_sort();

        start = std::chrono::high\_resolution\_clock::now();

        bucket\_sort(src);

        end = std::chrono::high\_resolution\_clock::now();

        std::chrono::duration<double, std::milli> duration = end - start;

        std::cout << std::to\_string(duration.count()) << std::endl;

    }

    std::cout << "[test\_bucket\_large\_stat] ENDED" << std::endl;

}

ПРИЛОЖЕНИЕ D

Листинг кода файла tests.cpp

#include <iostream>

#include <fstream>

#include <vector>

#include <chrono>

#include <string>

#include <random>

#ifndef LAB\_TESTS\_H

#define LAB\_TESTS\_H

std::random\_device \_tests\_random\_device;

std::default\_random\_engine \_tests\_random(\_tests\_random\_device());

std::vector<int> \_loaded\_vector;

void check\_results(std::vector<int> expected, std::vector<int> value, std::string test\_name)

{

    if (value == expected)

    {

        std::cout << "[" << test\_name << "] PASSED" << std::endl;

    }

    else

    {

        std::cout << "[" << test\_name <<"] FAILED" << std::endl;

    }

}

void load\_vector\_from\_file()

{

    \_loaded\_vector.clear();

    std::ifstream fin;

    fin.open("numbers.tsv");

    if (!fin.is\_open())

    {

        std::cout << "eror: file didn't opened";

    }

    std::string stri;

    while (!fin.eof())

    {

        std::getline(fin, stri);

    }

    fin.close();

    std::string current\_string;

    int current\_digit;

    for(int i=0; i < stri.size(); i++)

    {

        if(stri[i]!=';')

        {

            current\_string += stri[i];

        }

        else

        {

            current\_digit = std::stoi(current\_string);

            current\_string = "";

            \_loaded\_vector.push\_back(current\_digit);

        }

    }

}

std::vector<int> get\_data\_for\_sort()

{

    if (size(\_loaded\_vector) == 0) {

        load\_vector\_from\_file();

    }

    return \_loaded\_vector;

}

#endif

ПРИЛОЖЕНИЕ E

Листинг кода файла main.cpp

#include <iostream>

#include "cocktail.cpp"

#include "bucket.cpp"

#include "heapsort.cpp"

int main()

{

    std::cout << "cocktail sort:" << std::endl;

    test\_cocktail\_best();

    test\_cocktail\_common();

    test\_cocktail\_worst();

    test\_cocktail\_large\_stat();

    std::cout << "bucket sort:" << std::endl;

    test\_bucket\_best();

    test\_bucket\_common();

    test\_bucket\_worst();

    test\_bucket\_large\_stat();

    std::cout << "heap sort:" << std::endl;

    test\_heap\_best();

    test\_heap\_common();

    test\_heap\_worst();

    test\_heap\_large\_stat();

}