МИНОБРНАУКИ РОСИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Восточно-Сибирский государственный университет технологий и управления»

(ФГБОУ ВО ВСГУТУ)

Факультет компьютерных наук и технологий

Кафедра программной инженерии и искусственного интеллекта

**ОТЧЕТ**по лабораторной работе №2  
на тему «Анализ практической сложности методов сортировки»

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: студент гр. Б763-2 Угнивенко Д.А.

Проверил: ассистент Николаева Т. В.

Улан-Удэ

2024

**Словесная постановка задачи:**

Выполнить программную реализацию методов сортировки: подсчётом, включением, извлечением, пузырек.

1. **Алгоритмы решения задачи:**
2. Сортировка подсчётом. Главная идея алгоритма — посчитать, сколько раз встречается каждый элемент в массиве, а потом заполнить исходный массив результатами этого подсчёта. Для этого нам нужен вспомогательный массив, где мы будем хранить результаты подсчёта.

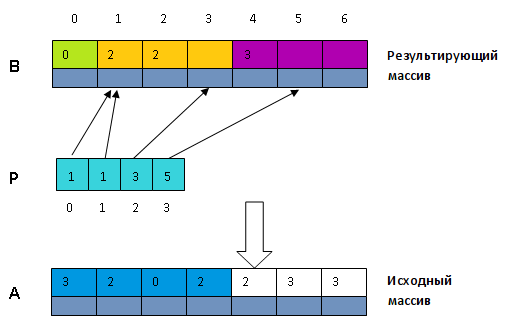


Рисунок 1 – Сортировка подсчетом

1. Сортировка включением. На каждом шаге алгоритма выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. Метод выбора очередного элемента из исходного массива произволен; может использоваться практически любой алгоритм выбора.

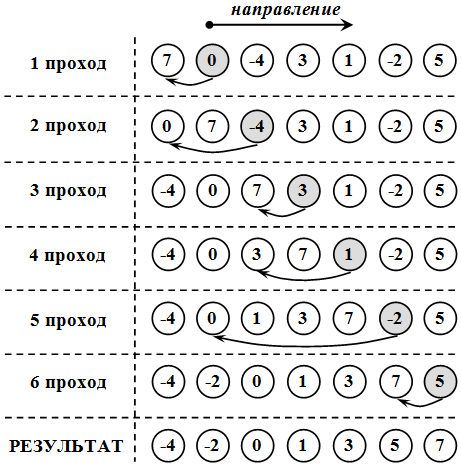


Рисунок 2 – Сортировка включением

1. Сортировка извлечением. Суть данного метода сортировки состоит в том, что в исходном массиве среди N элементов массива ищется элемент, имеющий минимальное (максимальное) значение. Он меняется местами с первым (последним) элементом массива. Элемент, ставший на своё место из дальнейшего рассмотрения, исключается. На следующей итерации рассматриваются N–1 элементов, и находится элемент, имеющий самое минимальное (максимальное) значение, который меняется местами со вторым (предпоследним) элементом массива. Для сортировки всех элементов массива необходимо выполнить N итераций.

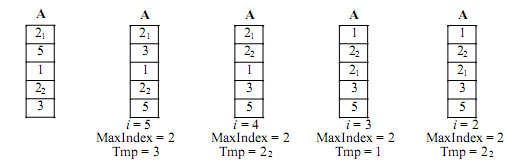


Рисунок 3 – Сортировка извлечением

1. Сортировка пузырьком. Суть данного метода сортировки состоит в том, что в исходном массиве среди N элементов массива ищется элемент, имеющий минимальное (максимальное) значение. Он меняется местами с первым (последним) элементом массива. Элемент, ставший на своё место из дальнейшего рассмотрения, исключается. На следующей итерации рассматриваются N–1 элементов, и находится элемент, имеющий самое минимальное (максимальное) значение, который меняется местами со вторым (предпоследним) элементом массива. Для сортировки всех элементов массива необходимо выполнить N итераций.

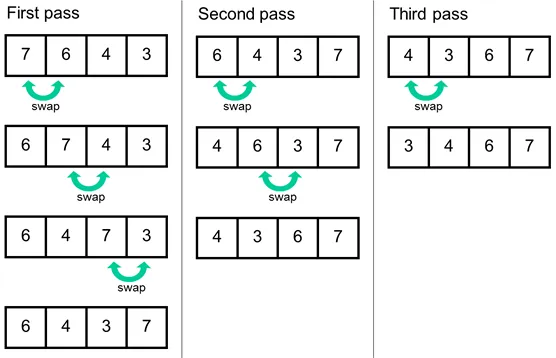


Рисунок 4 – Сортировка пузырьком

**Листинг программы:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <chrono>

#include <random>

#include <iomanip>

using namespace std;

using namespace std::chrono;

void countingSort(vector<int>& arr) {

if (arr.empty()) return; //проверка массива на пустоту.

int maxVal = \*max\_element(arr.begin(), arr.end()); //находит макс.значение массива для опр-ия размера всп-го массива

vector<int> count(maxVal + 1, 0); //созд-ие всп-ый массив count для подсчета кол-ва вхождения каждого эл-та

vector<int> output(arr.size()); //заполняем массив count кол-вом вхождений каждого эл-та

for (int num : arr) { //заполняет массив count кол-вом каждого вхождения

count[num]++;

}

for (int i = 1; i <= maxVal; i++) { //преобрезует массив count в массив накопленых сумм.

count[i] += count[i - 1];

}

for (int i = arr.size() - 1; i >= 0; i--) { //заполняет массив output отсорт-ми эл-ми

output[count[arr[i]] - 1] = arr[i];

count[arr[i]]--;

}

arr = output; //копирует отсорт-ый массив обратно в исходный

}

void insertionSort(vector<int>& arr) {

for (size\_t i = 1; i < arr.size(); i++) { // проходит по массиву с 2го эл-та

int key = arr[i]; // сохраняет текущий эл-т как ключ для сравнения.

int j = i - 1;

while (j >= 0 && arr[j] > key) { //сдвигает эл-ты, которые больше ключа, вправо

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;//вставляет ключ на правильное место

}

}

void selectionSort(vector<int>& arr) {

for (size\_t i = 0; i < arr.size() - 1; i++) { //проходит по массиву, выбирая мин-ый эл-т

size\_t minIndex = i; //инициализация мин эл-та

for (size\_t j = i + 1; j < arr.size(); j++) { //поиск мин эл-т

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

swap(arr[i], arr[minIndex]); //меняет местами текущий эл-т и мин найденный эл-т

}

}

void bubbleSort(vector<int>& arr) {

size\_t n = arr.size();

for (size\_t i = 0; i < n - 1; i++) {

for (size\_t j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

swap(arr[j], arr[j + 1]);

}

}

}

}

vector<int> randomArray(size\_t size) {

vector<int> arr(size);

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dis(0, 10000);

for (auto& num : arr) {

num = dis(gen);

}

return arr;

}

template <typename Func>

long long measureSortTime(Func sortFunc, vector<int> arr) {

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

sortFunc(arr);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

return duration.count();

}

int main() {

setlocale(LC\_ALL, "Russian");

vector<size\_t> sizes = { 1000, 3000, 5000, 7000, 10000 };

cout << "Размер\tПодсчетом\tВключением\tИзвлечением\t Пузырьком \n";

for (size\_t size : sizes) {

vector<int> arr = randomArray(size);

long long countingTime = measureSortTime(countingSort, arr);

long long insertionTime = measureSortTime(insertionSort, arr);

long long selectionTime = measureSortTime(selectionSort, arr);

long long bubbleTime = measureSortTime(bubbleSort, arr);

cout << size << "\t" << setw(5)

<< countingTime << " мкс" << "\t" << setw(6)

<< insertionTime << " мкс" << "\t" << setw(7)

<< selectionTime << " мкс" << "\t" << setw(6)

<< bubbleTime << " мкс" << "\n";

}

return 0;

}

1. **Тестирование:**

Задана последовательность элементов. Для чистоты эксперимента ищется последний элемент последовательности одной и той же последовательности, каждым методом сортировки.

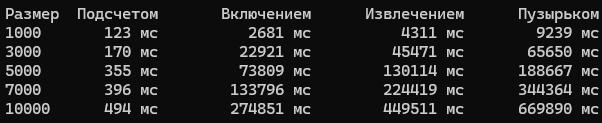


Рисунок 5 – Результаты работы всех сортировок

1. **Вычислительный эксперимент**

Для определения практической сложности алгоритмов сортировки были проведены вычислительные эксперименты. Время работы каждого метода определялось с помощью функции high\_resolution\_clock, посредством фиксации времени начала и конца работы метода. При помощи случайной сгенерированы 4 массива, количество элементов которых равны: 1000, 3000, 5000, 7000 и 10000.

Таблица 1 – Результаты эксперимента при удачном поиске

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов (size) | Время работы, миллисекунд (t) при сортировке подсчетом | Время работы, миллисекунд (t) при сортировке включением | Время работы, миллисекунд (t) сортировке извлечением | Время работы, миллисекунд (t) сортировке пузырьком |
| 1000 | 0,000123 | 0,002681 | 0,004311 | 0,009239 |
| 3000 | 0,000170 | 0,022921 | 0,045471 | 0,065650 |
| 5000 | 0,000355 | 0,073809 | 0,130114 | 0,188667 |
| 7000 | 0,000396 | 0,133796 | 0,224419 | 0,344364 |
| 10000 | 0,000494 | 0,274851 | 0,449511 | 0,669890 |

Рисунок 7 – График зависимости времени от объема выборки

**Заключение**

В ходе лабораторной работы я изучил алгоритмы сортировки, а именно подсчетом, включением, извлечением и пузырьком. И построил график зависимости времени t от количества элементов последовательности n. Тем самым определив различия работы по времени между этими методами.