ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Сортировки»

Выполнил работу

Морозов Дмитрий

Академическая группа №J3110

Принято

Ментор, Вершинин Владислав

Санкт-Петербург

2024

1. Введение

Цель работы: реализовать три алгоритма сортировки, выбранных согласно заданным критериям. Оценить их временную и пространственную сложность

Задачи:

­­­­­­­­­­— Реализовать алгоритмы сортировки

— Проанализировать затрачиваемую память и время работы программ

— Построить линейные и box plot графики зависимости времени от входных данных

1. Реализация
   1. Анализ задачи, поиск алгоритмов сортировки, удовлетворяющих всем критериям

На данном этапе был проведен анализ поставленной задачи. Под каждый из пунктов был выбран подходящий алгоритм. Для первого случая используется exchange sort, для второго случая используется cocktail shaker sort, для третьего случая используется pigeonhole sort.

* 1. Написание программ
     1. Реализация exchange sort

На данном этапе был реализован алгоритм exchange sort. Данный алгоритм предполагает полный перебор всех пар в списке. Если элемент с меньшим индексом оказывается наибольшим в паре, то мы меняем элементы местами. Реализация сортировки на представлена на изображении №1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок №1 – Реализация exchange sort

* + 1. Реализация cocktail shaker sort

На данном этапе был реализован алгоритм cocktail shaker sort. За первый проход наибольший элемент помещается в конец списка, за второй проход наименьший элемент помещается в начало списка. Алгоритм продолжает работать с неотсортированной частью списка до тех пор, пока список не отсортирован. Реализация cocktail shaker sort представлена на изображении №2.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок №2 – Реализация cocktail shaker sort

* + 1. Реализация pigeonhole sort

На данном этапе был реализован алгоритм pigeonhole sort. Сначала определяются минимальный и максимальный элементы списка. Далее на основе полученных значений определяется максимальное количество чисел, которое может встретиться в списке. Создается вектор из нулей, куда будет записываться сколько раз встречается каждое число. После прохода по всем элементам исходного списка мы можем получить отсортированный список, пройдя по полученному вектору. Реализация pigeonhole sort представлена на изображении №3.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок №3 – Реализация pigeonhole sort

* + 1. Проверка работы с помощью тестов

На данном этапе были написаны тесты для проверки корректности программ. Тесты проверяли работу для следующих случаев:

— Тесты для exchange sort. Лучший – список уже отсортирован, средний – элементы в случайном порядке, худший – отсортирован в обратном порядке. Тесты представлены на изображении №4.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок №4 – Реализация тестов для exchange sort

— Тесты для cocktail shaker sort. Лучший – список уже отсортирован, средний – элементы в случайном порядке, худший – отсортирован в обратном порядке. Тесты представлены на изображении №5.

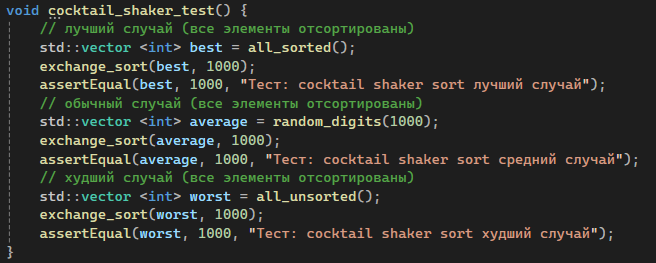


Рисунок №5 – Реализация тестов для cocktail shaker sort

— Тесты для pigeonhole sort. Лучший – все элементы одинаковые, средний – различны, но не все, худший – все элементы различны. Тесты представлены на изображении №6.

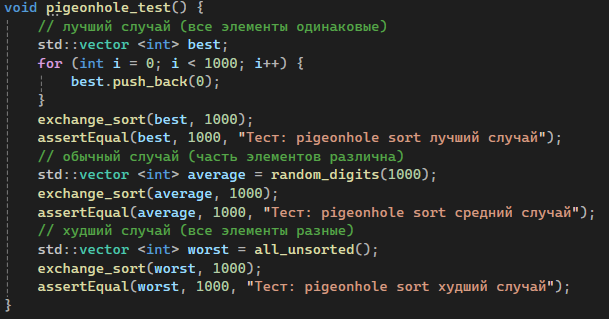


Рисунок №6 – Реализация pigeonhole sort

Реализация вывода результатов тестов представлена на изображении №7. В результате выводятся названия тестов, их статусы (пройден или нет), а также общее количество пройденных и непройденных тестов.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок №7 – Реализация тестов

* 1. Экспериментальная часть работы

На данном этапе производилась оценка затрачиваемой памяти и асимптотики. Также проводились замеры времени работы программы для разных входных данных. Для замера времени использовалась библиотека ctime. На основе этого был созданы графики, для удобного анализа зависимости роста времени работы от входных данных, а также среднего времени работы.

* 1. Анализ проделанной работы

На данном этапе был выполнен анализ проделанной работы, удалось ли достичь необходимую цель и выполнить задачи.

1. Экспериментальная часть

Подсчёт по памяти:

Функция exchange\_sort, представленная на изображении 1, имеет пространственную сложность O(1), так как не создает новых векторов. Функция cocktail\_shaker\_sort, представленная на изображении 2, имеет пространственную сложность O(1), так как не создаёт новых векторов. Функция pigeonhole\_sort, представленная на изображении 3, имеет пространственную сложность O(k), где k зависит от переменной range.

Подсчёт асимптотики:

Функция exchange\_sort, представленная на изображении 1, имеет временную сложность O(N^2) во всех случаях из-за перебора всех пар. Функция cocktail\_shaker\_sort, представленная на изображении 2, имеет временную сложность O(N^2) в среднем случае. Функция pigeonhole\_sort, представленная на изображении 3, имеет временную сложность O(N+k) в среднем, где k зависит от переменной range

График зависимости времени от числа элементов.

Был произведен замер времени работы алгоритмов при различных входных данных. Для pigeonhole sort были использованы массивы чисел размером от 1000 до 1000000 с шагом 1000. График, получившийся в результате продемонстрирован на изображении №8.

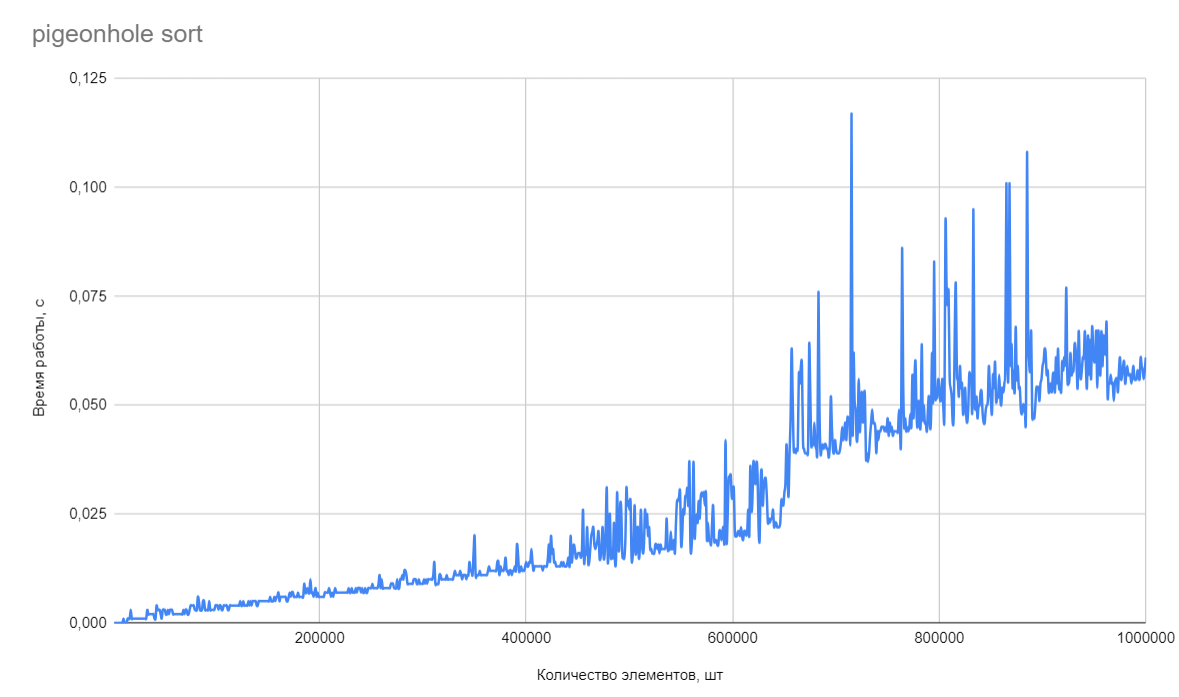


Рисунок №8 – Зависимость времени работы pigeonhole sort от количества элементов

Для остальных сортировок использовался размер от 10000 до 300000 и шаг в 10000 элементов. Результаты представлены на графике на изображении №9.

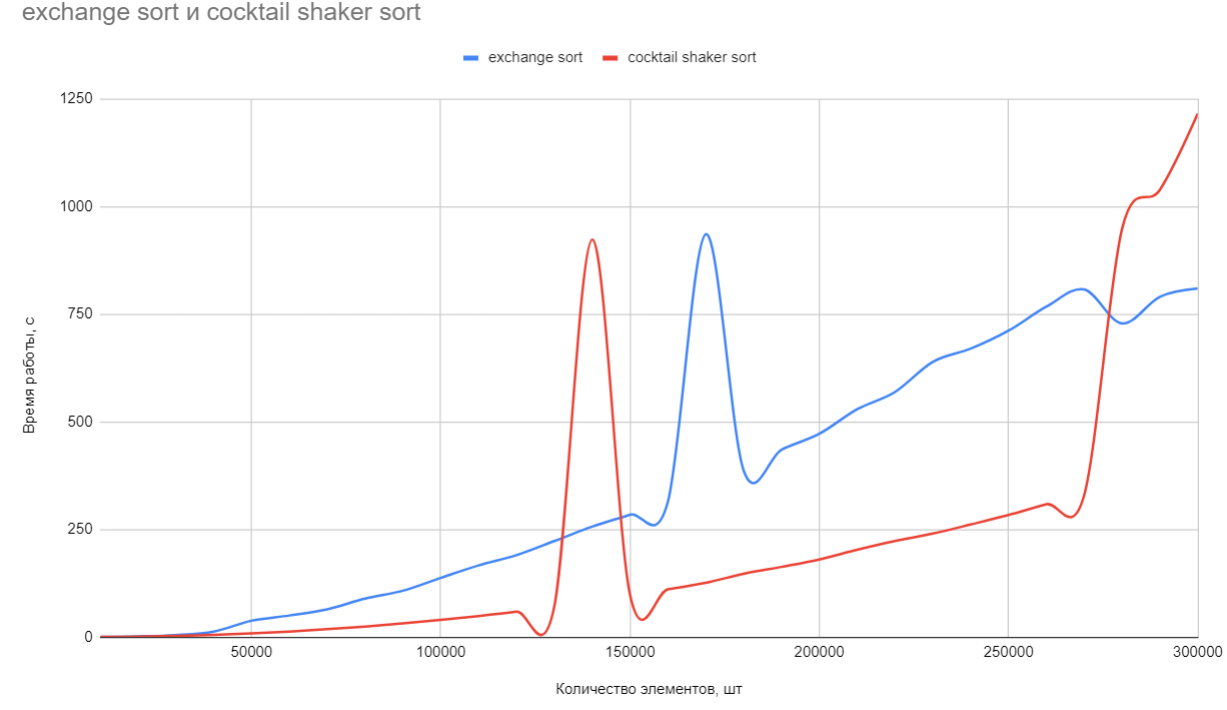


Рисунок №9 – Зависимость времени работы exchange sort и cocktail shaker sort sort от количества элементов

Полученные графики наглядно демонстрируют, что время работы pigeonhole sort растёт заметно меньше, чем у остальных сортировок. Также на графиках можно заметить резкие скачки времени работы – это плохие случаи, где алгоритм работал дольше среднего, так как массив значений оказалась не самым удачным для данной сортировки.

Также были построены box plot графики, чтобы увидеть среднее время работы на разном количестве элементов. Для каждой сортировки было выполнено 50 тестовых запусков, чтобы сделать влияние плохих случаев минимальным. Результаты для 1е4 элементов представлены на изображении №10. Для 1е5 элементов на изображении №11.

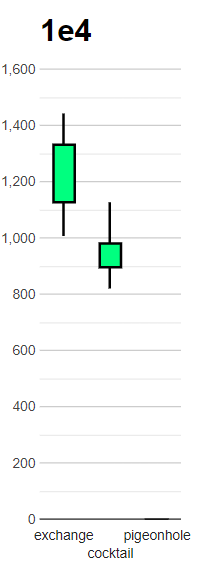


Рисунок №10 – box plot график для 1е4 элементов

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок №11 – box plot график для 1е5 элементов

Приведённые выше графики ещё лучше отображают, насколько быстрее работает pigeonhole. График её работы практически незаметен.

1. Заключение

Во многих программах используются алгоритмы сортировок. Вариантов их реализации существует довольно много, но каждый может быть полезен в определенной ситуации, поэтому следует анализировать задачу и подбирать соответствующий алгоритм, так как иначе можно получить слишком большое время работы.

ПРИЛОЖЕНИЕ А

Листинг кода файла lab\_5.cpp

#include <iostream>

#include <vector>

#include <ctime>

#include <Windows.h>

#include <algorithm>

#include <string>

void exchange\_sort(std::vector <int>& digits, const int& size) {

for (int i = 0; i < size - 1; i++) {

for (int j = i + 1; j < size; j++) {

if (digits[i] > digits[j]) {

std::swap(digits[i], digits[j]);

}

}

}

}

void cocktail\_shaker\_sort(std::vector <int>& digits, const int& size) {

bool swapped = true;

int start = 0;

int end = size - 1;

while (swapped) {

swapped = false;

for (int i = start; i < end; ++i) { // Первый проход: максимальный элемент помещается в конец

if (digits[i] > digits[i + 1]) {

std::swap(digits[i], digits[i + 1]);

swapped = true;

}

}

if (!swapped)

break;

swapped = false;

--end;

for (int i = end - 1; i >= start; --i) { // Второй проход: минимальный элемент помещается в конец

if (digits[i] > digits[i + 1]) {

std::swap(digits[i], digits[i + 1]);

swapped = true;

}

}

++start;

}

}

void pigeonhole\_sort(std::vector <int>& digits, const int& size) {

if (size > 0) {

int minimum = digits[0];

int maximum = digits[0];

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

if (digits[i] > maximum) {

maximum = digits[i];

}

else if (digits[i] < minimum) {

minimum = digits[i];

}

}

int range = maximum - minimum + 1; // Количество чисел, которые могут встретиться

std::vector <int> holes(range, 0);

for (int i = 0; i < size; i++) { // Сколько раз встречается каждое число

holes[digits[i] - minimum]++;

}

int index = 0;

for (int i = 0; i < range; ++i) {

while (holes[i] > 0) {

digits[index++] = i + minimum;

holes[i]--;

}

}

}

}

bool is\_sorted(const std::vector <int>& digits, const int& size) {

bool flag = true;

for (int i = 0; i < size - 1; i++) {

if (digits[i] > digits[i + 1]) {

flag = false;

}

}

return flag;

}

std::vector <int> random\_digits(const int& size) { // заполнение случайными числами

std::vector<int> random\_digits(size);

srand(time(0));

std::generate(random\_digits.begin(), random\_digits.end(), std::rand);

return random\_digits;

}

std::vector <int> all\_sorted() { // отсортированные числа

std::vector<int> vec;

for (int i = 0; i < 1000; i++) {

vec.push\_back(i);

}

return vec;

}

std::vector <int> all\_unsorted() { // неотсортированные числа

std::vector<int> vec;

for (int i = 1000; i > 0; i--) {

vec.push\_back(i);

}

return vec;

}

int test\_passed = 0; // Пройденные тесты

int test\_failed = 0; // Нейпройденные тесты

void assertEqual(const std::vector <int>& digits, const int& size, const std::string& testName) {

if (is\_sorted(digits, size) == true) {

std::cout << "[PASSED]" << testName << "\n";

test\_passed++;

}

else {

std::cout << "[NOT PASSED]" << testName << "\n";

test\_failed++;

}

}

void report() { // Отчёт по тестам

std::cout << "\nИтого тестов пройдено: " << test\_passed << "\n";

std::cout << "\nИтого тестов не пройдено: " << test\_failed << "\n";

}

void exchange\_test() {

// лучший случай (все элементы отсортированы)

std::vector <int> best = all\_sorted();

exchange\_sort(best, 1000);

assertEqual(best, 1000, "Тест: exchange sort лучший случай");

// обычный случай (все элементы отсортированы)

std::vector <int> average = random\_digits(1000);

exchange\_sort(average, 1000);

assertEqual(average, 1000, "Тест: exchange sort средний случай");

// худший случай (все элементы отсортированы)

std::vector <int> worst = all\_unsorted();

exchange\_sort(worst, 1000);

assertEqual(worst, 1000, "Тест: exchange sort худший случай");

}

void cocktail\_shaker\_test() {

// лучший случай (все элементы отсортированы)

std::vector <int> best = all\_sorted();

exchange\_sort(best, 1000);

assertEqual(best, 1000, "Тест: cocktail shaker sort лучший случай");

// обычный случай (все элементы отсортированы)

std::vector <int> average = random\_digits(1000);

exchange\_sort(average, 1000);

assertEqual(average, 1000, "Тест: cocktail shaker sort средний случай");

// худший случай (все элементы отсортированы)

std::vector <int> worst = all\_unsorted();

exchange\_sort(worst, 1000);

assertEqual(worst, 1000, "Тест: cocktail shaker sort худший случай");

}

void pigeonhole\_test() {

// лучший случай (все элементы одинаковые)

std::vector <int> best;

for (int i = 0; i < 1000; i++) {

best.push\_back(0);

}

exchange\_sort(best, 1000);

assertEqual(best, 1000, "Тест: pigeonhole sort лучший случай");

// обычный случай (часть элементов различна)

std::vector <int> average = random\_digits(1000);

exchange\_sort(average, 1000);

assertEqual(average, 1000, "Тест: pigeonhole sort средний случай");

// худший случай (все элементы разные)

std::vector <int> worst = all\_unsorted();

exchange\_sort(worst, 1000);

assertEqual(worst, 1000, "Тест: pigeonhole sort худший случай");

}

void run\_tests() { // Запуск тестов

// exchange sort

exchange\_test();

// cocktail shaker sort

cocktail\_shaker\_test();

// pigeonhole sort

pigeonhole\_test();

report();

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

run\_tests()

return 0;

}