ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Алгоритмы сортировки»

Выполнил работу

Тиганов Вадим

Академическая группа J3112

Принято

Преподаватель, Дунаев Максим

Санкт-Петербург

2024

Структура отчёта:

1. Введение

1.1 Цель работы — изучение различных алгоритмов сортировки.

Задачи:

1.2 Выбрал для изучения алгоритмы сортировки: Selection sort, Insertion sort, Counting sort.

1.3 Реализовать все алгоритмы на языке c++

1.4 Произвести подсчеты по памяти для каждого из алгоритмов.

1.5 Произвести подсчеты асимптотики для каждого из алгоритмов.

1.6 Протестировать алгоритмы на массивах длины от тысячи до ста тысяч элементов.

1.7 Проанализировать полученные результаты и сделать выводы.

2. Теоретическая подготовка

Коротко про каждый из алгоритмов:

2.1 Selection sort — сортировка выбором:

Начинаем с первого элемента массива и будем проходить по каждому индексу.

В «неотсортированной» части массива ищем минимальный элемент и меняем местами с первым элементом этой части. Сдвигаем индекс на один вперед и повторяем процесс, таким

образом для каждой следующей части массива минимальный элемент будет перемещаться в его начало, а перед ним стоят уже раннее найденные также минимальные. Повторяем алгоритм для каждого индекса до предпоследнего, массив будет отсортирован.

Временная сложность алгоритма для любых случаев — *O(n^2)*: в любом случае для каждого элемента придется делать одинаковое количество операций.

Пространственная сложность — *O(1)*, т. к. не требуются дополнительные массивы и другие структуры данных, а каждая из переменных имеет фиксированный размер.

2.2 Insertion sort — сортировка вставками.

Insertion sort — сортировка вставками: Начинаем с первого элемента, который считается отсортированным. Берем следующий элемент и вставляем его в правильное место в уже отсортированной части массива. Повторяем процесс для всех элементов массива.

Временная сложность в худшем случае — *O(n^2)*, в среднем случае — *O(n^2)*, в лучшем случае (когда массив уже отсортирован) — *O(n)*.

Пространственная сложность — *O(1)*, так как не требуются дополнительные массивы или структуры данных.

2.3 Counting sort — сортировка подсчетом.

Определяем диапазон значений элементов массива. Создаем вспомогательный массив (часто называемый массивом подсчета) для хранения количества вхождений каждого элемента.

Заполняем массив подсчета, подсчитывая количество вхождений каждого элемента. Обновляем исходный массив, используя информацию из массива подсчета.

Временная сложность — *O(n + k)*, где n — количество элементов в массиве, а k — диапазон значений элементов.

Пространственная сложность — *O(n + k)*, так как требуется дополнительный массив для подсчета.

3. Реализация

3.1 Selection sort

```
#include <iostream>
#include <utility>
#include <vector>
#include <chrono>
#include <iomanip>
#include <fstream>
```

Импорт стандартных библиотек для ввода-вывода, чтения из файла, подсчета времени выполнения, регулирования точности измерений и реализации замены элементов местами.

Полный код см. в приложении А.

```
void selectionSort(std::vector<int>& array) {
    std::vector<int>::size_type n = array.size();
```

Из интересных наблюдений — пришлось использовать вместо типа данных int данную конструкцию по причине типовой безопасности и корректной работы с любыми массивами. (Мой компилятор при размерах более 100 тысяч не мог работать с

типом int) Та же практика была использована для объявления переменных, которые проходятся по индексам.

```
std::swap(&a: array[indexOfMinimal], &b: array[i]);
```

Использую встроенную функцию swap из библиотеки <utility>

```
auto start: time_point = std::chrono::high_resolution_clock::now();

selectionSort(&: array);

auto end: time_point = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = end - start;
```

```
std::cout << std::fixed << std::setprecision(n: 5);
std::cout << "SelectionSort: " << duration.count() << " seconds" << std::endl;
```

С помощью библиотеки <chrono> подсчитываю время работы алгоритма, с помощью <iomanip> форматирую до пяти знаков после запятой и вывожу.

3.2 Insertation sort

Использованы аналогичные библиотеки.

Снова пришлось использовать прием с определением счетчика в зависимости от длины массива, чтобы мой компилятор все устроило.

```
for (std::vector<int>::size_type i = 1; i < n; i++) {
    int key = array[i];
    int j = i - 1;

    while (j >= 0 && array[j] > key) {
        array[j + 1] = array[j];
        j = j - 1;
    }
    array[j + 1] = key;
}
```

Основа алгоритма на языке c++ (Полный код см. в приложении Б)

Работает простым перебором уже отсортированной части массива и ищет «место», куда можно вставить следующий элемент. Так проходим по всем элементам.

Функция подсчета времени аналогичная с первым алгоритмом.

3.3 Counting sort

Дополнительно использовал библиотеку <algorithm> для поиска максимального значения в массиве.

```
void countingSort(std::vector<int>& array) {
    if (array.empty()) return;

    int maxElement = *std::max_element(first: array.begin(), last: array.end());

    std::vector<int> count(n: maxElement + 1, value: 0);
```

Также «звездочка» позволяет получить конкретное значение найденного максимума. Далее инициализируем массив (вектор) длины, соответствующей максимальному элементу + 1 и для начала заполняем его нулями.

```
std::vector<int>::size_type index = 0;
for (int i = 0; i <= maxElement; i++) {
    while (count[i] > 0) {
        array[index++] = i;
        count[i]--;
    }
}
```

И самая интересная часть алгоритма. Подсчитал количество вхождений каждого элемента и записал в массив count. Создан вспомогательный массив index для записи отсортированных элементов в исходный массив. Проходим по массиву подсчета и записываем в изначальный в порядке возрастания. (Полный код см. в приложении В)

4. Экспериментальная часть

4.1 Линейный график времени работы алгоритмов с размером входных массивов от 1000 до 100.000 элементов. (Средний случай)

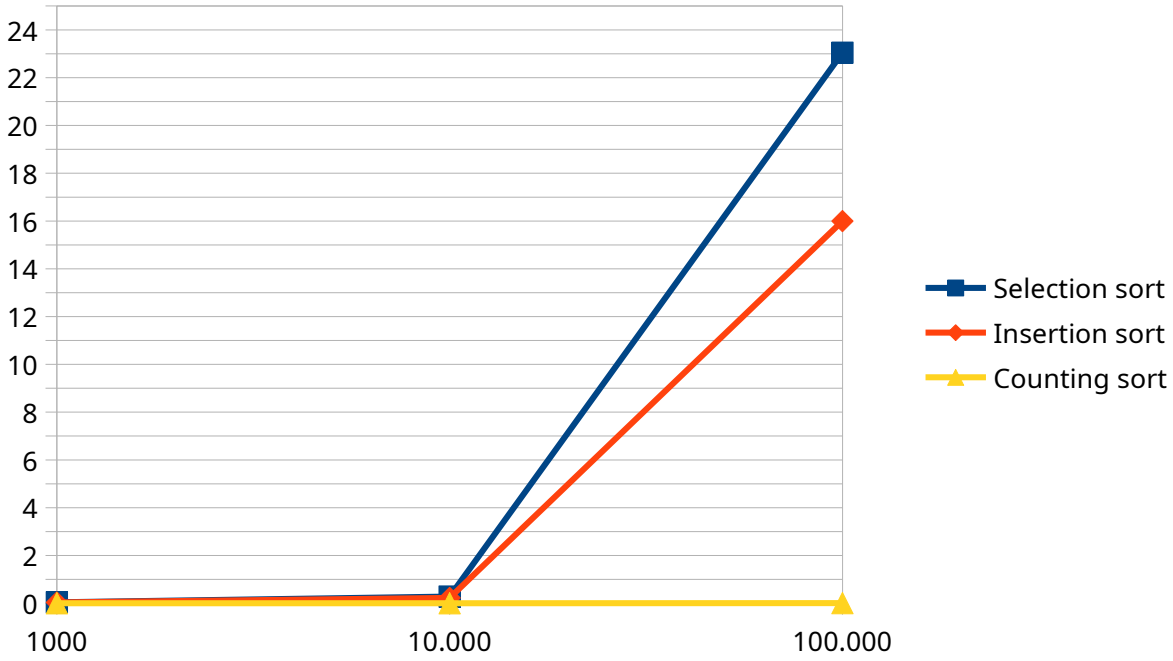

График 1. Линейный график работы алгоритмов. По горизонтальной оси — количество элементов, по вертикальной оси — время работы в секундах.

4.2 Box plot для трех алгоритмов (размеры массивов — 10.000 и 100.000 элементов) Ввиду очень высокой скорости работы Counting sort, график почти не виден.

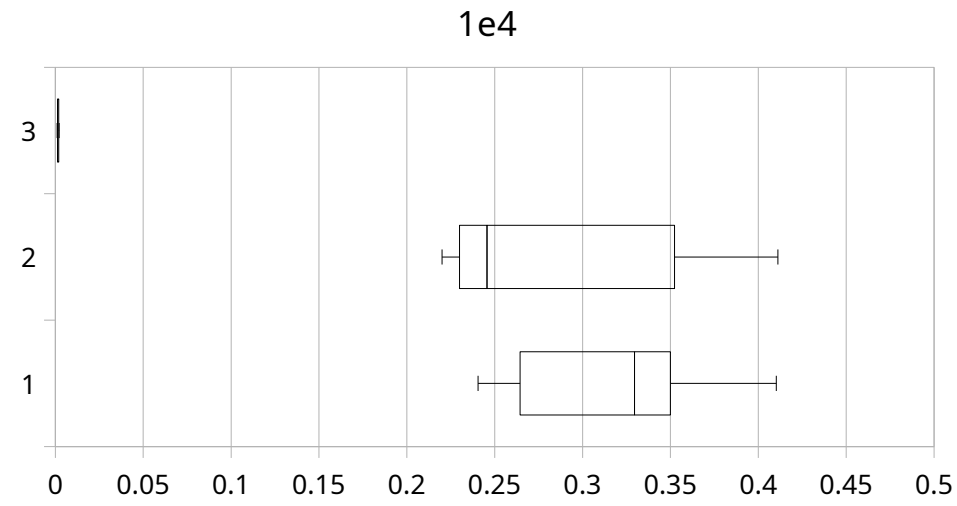

График 2. Для массивов размера 10.000 элементов. По горизонтальной оси — время работы алгоритма.

По вертикальной оси:

1) Selection sort

2) Insertion sort

3) Counting sort

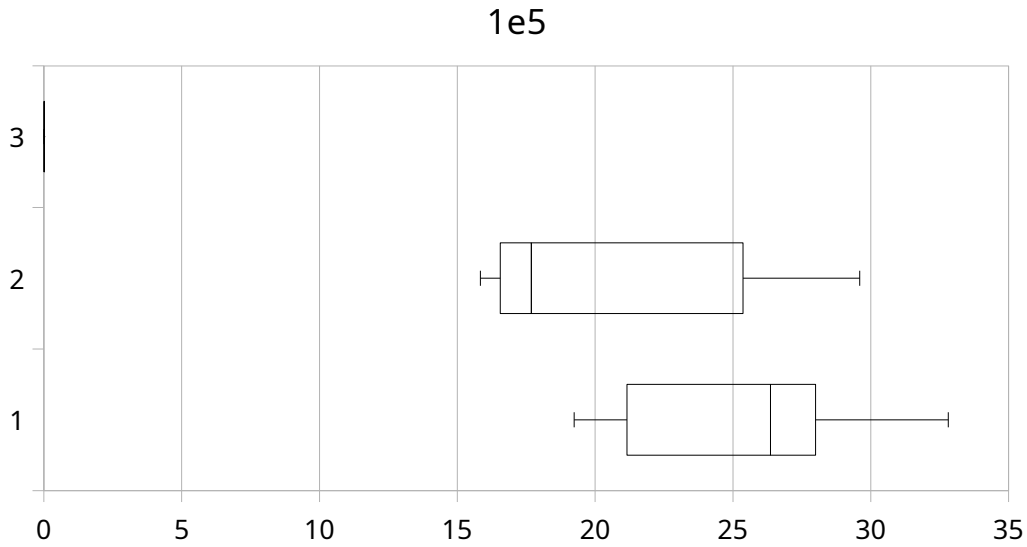

График 3. Для массивов размера 100.000 элементов. По горизонтальной оси — время работы алгоритма.

По вертикальной оси:

1) Selection sort

2) Insertion sort

3) Counting sort

Можно заметить, что во всех случаях с огромным отрывом по времени выигрывает алгоритм сортировки подсчетом.

5. Заключение

В ходе выполнения лабораторной работы были изучены и реализованы три алгоритма сортировки: Selection sort, Insertion sort и Counting sort. Каждый из алгоритмов имеет свои особенности и области применения.

Selection sort (сортировка выбором) проста в реализации и понимании, но неэффективна для больших массивов из-за своей квадратичной временной сложности. Она может быть полезна в условиях ограниченной памяти, так как не требует дополнительных массивов.

Insertion sort (сортировка вставками) также имеет квадратичную временную сложность в худшем и среднем случаях, но в лучшем случае, когда массив уже отсортирован, её временная сложность линейна. Это делает её эффективной для небольших или частично отсортированных массивов.

Counting sort (сортировка подсчетом) показала наилучшие результаты по времени выполнения, особенно для больших массивов. Она очень эффективна для массивов с ограниченным

диапазоном значений, но требует дополнительной памяти для массива подсчета.

Экспериментальные результаты показали, что Counting sort значительно превосходит по времени выполнения Selection sort и Insertion sort, особенно для больших массивов. Это подтверждается как линейными графиками времени работы, так и box plot'ами для массивов размером 10.000 и 100.000 элементов.

Таким образом, выбор алгоритма сортировки должен зависеть от конкретных условий задачи, таких как размер массива, диапазон значений элементов и доступная память.

6. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла selectionSort.cpp

```cpp
#include <iostream>
#include <utility>
#include <vector>
#include <chrono>
#include <iomanip>
#include <fstream>

void selectionSort(std::vector<int>& array) {
    std::vector<int>::size_type n = array.size();

    for (std::vector<int>::size_type i = 0; i < n - 1; i++) {
        std::vector<int>::size_type indexOfMinimal = i;
        for (std::vector<int>::size_type j = i + 1; j < n; j++) {
            if (array[j] < array[indexOfMinimal]) {
                indexOfMinimal = j;
            }
        }
        std::swap(array[indexOfMinimal], array[i]);
    }
}

void readFromFile(std::vector<int>& array, const std::string& filename) {
    std::ifstream file(filename);

    int number;
    while (file >> number) {
        array.push_back(number);
    }

    file.close();
}

int main() {
    std::vector<int> array;
    std::string filename = "numbers10e5.txt";

    readFromFile(array, filename);

    auto start = std::chrono::high_resolution_clock::now();

    selectionSort(array);

    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> duration = end - start;

    std::cout << std::endl;

    std::cout << std::fixed << std::setprecision(5);
    std::cout << "SelectionSort: " << duration.count() << " seconds" << std::endl;
    std::cout << "Array size: " << array.size() << std::endl;

    return 0;
}
```

ПРИЛОЖЕНИЕ Б

Листинг кода файла insertionSort.cpp

```cpp
#include <iostream>
#include <vector>
#include <chrono>
#include <iomanip>
#include <fstream>

void insertionSort(std::vector<int>& array) {
    std::vector<int>::size_type n = array.size();

    for (std::vector<int>::size_type i = 1; i < n; i++) {
        int key = array[i];
        int j = i - 1;

        while (j >= 0 && array[j] > key) {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
    }
}

void readFromFile(std::vector<int>& array, const std::string& filename) {
    std::ifstream file(filename);

    int number;
    while (file >> number) {
        array.push_back(number);
    }

    file.close();
}

int main() {
    std::vector<int> array;
    std::string filename = "numbers10e4.txt";

    readFromFile(array, filename);

    auto start = std::chrono::high_resolution_clock::now();

    insertionSort(array);

    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> duration = end - start;

    std::cout << std::endl;

    std::cout << std::fixed << std::setprecision(5);
    std::cout << "InsertionSort: " << duration.count() << " seconds" << std::endl;
    std::cout << "Array size: " << array.size() << std::endl;

    return 0;
}
```

ПРИЛОЖЕНИЕ В

Листинг кода файла countingSort.cpp

```cpp
#include <iostream>
#include <vector>
#include <chrono>
#include <iomanip>
#include <fstream>
#include <algorithm>

void countingSort(std::vector<int>& array) {
    if (array.empty()) return;

    int maxElement = *std::max_element(array.begin(), array.end());

    std::vector<int> count(maxElement + 1, 0);

    for (const auto& num : array) {
        count[num]++;
    }

    std::vector<int>::size_type index = 0;
    for (int i = 0; i <= maxElement; i++) {
        while (count[i] > 0) {
            array[index++] = i;
            count[i]--;
        }
    }
}

void readFromFile(std::vector<int>& array, const std::string& filename) {
    std::ifstream file(filename);

    int number;
    while (file >> number) {
        array.push_back(number);
    }

    file.close();
}

int main() {
    std::vector<int> array;
    std::string filename = "numbers10e5.txt";

    readFromFile(array, filename);

    auto start = std::chrono::high_resolution_clock::now();

    countingSort(array);

    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> duration = end - start;

    std::cout << std::endl;

    std::cout << std::fixed << std::setprecision(5);
    std::cout << "CountingSort: " << duration.count() << " seconds" << std::endl;
    std::cout << "Array size: " << array.size() << std::endl;

    return 0;
}
```