**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №1**

**по дисциплине «Программирование»**

Тема: Одномерные статические массивы

|  |  |  |
| --- | --- | --- |
| Студентка гр. 2372 |  | Астафьева У.А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение массивов: объявление массивов, ввод-вывод массивов, различные сортировки массивов; получение практических навыков работы с одномерными статическими массивами; определение времени, затраченного на сортировку массива разными способами.

**Основные теоретические положения.**

Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение. Каждый элемент в этих массивах определяется значением индекса элемента.

Все массивы можно разделить на две группы: *одномерные* и *многомерные*. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.

Объявление в программах одномерных массивов выполняется в соответствии со следующим правилом:

<Базовый тип элементов> <Идентификатор массива> [<Количество элементов>]

Значения индексов элементов массивов всегда начинается с 0. Поэтому максимальное значение индекса элемента в массиве всегда на единицу меньше количества элементов в массиве.

Ввод и вывод массивов реализуются с помощью циклов.

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке.

Сортировка обменом (bubble sort). Вся суть метода заключается в попарном сравнении элементов и последующем обмене. Таким образом, если следующий элемент меньше текущего, то они меняются местами, максимальный элемент массива постепенно смещается в конец массива, а минимальный – в начало. Один полный проход по массиву может гарантировать, что в конце массива находится максимальный элемент. Затем процесс повторяется до тех пор, пока вся последовательность не будет упорядочена. Важно заметить, что после первого прохода по массиву, уже имеется один упорядоченный элемент, он стоит на своем месте, и менять его не надо. Таким образом на следующем шаге будут сравниваться *N*-1 элемент.

Шейкер-сортировка массива (shaker sort)– модификация пузырьковой сортировки. Принцип работы этой сортировки аналогичен bubble sort: попарное сравнение элементов и последующий обмен местами. Но имеется существенное отличие. Как только максимальный элемент становится на свое место, алгоритм не начинает новую итерацию с первого элемента, а запускает сортировку в обратную сторону. Алгоритм гарантирует, что после выполнения первой итерации, минимальный и максимальный элемент будут в начале и конце массива соответственно.Затем процесс повторяется до тех пор, пока массив не будет отсортирован.

Сортировка расческой (comb sort) – алгоритм был разработан специально для случаев, когда минимальные элементы стоят слишком далеко, или максимальные – слишком близко к началу массива. В сортировке расческой переставляются элементы, стоящие на расстоянии. Оптимально изначально взять расстояние равным длине массива, а далее уменьшать его на определенный коэффициент, который примерно равен 1.247. Когда расстояние станет равно 1, выполняется обычная сортировка пузырьком.

Сортировка вставками (insert sort) – алгоритм сортировки, в котором элементы массива просматриваются по одному, и каждый новый элемент размещается в подходящее место среди ранее упорядоченных элементов. Сортировка вставками делить массив на 2 части – отсортированную и неотсортированную. С каждым новым элементом отсортированная часть будет увеличиваться, а неотсортированная уменьшаться. Причем найти нужное место для очередного элемента в отсортированном массиве достаточно легко.

Алгоритм бинарного поиска – классический алгоритм поиска элемента в отсортированном массиве, который использует дробление массива на половины. Если элемент, который необходимо найти, присутствует в списке, то бинарный поиск возвращает ту позицию, в которой он был найден.

**Постановка задачи.**

Необходимо написать программу, которая:

1)    Создает целочисленный массив размерности *n* = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.

2)    Отсортировывает заданный в пункте 1 массив сортировкой (bubble sort, shaker sort, insert sort, comb sort) от меньшего к большему. Определяет время, затраченное на сортировки, используя библиотеку chrono.

3)    Находит максимальный и минимальный элемент массива. Подсчитывает время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.

4)    Выводит среднее значение максимального и минимального значения в отсортированном и неотсортированном массивах. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитывает время поиска.

5)    Выводит количество элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.

6)    Выводит количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

7)    Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве (реализация бинарным поиском). Сравнивает скорость его работы с обычным перебором.

8)     Меняет местами элементы массива, индексы которых вводит пользователь. Выводит скорость обмена, используя библиотеку chrono.

**Выполнение работы.**

Код программы представлен в приложении А.

1. При запуске программы пользователю выводится неотсортированный массив из 100 элементов, которые принимают случайное значение в диапазоне от -99 до 99.



Рисунок . Задача 1 – вывод неотсортированного массива

1. 1) Далее на экран выводится отсортированный массив из п.1.



Рисунок . Задача 2.1 – вывод отсортированного массива

2) После вывода отсортированного массива, на экран выводится время, которое было затрачено на сортировку массива реализованными сортировками: bubble sort, shaker sort, insert sort, comb sort.

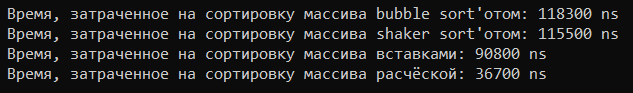


Рисунок . Задача 2.2 – время работы разных сортировок массива

1. Далее выводится максимальный и минимальный элементы массива и время поиска этих элементов в отсортированном и неотсортированном массивах

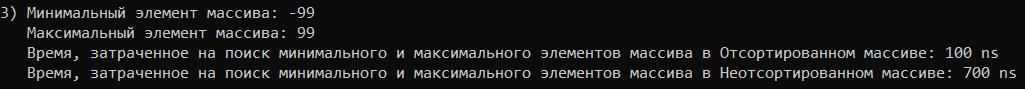


Рисунок 4. Задача 3

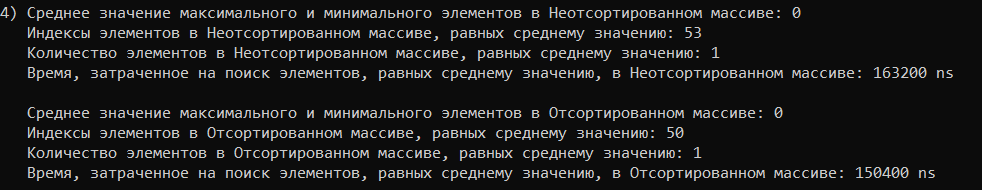
1. После этого выводятся среднее значение максимального и минимального значения в отсортированном и неотсортированном массивах. А также выводятся индексы всех элементов, которые равны этому значению, и их количество, подсчёт времени поиска. 

Рисунок 5. Задача 4

1. Далее пользователю предлагается ввести число. Программа выводит количество элементов в отсортированном массиве, которые меньше этого числа.



Рисунок 6. Задача 5

1. После, пользователю предлагается ввести другое число. Программа выводит количество элементов в отсортированном массиве, которые больше этого числа.



Рисунок 7. Задача 6

1. Дальше пользователю предлагается ввести число. Программа с помощью бинарного поиска определяет, есть ли это число в массиве, и выводит индекс этого элемента в отсортированном массиве (при наличии элемента в массиве).

Если числа в массиве нет, то выводит «Вашего числа нет в массиве :(». То же самое с обычным перебором элементов. Сравнивается скорость работы этих алгоритмов.

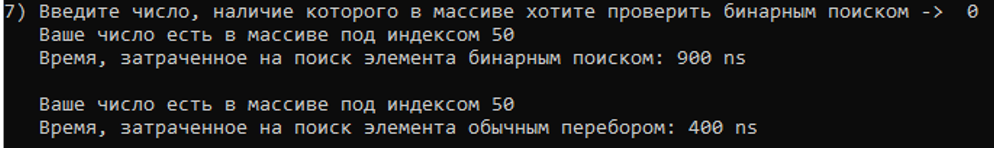


Рисунок 8. Задача 7

1. После этого пользователю предлагается ввести два индекса, элементы которых в массиве (отсортированном) нужно поменять местами. Для наглядности на экран выводится весь массив с уже перемещёнными элементами. Также выводится время скорости обмена элементов.

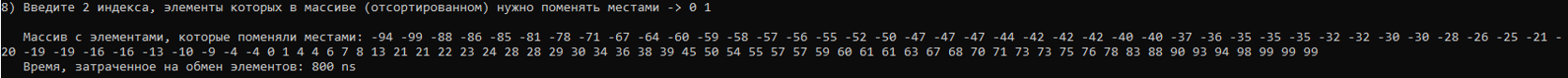
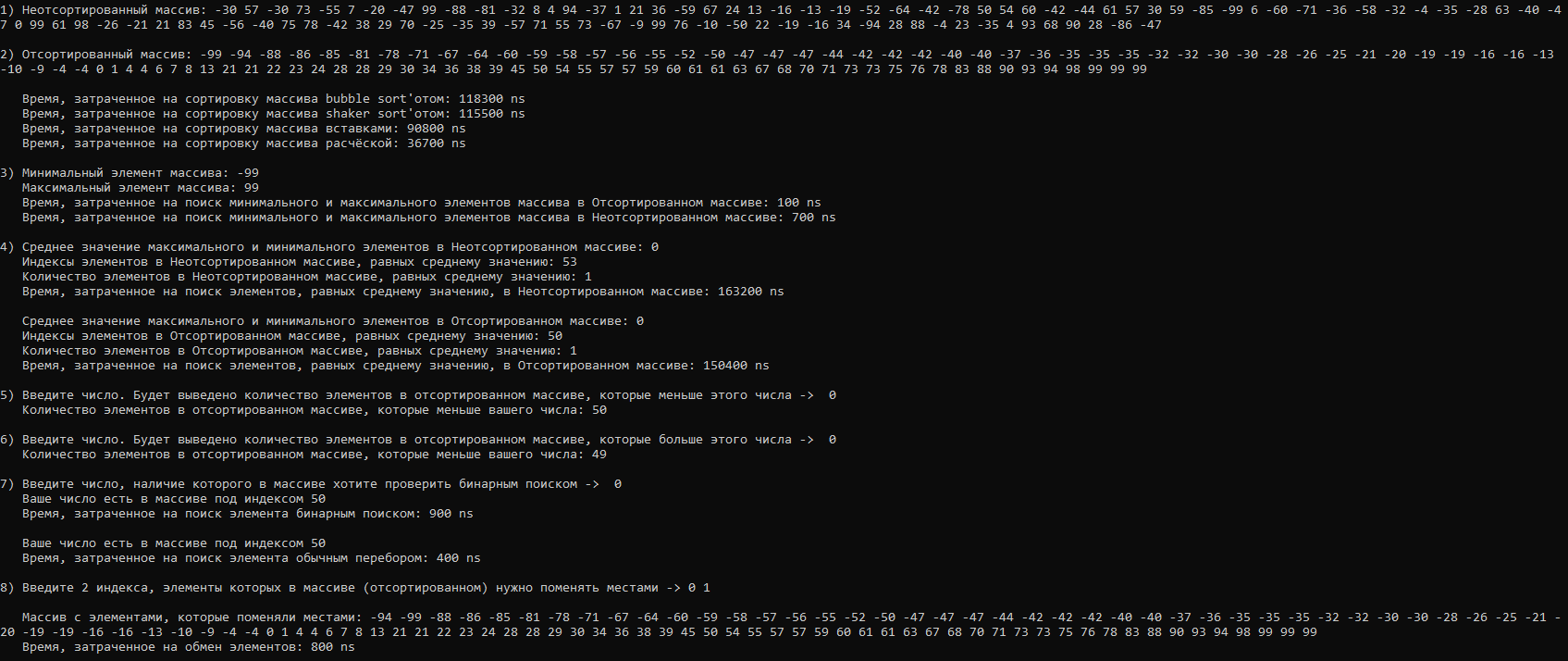


Рисунок 9. Задача 8

Общий результат кода с примерами:



**Выводы.**

В ходе работы было изучено: объявление массивов, ввод-вывод массивов, сортировки массивов: bubble sort, shaker sort, insert sort, comb sort; определение времени, затраченного на сортировку массива разными способами.

Приложение А

рабочий код

#include <iostream>

#include <algorithm>

#include <chrono>

#include <ctime>

using namespace std;

using namespace chrono;

int main()

{

setlocale(0, "");

srand(time(0));

const int n = 100;

int arr[n]{};

int copyArr[n]{};

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

// 1 task

cout << "\n1) Неотсортированный массив: ";

for (int v = 0; v < n; v++)

{

arr[v] = rand() % 199 - 99;

cout << arr[v] << ' ';

}

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

copyArr[i] = arr[i];

// 2 task

start = steady\_clock::now();

for (int i = 0; i < n - 1; i++) // bubble sort

{

for (int j = 0; j < n - 1 - i; j++)

{

if (copyArr[j] > copyArr[j + 1])

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

}

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "\n\n2) Отсортированный массив: ";

for (int w = 0; w < n; w++)

cout << copyArr[w] << ' ';

cout << "\n\n Время, затраченное на сортировку массива bubble sort'отом: " << result.count() << " ns";

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

copyArr[i] = arr[i];

start = steady\_clock::now();

for (int i = 0; i < n - 1; i++) // shaker sort

{

for (int j = i; j < n - 1 - i; j++)

{

if (copyArr[j] > copyArr[j + 1])

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

}

for (int l = n - 2 - i; l > i; l--)

{

if (copyArr[l] < copyArr[l - 1])

swap(copyArr[l], copyArr[l - 1]);

}

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "\n Время, затраченное на сортировку массива shaker sort'отом: " << result.count() << " ns";

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

copyArr[i] = arr[i];

start = steady\_clock::now();

for (int i = 1; i < n; i++) // insert sort

for (int j = i; copyArr[j - 1] > copyArr[j]; j--)

swap(copyArr[j - 1], copyArr[j]);

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "\n Время, затраченное на сортировку массива вставками: " << result.count() << " ns";

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

copyArr[i] = arr[i];

float k = 1.247;

int step = n - 1;

bool flag = true;

start = steady\_clock::now();

while (step >= 1) // comb sort

{

for (int i = 0; i + step < n; i++)

{

if (copyArr[i] > copyArr[int(i + step)])

swap(copyArr[i], copyArr[int(i + step)]);

}

step = step / k;

}

while (flag)

{

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

{

flag = false;

for (int j = 0; j < n - 1 - i; j++)

{

if (copyArr[j] > copyArr[j + 1])

{

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

flag = true;

}

}

}

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "\n Время, затраченное на сортировку массива расчёской: " << result.count() << " ns";

// 3 task

start = steady\_clock::now();

int minOm = copyArr[0];

int maxOm = copyArr[n - 1];

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "\n\n3) Минимальный элемент массива: " << minOm;

cout << "\n Максимальный элемент массива: " << maxOm;

cout << "\n Время, затраченное на поиск минимального и максимального элементов массива в Oтсортированном массиве: " << result.count() << " ns";

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

copyArr[i] = arr[i];

int maxNm = -100;

int minNm = 100;

start = steady\_clock::now();

for (int l = 0; l < n; l++)

{

if (copyArr[l] > maxNm)

maxNm = copyArr[l];

if (copyArr[l] < minNm)

minNm = copyArr[l];

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "\n Время, затраченное на поиск минимального и максимального элементов массива в Неотсортированном массиве: " << result.count() << " ns";

// 4 task

int midNm, count = 0;

midNm = (minNm + maxNm) / 2;

cout << "\n\n4) Среднее значение максимального и минимального элементов в Неотсортированном массиве: " << midNm;

cout << "\n Индексы элементов в Неотсортированном массиве, равных среднему значению: ";

start = steady\_clock::now();

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

{

if (copyArr[i] == midNm)

{

count += 1;

cout << i << " ";

}

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "\n Количество элементов в Неотсортированном массиве, равных среднему значению: " << count;

cout << "\n Время, затраченное на поиск элементов, равных среднему значению, в Неотсортированном массиве: " << result.count() << " ns";

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

{

for (int j = 0; j < n - 1 - i; j++)

{

if (copyArr[j] > copyArr[j + 1])

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

}

}

int midOm;

count = 0;

int in = 100;

midOm = (copyArr[0] + copyArr[n - 1]) / 2;

cout << "\n\n Среднее значение максимального и минимального элементов в Oтсортированном массиве: " << midOm;

cout << "\n Индексы элементов в Oтсортированном массиве, равных среднему значению: ";

for (int u = 0; u < n; u++)

if (copyArr[u] > midOm)

in = u - 1;

start = steady\_clock::now();

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

{

while (copyArr[i] <= midOm)

{

if (copyArr[i] == midOm)

{

count += 1;

cout << i << " ";

}

break;

}

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "\n Количество элементов в Oтсортированном массиве, равных среднему значению: " << count;

cout << "\n Время, затраченное на поиск элементов, равных среднему значению, в Oтсортированном массиве: " << result.count() << " ns";

// 5 task

int a;

count = 0;

cout << "\n\n5) Введите число. Будет выведено количество элементов в отсортированном массиве, которые меньше этого числа -> ";

cin >> a;

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

{

if (copyArr[i] < a)

count += 1;

if (copyArr[i] > a)

break;

}

cout << " Kоличество элементов в отсортированном массиве, которые меньше вашего числа: " << count;

// 6 task

int b;

count = 0;

cout << "\n\n6) Введите число. Будет выведено количество элементов в отсортированном массиве, которые больше этого числа -> ";

cin >> b;

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

{

if (copyArr[i] > b)

count += 1;

}

cout << " Kоличество элементов в отсортированном массиве, которые больше вашего числа: " << count;

// 7 task (\*)

int key;

cout << "\n\n7) Введите число, наличие которого в массиве хотите проверить бинарным поиском -> ";

cin >> key;

int midIndex;

int left = 0;

int right = n - 1;

bool flagBin = false;

start = steady\_clock::now();

while (left <= right) // бинарный поиск

{

midIndex = (left + right) / 2;

if (copyArr[midIndex] == key)

{

flagBin = true;

break;

}

if (copyArr[midIndex] > key)

right = midIndex - 1;

else

left = midIndex + 1;

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

if (flagBin == true)

cout << " Ваше число есть в массиве под индексом " << midIndex;

else

cout << " Вашего числа нет в массиве :(";

cout << "\n Время, затраченное на поиск элемента бинарным поиском: " << result.count() << " ns";

int countI = 0;

start = steady\_clock::now();

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

{

if (copyArr[i] == key)

{

countI += 1;

break;

}

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

if (countI > 0)

cout << "\n\n Ваше число есть в массиве под индексом " << midIndex;

else

cout << "\n Вашего числа нет в массиве :(";

cout << "\n Время, затраченное на поиск элемента обычным перебором: " << result.count() << " ns";

// 8 task

int el1, el2;

cout << "\n\n8) Введите 2 индекса, элементы которых в массиве (отсортированном) нужно поменять местами -> ";

cin >> el1;

cin >> el2;

start = steady\_clock::now();

swap(copyArr[el1], copyArr[el2]);

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "\n Массив с элементами, которые поменяли местами: ";

for (int w = 0; w < n; w++)

cout << copyArr[w] << ' ';

cout << "\n Время, затраченное на обмен элементов: " << result.count() << " ns";

return 0;

}