**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: Одномерные статические массивы

|  |  |  |
| --- | --- | --- |
| Студент гр. 2372 | Юрин А. А. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

**2022**

**Цель работы.**

Изучение одномерных статических массивов и сортировок. Написание программы, показывающей разницу между скоростями различных сортировок, а также изучение принципа бинарного поиска и его реализации.

**Основные теоретические положения.**

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнить однообразные действия, им дают одно имя, а различают по порядковому номеру (индексу). Это дает возможность компактно записать множество операций с использованием циклов.

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.

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

Обмен элементов массива осуществляется через буферную переменную либо через функцию swap (a, b).

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

Простейшая из сортировок – сортировка обменом (пузырьковая сортировка). 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.

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. За счет того, что сортировка работает в обе стороны, массив сортируется на порядок быстрее. Очевидным примером этого был бы случай, когда в начале массива стоит максимальный элемент, а в конце массива – минимальный. Shaker sort справится с этим за 1 итерацию, при условии, что другие элементы стоят на правильном месте.

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.

Общая суть сортировки вставками такова:

1)    Перебираются элементы в неотсортированной части массива.

2)    Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находится.

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.

Рассмотрим самый простой способ (рис. 3.5). Необходимо пройти массив слева направо и обработать каждый элемент. Слева будет наращиваться отсортированная часть массива, а справа – уменьшаться неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер, что освобождает место в массиве и позволяет сдвинуть элементы и освободить точку вставки.

**Постановка задачи.**

Разработать алгоритм и написать программу, которая позволяет:

1. Создать целочисленный массив размерности N = 100. Элементы массива должны принимать случайное значение в диапазоне от -99 до 99.
2. Отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.
3. Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.
4. Вывести среднее значение (если необходимо, число нужно округлить) максимального и минимального в отсортированном и неотсортированном. Вывести индексы всех элементов, которые равны этому значению, и их количество. Подсчитать время поиска.
5. Вывести количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем.
6. Вывести количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем.
7. Вывести информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализовать алгоритм бинарного поиска. Сравнить скорость его работы с обычным перебором.
8. Поменять местами элементы массива, индексы которых вводит пользователь. Вывести скорость обмена, используя библиотеку chrono.

**Выполнение работы.**

Код программы представлен в приложении 1.

1. При запуске программы ожидается ввод команды с клавиатуры для запуска цикла (рис. 1).

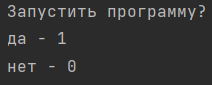
****

Рисунок 1. Запуск программы

1. Следующий шаг зависит от введенной команды, если пользователь ввёл:
   1. Если пользователь введёт “0”, то выполнение программы завершается.
   2. Если “1”, то выводится строка, которая предлагает создать массив.
2. После запуска предлагается выбор (рис. 2):
   1. Если пользователь введёт “0”, то цикл завершится и массив не будет создан.
   2. Если “1”, то программа создаст неотсортированный массив.

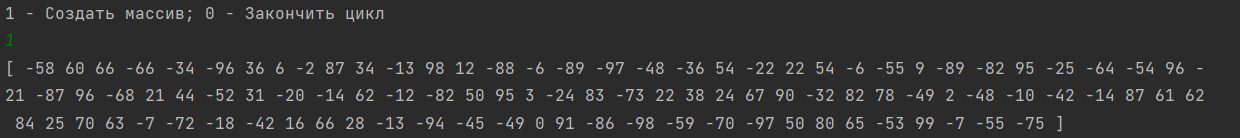


Рисунок 2. Создание массива

1. Далее пользователю будет предложено отсортировать массив:
   1. Если пользователь ввёл “0”, то цикл завершится и массив не будет отсортирован.
   2. Если “1”, то программа предложит отсортировать массив с помощью трёх на выбор сортировок: Bubble sort, Shaker sort или Insert sort.
2. После выбора сортировки в терминал будет выведен отсортированный массив и скорость его сортировки:

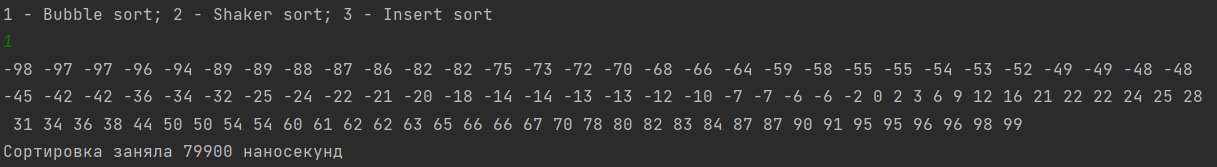


Рисунок 3. Bubble sort.

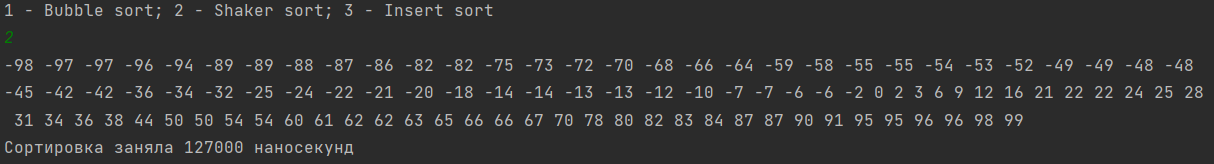


Рисунок 4. Shaker sort.

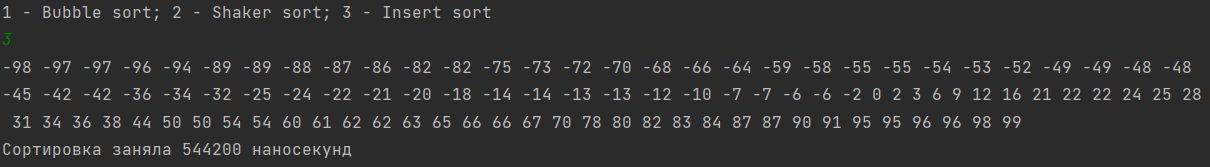


Рисунок 5. Insert sort.

1. Далее пользователю будет предложено вывести максимальный и минимальный элемент массива с помощью отсортированного или неотсортированного массива, чтобы сравнить скорость их нахождения.
   1. Сортированный массив:

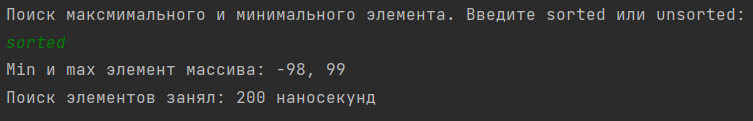


Рисунок 6. Sorted.

* 1. Неотсортированный массив:

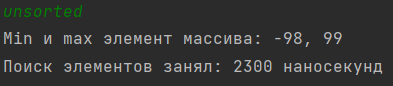


Рисунок 7. Unsorted.

1. Сразу после выполнения прошлой программы в терминал будет выведено среднее значение максимального и минимального в отсортированном и неотсортированном массиве. Также будут выведены индексы всех элементов, которые равны этому значению, их количество, и время поиска.



Рисунок 8. Avg.

1. Далее пользователю будет предложено вывести количество элементов в отсортированном массиве, которые меньше числа a, инициализируемым им же.

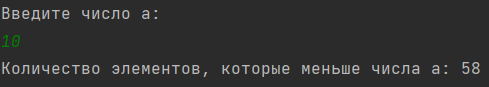


Рисунок 9. То, что меньше числа а.

1. Далее пользователю будет предложено вывести количество элементов в отсортированном массиве, которые больше числа b, инициализируемым им же.

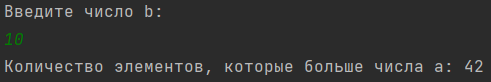


Рисунок 10. То, что больше числа b.

1. Следующий шаг – бинарный поиск и перебор, а также их сравнение в скорости. Пользователь должен ввести число, которое есть в данном массиве от -99 до 99. Например “3”.

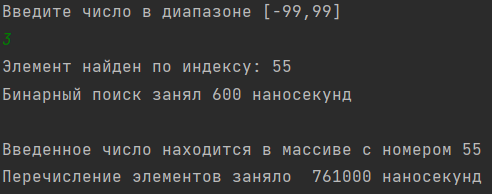
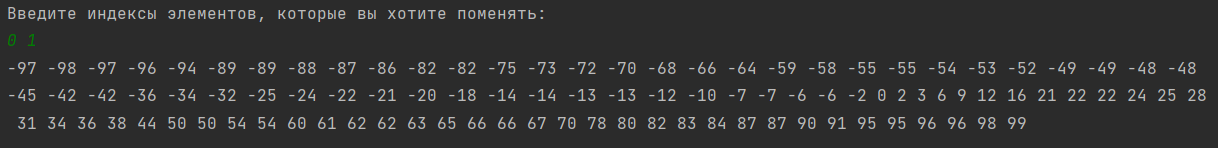


Рисунок 11. Бинарный поиск.

1. Последний цикл в программе меняет местами элементы массива, индексы которых введет пользователь.



**Вывод.**

Мы изучили одномерные статические массивы и сортировки. Написали программу, показывающую разницу между скоростями различных сортировок, а также изучили принцип бинарного поиска и его реализацию.

**Приложение**

**Код**

#include <iostream>  
#include <chrono>  
#include <string>  
#include <cmath>  
  
using namespace std;  
using namespace chrono;  
  
const int N = 100;  
  
void coutarr(int arr[], int N) {  
 for (int i = 0; i < N; i++)  
 cout << arr[i] << " ";  
}  
  
void bubblesort(int \*arr, int N) {  
 for (int i = 0; i < N - 1; i++)  
 for (int j = 0; j < N - i - 1; j++)  
 if (arr[j] > arr[j + 1]) {  
 swap(arr[j], arr[j + 1]);  
 }  
}  
  
void shakersort(int \*arr, int N) {  
 for (int j = 0; j < N / 2; j++) {  
 for (int i = 0; i < N - 1; i++) {  
 if (arr[i] > arr[i + 1])  
 swap(arr[i], arr[i + 1]);  
 }  
 for (int i = N - 2; i > 0; i--) {  
 if (arr[i] < arr[i - 1])  
 swap(arr[i], arr[i - 1]);  
 }  
 }  
}  
  
void insertsort(int \*arr, int N) {  
 for (int i = 1; i < N; i++) {  
 for (int j = i - 1; j >= 0; j--)  
 if (arr[i] < arr[j]) {  
 swap(arr[i], arr[j]);  
 i -= 1;  
 }  
 }  
}  
  
int binarysearch(int arr[], int x, int low, int high) {  
 while (low <= high) {  
 int mid = low + (high - low) / 2;  
 if (arr[mid] == x)  
 return mid;  
 if (arr[mid] < x)  
 low = mid + 1;  
 else  
 high = mid - 1;  
 }  
 return -1;  
}  
  
int main() {  
 setlocale(LC\_ALL, "ru\_RU");  
 int arr[N] = {};  
 int commander;  
 cout << "Запустить программу?\nда - 1\nнет - 0" << endl;  
 cin >> commander;  
 switch (commander) {  
 case 1: {  
 // 1. Целочисленный массив из 100 элементов  
 cout << "1 - Создать массив; 0 - Закончить цикл" << endl;  
 int firststart;  
 cin >> firststart;  
 while (firststart) {  
 cout << "[ ";  
 for (int i = 0; i < N; i++) {  
 arr[i] = -99 + rand() % (199);  
 cout << arr[i] << " ";  
 }  
 cout << "]" << endl;  
 break;  
 }  
 }  
 case 2: {  
 // 2. Сортировка массива  
 int sortname, cycleone, bubble[N] = {}, shaker[N] = {}, insert[N] = {};  
 cout << "1 - Отсортировать массив; 0 - Закончить цикл" << endl;  
 cin >> cycleone;  
 while (cycleone) {  
 cout << "1 - Bubble sort; 2 - Shaker sort; 3 - Insert sort" << endl;  
 cin >> sortname;  
 switch (sortname) {  
 case 1: {  
 auto start = steady\_clock::now();  
 for (int i = 0; i < N; i++) {  
 bubble[i] = arr[i];  
 }  
 bubblesort(bubble, N);  
 auto end = steady\_clock::now();  
 auto elapsed = duration\_cast<nanoseconds>(end - start);  
 coutarr(bubble, N);  
 cout << endl;  
 cout << "Сортировка заняла " << elapsed.count() << " наносекунд\n";  
 cout << endl;  
 break;  
 }  
 case 2: {  
 auto start = steady\_clock::now();  
 for (int i = 0; i < N; i++) {  
 shaker[i] = arr[i];  
 }  
 shakersort(shaker, N);  
 auto end = steady\_clock::now();  
 auto elapsed = duration\_cast<nanoseconds>(end - start);  
 coutarr(shaker, N);  
 cout << endl;  
 cout << "Сортировка заняла " << elapsed.count() << " наносекунд\n";  
 cout << endl;  
 break;  
 }  
 case 3: {  
 auto start = steady\_clock::now();  
 for (int i = 0; i < N; i++) {  
 insert[i] = arr[i];  
 }  
 insertsort(insert, N);  
 auto end = steady\_clock::now();  
 auto elapsed = duration\_cast<nanoseconds>(end - start);  
 coutarr(insert, N);  
 cout << endl;  
 cout << "Сортировка заняла " << elapsed.count() << " наносекунд";  
 cout << endl;  
 break;  
 }  
 default:  
 cout << endl;  
 }  
 cout << "\nВыбрать другую сортировку? да - 1; нет - 0" << endl;  
 cin >> cycleone;  
 }  
 }  
 case 3: {  
 // Поиск минимального и максимального в неотсротированом массиве  
 int arrMaxMin[N] = {};  
 for (int i = 0; i < N; i++)  
 arrMaxMin[i] = arr[i];  
 auto startFind = steady\_clock::now();  
 for (int i = 0; i < N - 1; i++)  
 if (arrMaxMin[i] > arrMaxMin[i + 1])  
 swap(arrMaxMin[i], arrMaxMin[i + 1]);  
 for (int i = N - 1; i > 0; i--)  
 if (arrMaxMin[i] < arrMaxMin[i - 1])  
 swap(arrMaxMin[i], arrMaxMin[i - 1]);  
 auto endFind = steady\_clock::now();  
 auto resultFind = duration\_cast<nanoseconds>(endFind - startFind);  
 // 3. Поиск максмимального и минимального элемента.  
 cout << "Поиск максмимального и минимального элемента. " << "Введите sorted или unsorted:" << endl;  
 string array;  
 int j = 0, shaker[N] = {};  
 while (j != 2) {  
 cin >> array;  
 if (array == "unsorted") {  
 cout << "Min и max элемент массива: " << arrMaxMin[0] << ", " << arrMaxMin[N - 1] << endl;  
 j += 1;  
 cout << "Поиск элементов занял: " << resultFind.count() << " наносекунд\n";  
 } else if (array == "sorted") {  
 for (int i = 0; i < N; i++) {  
 shaker[i] = arr[i];  
 }  
 shakersort(arr, N);  
 auto startFindS = steady\_clock::now();  
 auto endFindS = steady\_clock::now();  
 auto resultFindS = duration\_cast<nanoseconds>(endFindS - startFindS);  
 cout << "Min и max элемент массива: " << arr[0] << ", " << arr[N - 1] << endl;  
 cout << "Поиск элементов занял: " << resultFindS.count() << " наносекунд\n\n";  
 j += 1;  
 }  
 }  
 }  
 case 4: {  
 // 4. Среднее значение минимального и максимального элемента. Индексы равные этому значению и их кол-во.  
 int avg = (arr[N - 1] + arr[0]) / 2, k = 0;  
 cout << endl << "Среднее значение: " << avg << endl;  
 for (int i = 0; i < N; i++)  
 if (arr[i] == avg) {  
 k += 1;  
 cout << "arr[" << i << "] = ";  
 }  
 if (k != 0) cout << avg << " ";  
 cout << "Количество равных значений: " << k << endl;  
 }  
 case 5: {  
 shakersort(arr, N);  
 int a, cnt(0);  
 cout << "\nВведите число а:\n";  
 cin >> a;  
 for (int i = 0; i < N; i++) {  
 if (arr[i] < a) {  
 cnt++;  
 } else  
 break;  
 }  
 if (cnt > 0)  
 cout << "Количество элементов, которые меньше числа а: " << cnt << "\n";  
 else  
 cout << "\nНет таких элементов";  
 }  
 case 6: {  
 shakersort(arr, N);  
 int b, cnt(0);  
 cout << "\nВведите число b:\n";  
 cin >> b;  
 for (int i = N - 1; i >= 0; i--) {  
 if (arr[i] > b) {  
 cnt++;  
 } else  
 break;  
 }  
 if (cnt > 0)  
 cout << "Количество элементов, которые больше числа а: " << cnt << "\n";  
 else  
 cout << "\nНет таких элементов";  
 }  
 case 7: {  
 cout << "\nБинарный поиск: 1 - запустить цикл; 0 - закончить цикл" << endl;  
 int cycleseven;  
 cin >> cycleseven;  
 while (cycleseven) {  
 cout << "\nВведите число в диапазоне [-99,99]\n";  
 int x;  
 cin >> x;  
 {  
 auto start = steady\_clock::now();  
 int result = binarysearch(arr, x, 0, N - 1);  
 auto end1 = steady\_clock::now();  
 auto elapsed = duration\_cast<nanoseconds>(end1 - start);  
 if (result == -1)  
 cout << "Число не найдено";  
 else  
 cout << "Элемент найден по индексу: " << result;  
 cout << endl;  
 cout << "Бинарный поиск занял " << elapsed.count() << " наносекунд\n";  
 }  
 cout << endl;  
 //------------------------------------------------------------------------------------------------------  
 auto start = steady\_clock::now();  
 for (int i = 0, k = 0; i < N; i++) {  
 if (arr[i] == x) cout << "Введенное число находится в массиве с номером " << i << endl;  
 k += 1;  
 if (k == 0 && i == 99) cout << "Введенное число не входит в массив\n";  
 }  
 auto endTime = steady\_clock::now();  
 auto elapsed = duration\_cast<nanoseconds>(endTime - start);  
 cout << "Перечисление элементов заняло " << elapsed.count() << " наносекунд\n";  
 cout << "\nПродолжить цикл? 1 - да; 2 - нет\n";  
 cin >> cycleseven;  
 }  
 }  
 case 8: {  
 int a, b;  
 cout << "\nВведите индексы элементов, которые вы хотите поменять: \n";  
 cin >> a >> b;  
 swap(arr[a], arr[b]);  
 for (int i = 0; i < N; i++)  
 cout << arr[i] << " ";  
 cout << endl;  
 }  
 default:  
 cout << endl;  
 }  
}