**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: Одномерные статистические массивы

|  |  |  |
| --- | --- | --- |
| Студент гр. 2372 |  | Михеева Э. Д. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение понятия «массив», правил оформления алгоритмов и программ с массивами; формирование навыков решения задач с применением одномерных массивов; закрепление навыков по составлению циклических программ и алгоритмов.

**Основные теоретические положения.**

Статические одномерные массивы — это комплект данных одного типа, располагающихся в непрерывном участке памяти так, что по индексу компонента определяется адрес его значения.

Массивы являются широко используемым способом структурного отображения информационных данных, позволяющим в удобной форме выполнять размещение и обработку в компьютерном оборудовании большие объёмы информации. Под массивом понимается набор данных, обладающих одинаковым типом и имеющих единое общее наименование, которое является именем массива, а также отличающихся местоположением в составе массива.

Массивы подразделяются на статические и динамические. Границы статических массивов определяются при их формировании и изменить их можно лишь в другом варианте программы. Статический массив может быть определён только глобально, и его невозможно задать локально внутри процедуры. В языке программирования С++ массив не является стандартизированным видом информационных данных, и, кроме того, он может обладать различными типами, такими как ìnt, char, float, double и другие. В составе массива хранятся определённые значения, именуемые элементами или компонентами массива.

Все элементы, входящие в состав массива, должны иметь одинаковый тип. Каждый элемент массива обладает собственным индексом, определяющим его расположение в массиве. Размерностью массива является количество измерений массива, то есть число индексов. Объём, включающий все измерения массива, вместе с типом его элементов определяет количество байт, необходимых, чтобы сохранить массив в памяти. Данное число байт вычисляется в процессе работы программы компиляции.

Массивы, у которых присутствует больше одного индекса, являются многомерными. У одномерного массива имеется набор однотипных элементов, обратиться к которым можно при помощи одного индекса. Весь набор элементов массива расположен в памяти в порядке очерёдности, причём первый элемент обладает индексом с номером нуль, а все следующие элементы обладают индексами, увеличивающимися на единицу. То есть индексами является порядковые номера элементов, находящихся в массиве. Чтобы можно было применять массив далее в программном приложении, он, подобно обычным переменным, должен быть вначале объявлен.

Операция инициализации статического одномерного массива состоит в задании начальных значений его элементов, которое осуществляется при объявлении массива. Данная процедура реализуется указанием в фигурных скобках начальных (исходных) значений. Если осуществляется инициализация статического одномерного массива, но при этом назначается меньшее число начальных данных, чем установленный объём массива, то остальные элементы получают нулевые значения.

**Постановка задачи.**

Разработать алгоритм и написать программу, которая позволяет:

1. Создать целочисленный массив размерности N = 100, элементы которого принимают случайное значение в диапазоне от -99 до 99.
2. Отсортировать массив различными сортировками (от меньшего к большему). Определить время, затраченное на сортировку.
3. Найти максимальный и минимальный элемент массива, подсчитать время поиска этих элементов в отсортированном массиве и неотсортированном.
4. Выводить среднее значение максимального и минимального значений в отсортированном и неотсортированном массиве, выводить индексы всех элементов, которые равны этому значению, и их количество. Подсчитать время поиска.
5. Выводить количество элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.
6. Выводить количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.
7. Выводить информацию о том, есть ли введенное пользователем число в отсортированном массиве.
8. Менять местами элементы массива, индексы которых вводит пользователь. Вывести скорость обмена.

**Выполнение работы.**

Код программы представлен в приложении А.

1. При запуске программы пользователю выводится время выполнения программы и меню доступных команд, ожидается ввод команды с клавиатуры (рис. 1).

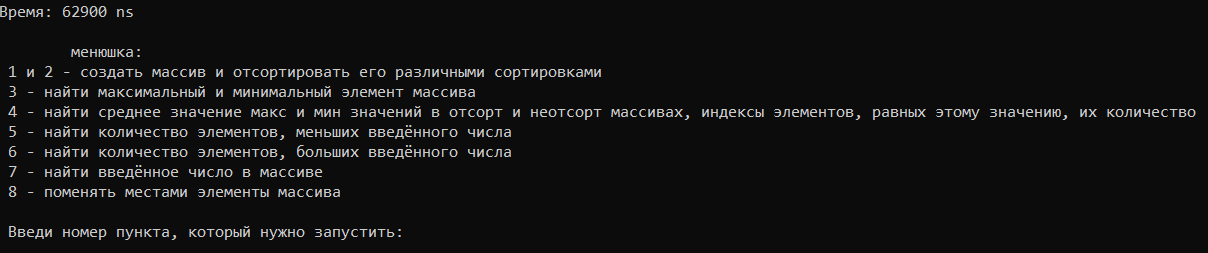


Рисунок . Запуск программы

1. Следующий шаг зависит от введенной команды. Если пользователь ввёл:
   1. “1” или “2”, то выводится сгенерированный массив со случайными значениями от -99 до 99 и различные виды сортировок, а также с использованием библиотеки chrono посчитано время, затраченное на каждую сортировку (рис. 2). Также сразу после выполнения каждого пункта выводится вопрос «Продолжить выполнение?» и предлагается два варианта «1 – да, 0 – нет». При вводе пользователем «1» программа продолжит выполнение и выведет фразу «Введи номер пункта, который нужно запустить», после чего появляется возможность вновь ввести номер пункта (рис. 3). При вводе пользователем «0» программа завершает выполнение.

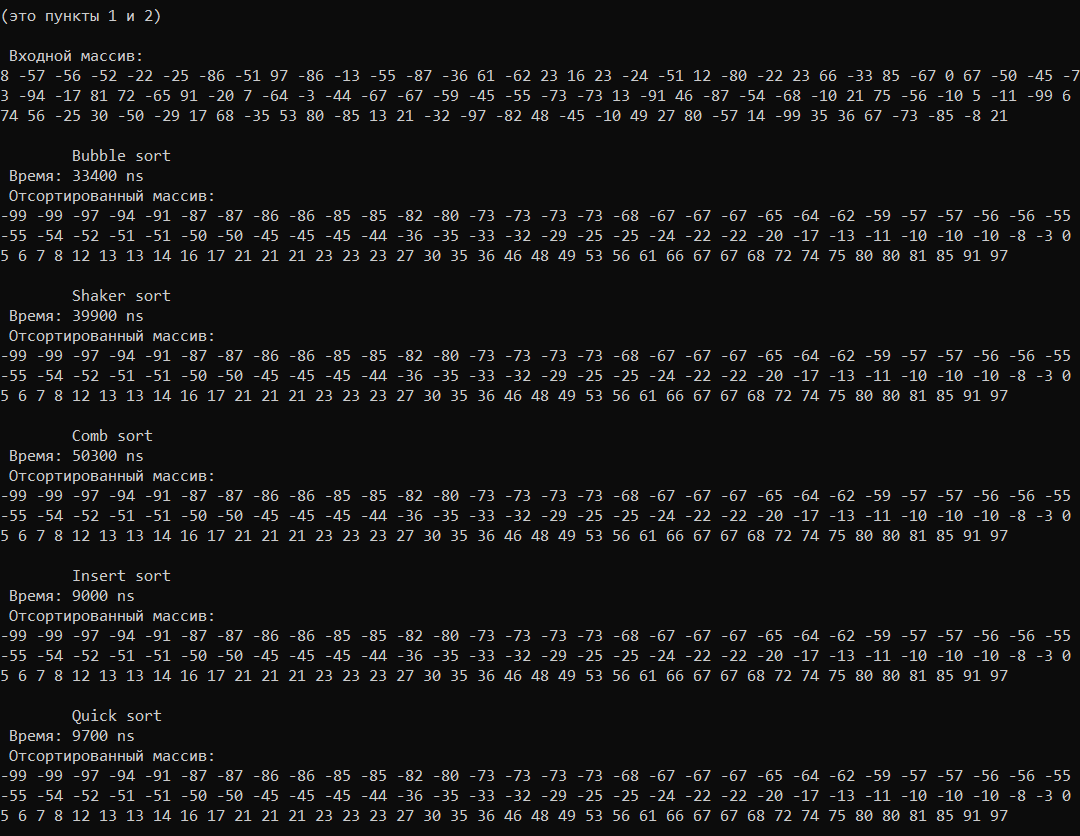


Рисунок 2. Задача 1 и 2.

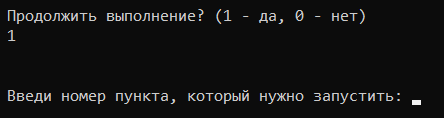


Рисунок 3.

* 1. “3” – выводятся максимальный и минимальный элементы массива, а также время, затраченное на поиск этих элементов в отсортированном и неотсортированном массивах (рис. 4). Вновь выводится вопрос о продолжении работы программы.

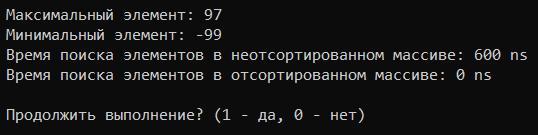


Рисунок 4. Задача 3.

* 1. “4” – выводится среднее значение между минимальным и максимальным элементами массива; индексы элементов, равных этому среднему значению; количество таких элементов; затраченное время на выполнение программы для отсортированного и неотсортированного массива (рис. 5).

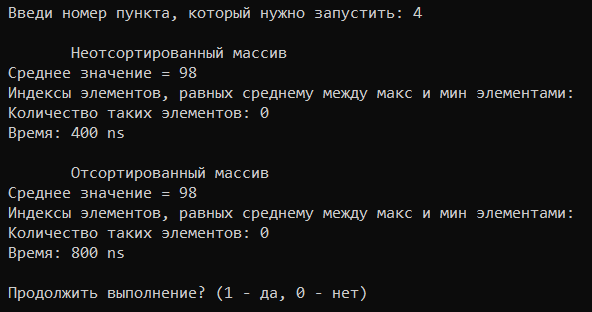
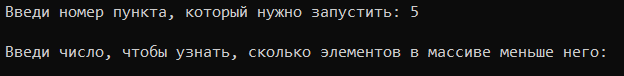


Рисунок 5. Задача 4.

* 1. “5” – ожидается ввод пользователем числа (рис. 6). При вводе пользователем числа выводится количество элементов, которые меньше введённого числа, и время, затраченное на выполнение этой задачи (рис. 7).

Рисунок 6. Задача 5, ожидание ввода числа.

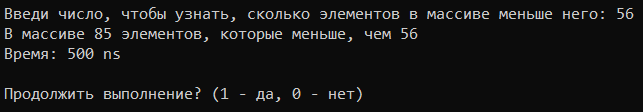


Рисунок 7. Задача 5.

* 1. “6” – ожидается ввод пользователем числа (рис. 8). При вводе пользователем числа выводится количество элементов, которые больше введённого числа, и время, затраченное на выполнение этой задачи (рис. 9).

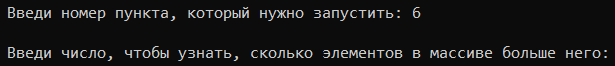


Рисунок 8. Задача 6, ожидание ввода числа.

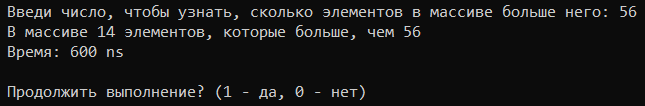


Рисунок 9. Задача 6.

* 1. “7” – ожидается ввод числа, которое нужно найти в массиве (рис. 10). После ввода пользователем числа выводится индекс введённого числа в массиве, если такое число в массиве имеется (рис. 11); если введённое число в массиве не найдено, пользователю сообщается о том, что такого числа в массиве нет; в обоих случаях выводится затраченное время на поиск числа при обычном переборе (линейном поиске) и с применением бинарного поиска (рис. 12).



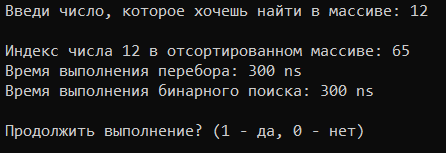
Рисунок 10. Задача 7, ожидание ввода числа.

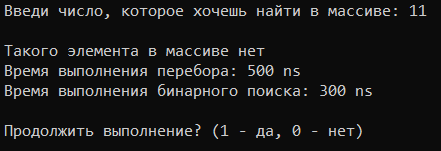
Рисунок 11. Задача 7, число найдено в массиве.

Рисунок 12. Задача 7, число не найдено в массиве.

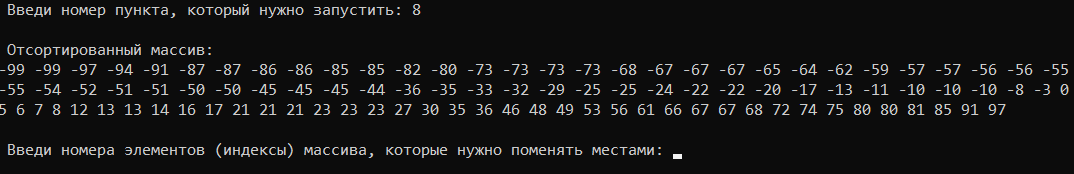
* 1. “8” – выводится отсортированный массив и ожидается ввод пользователем двух чисел – индексов элементов, которые нужно поменять местами (рис. 13). После ввода пользователем двух чисел выводится изменённый массив, в котором введённые пользователем элементы поменялись местами (рис. 14).

Рисунок 13. Задание 8, ожидание ввода индексов элементов.

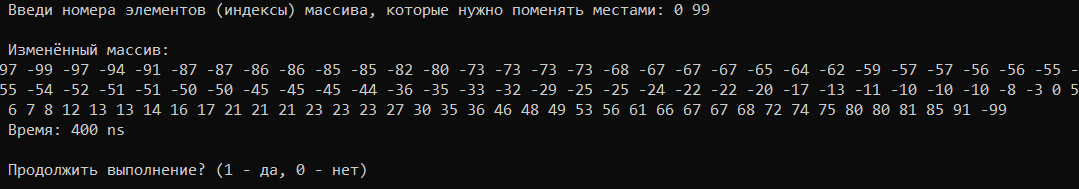


Рисунок 14. Задание 8, изменённый массив.

**Выводы.**

В ходе работы были изучены одномерные статистические массивы и действия, которые можно совершать над ними; различные виды сортировок массивов.

Приложение А

рабочий код

#include <iostream>  
#include <random>  
#include <ctime>  
#include <chrono>  
#include <cstdlib>  
#include <Windows.h>  
using namespace std;  
using namespace chrono;  
  
#define N 100  
  
int\* generate\_array(int array[], int sortedArray[]) {  
 for (int i = 0; i < N; ++i) {  
 array[i] = rand() % 199 - 99;  
 sortedArray[i] = array[i];  
 }  
}  
  
int\* print\_array(int array[]) {  
 for (int i = 0; i < N; i++) {  
 cout << array[i] << " ";  
 }  
}  
  
int\* print\_sortedArray(int array[], int sortedArray[]) {  
 for (int i = 0; i < N; i++) {  
 cout << sortedArray[i] << " ";  
 sortedArray[i] = array[i];  
 }  
}  
  
int\* print\_sortedArray1(int sortedArray[]) {  
 for (int i = 0; i < N; i++) {  
 cout << sortedArray[i] << " ";  
 }  
}  
  
void bubble\_sort(int \*sortedArray) {  
 for (int i = 0; i < N - 1; i++) {  
 for (int j = 0; j < N - i - 1; j++) {  
 if (sortedArray[j] > sortedArray[j + 1])  
 swap(sortedArray[j], sortedArray[j + 1]);  
 }  
 }  
}  
  
void shaker\_sort(int \*sortedArray){  
 for (int i = 0; i < N; i++){  
 for (int j = i; j < N - 1 - i; j++){  
 if (sortedArray[j] > sortedArray[j+1])  
 swap(sortedArray[j], sortedArray[j+1]);  
 }  
 for (int j = N-1; j > i; j--){  
 if (sortedArray[j] < sortedArray[j-1])  
 swap(sortedArray[j], sortedArray[j-1]);  
 }  
 }  
}  
  
void comb\_sort(int \*sortedArray){  
 const float k = 1.247;  
 float S = N;  
 while (k < S){  
 for (int i = 0; i + k < S; i++){  
 if (sortedArray[i] > sortedArray[int(i + k)])  
 swap(sortedArray[i], sortedArray[int(i + k)]);  
  
 }  
 S /= k;  
 }  
 int count = 0;  
 while (true){  
 for (int i = 0; i < N - 1; i++){  
 if (sortedArray[i] > sortedArray[i + 1])  
 swap(sortedArray[i], sortedArray[i+1]);  
 else count++;  
 }  
 if (count == N - 1)  
 break;  
 else  
 count = 0;  
 }  
}  
  
void insert\_sort(int \*sortedArray){  
 int x, j;  
 for (int i = 1; i < N; i++){  
 x = sortedArray[i];  
 j = i;  
 while ((j > 0) && (x < sortedArray[j - 1])){  
 sortedArray[j] = sortedArray[j - 1];  
 j--;  
 }  
 sortedArray[j] = x;  
 }  
}  
  
void quicksort(int\* sortedArray, int endd, int begin)  
{  
 int mid;  
 int f = begin;  
 int l = endd;  
 mid = sortedArray[(f + l) / 2];  
 while (f < l)  
 {  
 while (sortedArray[f] < mid) f++;  
 while (sortedArray[l] > mid) l--;  
 if (f <= l)  
 {  
 swap(sortedArray[f], sortedArray[l]);  
 f++;  
 l--;  
 }  
 }  
 if (begin < l)  
 quicksort(sortedArray, l, begin);  
 if (f < endd)  
 quicksort(sortedArray, endd, f);  
  
}  
  
int linear\_search(int sortedArray[], int x) {  
 for (int i = 0; i < N; ++i) {  
 if (sortedArray[i] == x)  
 return i;  
 }  
 return -1;  
}  
  
int binary\_search(int sortedArray[], int l, int r, int x) {  
 int mid;  
 while (l <= r) {  
 mid = (l + r) / 2;  
 if (sortedArray[mid] == x)  
 return mid;  
 if (sortedArray[mid] > x)  
 return binary\_search(sortedArray, l, mid - 1, x);  
 if (sortedArray[mid] < x)  
 return binary\_search(sortedArray, mid + 1, r, x);  
  
 }  
 return -1;  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 srand(time(0));  
 auto start = steady\_clock::now();  
 cout << endl;  
 auto end = steady\_clock::now();  
 auto result = duration\_cast<nanoseconds>(end - start);  
 cout << "Время: " << result.count() << " ns";  
 cout << "\n\n\tменюшка:" << "\n 1 и 2 - создать массив и отсортировать его различными сортировками" <<  
 "\n 3 - найти максимальный и минимальный элемент массива\n 4 - найти среднее значение макс и мин значений в отсорт и неотсорт массивах, индексы элементов, равных этому значению, их количество" <<  
 "\n 5 - найти количество элементов, меньших введённого числа\n 6 - найти количество элементов, больших введённого числа" <<  
 "\n 7 - найти введённое число в массиве\n 8 - поменять местами элементы массива";  
  
 int p, YorN = 1;  
  
 int array[N], sortedArray[N];  
 generate\_array(array, sortedArray);  
  
 while (YorN == 1) {  
 cout << "\n\n Введи номер пункта, который нужно запустить: " ;  
 cin >> p;  
 switch (p) {  
 case 1:  
 case 2: {  
 cout << "\n\n(это пункты 1 и 2)";  
 cout << "\n\n Входной массив:\n";  
 print\_array(array);  
  
 cout << "\n\n\tBubble sort" << endl;  
 start = steady\_clock::now();  
 bubble\_sort(sortedArray);  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << " Время: " << result.count() << " ns";  
 cout << "\n Отсортированный массив:\n";  
 print\_sortedArray(array, sortedArray);  
  
 cout << "\n\n\tShaker sort" << endl;  
 start = steady\_clock::now();  
 shaker\_sort(sortedArray);  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << " Время: " << result.count() << " ns";  
 cout << "\n Отсортированный массив:\n";  
 print\_sortedArray(array, sortedArray);  
  
 cout << "\n\n\tComb sort" << endl;  
 start = steady\_clock::now();  
 comb\_sort(sortedArray);  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << " Время: " << result.count() << " ns";  
 cout << "\n Отсортированный массив:\n";  
 print\_sortedArray(array, sortedArray);  
  
 cout << "\n\n\tInsert sort" << endl;  
 start = steady\_clock::now();  
 insert\_sort(sortedArray);  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << " Время: " << result.count() << " ns";  
 cout << "\n Отсортированный массив:\n";  
 print\_sortedArray(array, sortedArray);  
  
 cout << "\n\n\tQuick sort" << endl;  
 start = steady\_clock::now();  
 quicksort(sortedArray, N - 1, 0);  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << " Время: " << result.count() << " ns";  
 cout << "\n Отсортированный массив:\n";  
 print\_sortedArray1(sortedArray);  
 }break;  
  
 case 3: {  
 int max1 = array[0], min1 = array[0];  
 start = steady\_clock::now();  
 for (int i = 0; i < N; ++i) {  
 max1 = array[i] > max1 ? array[i] : max1;  
 min1 = array[i] < min1 ? array[i] : min1;  
 }  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << endl;  
 cout << "\n Максимальный элемент: " << max1 << "\n Минимальный элемент: " << min1;  
 cout << "\n Время поиска элементов в неотсортированном массиве: " << result.count() << " ns";  
  
 insert\_sort(sortedArray);  
 start = steady\_clock::now();  
 int max2 = sortedArray[0];  
 int min2 = sortedArray[N - 1];  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\n Время поиска элементов в отсортированном массиве: " << result.count() << " ns";  
 }break;  
  
 case 4: {  
 insert\_sort(sortedArray);  
 int max1 = array[0], min1 = array[0];  
 int cou = 0, sredn;  
 for (int i = 0; i < N; ++i) {  
 max1 = array[i] > max1 ? array[i] : max1;  
 min1 = array[i] < min1 ? array[i] : min1;  
 }  
 sredn = (max1 - min1) / 2;  
 cout << "\n\tНеотсортированный массив\n" << " Среднее значение = " << sredn << "\n" << " Индексы элементов, равных среднему между макс и мин элементами: ";  
 start = steady\_clock::now();  
 for (int i = 0; i < N; ++i) {  
 if (array[i] == sredn) {  
 cou++;  
 cout << i << " ";  
 }  
 }  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\n Количество таких элементов: " << cou;  
 cout << "\n Время: " << result.count() << " ns";  
  
 cou = 0;  
 sredn = (sortedArray[N-1] - sortedArray[0]) / 2;  
 cout << "\n\n\tOтсортированный массив\n" << " Среднее значение = " << sredn << "\n" << " Индексы элементов, равных среднему между макс и мин элементами: ";  
 start = steady\_clock::now();  
 for (int i = 0; i < N; ++i) {  
 if (sortedArray[i] == sredn) {  
 cou++;  
 cout << i << " ";  
 }  
 }  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\n Количество таких элементов: " << cou;  
 cout << "\n Время: " << result.count() << " ns";  
 }break;  
  
 case 5: {  
 insert\_sort(sortedArray);  
 int a, cou = 0;  
 cout << "\n Введи число, чтобы узнать, сколько элементов в массиве меньше него: ";  
 cin >> a;  
 start = steady\_clock::now();  
 for (int i = 0; i < N; i++) {  
 if (sortedArray[i] < a)  
 cou++;  
 else  
 break;  
 }  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << " В массиве " << cou << " элементов, которые меньше, чем " << a;  
 cout << "\n Время: " << result.count() << " ns";  
 } break;  
  
 case 6: {  
 insert\_sort(sortedArray);  
 int b, cou = 0;  
 cout << "\n Введи число, чтобы узнать, сколько элементов в массиве больше него: ";  
 cin >> b;  
 start = steady\_clock::now();  
 for (int i = N; i > 0; i--) {  
 if (sortedArray[i] > b)  
 cou++;  
 //else  
 // break;  
 }  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << " В массиве " << cou << " элементов, которые больше, чем " << b;  
 cout << "\n Время: " << result.count() << " ns";  
 } break;  
  
 case 7: {  
 insert\_sort(sortedArray);  
 int x, ind;  
 cout << " Введи число, которое хочешь найти в массиве: ";  
 cin >> x;  
 start = steady\_clock::now();  
 ind = linear\_search(sortedArray, x);  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 if (ind == -1)  
 cout << "\n Такого элемента в массиве нет";  
 if (ind != -1)  
 cout << "\n Индекс числа " << x << " в отсортированном массиве: " << ind;  
 cout << "\n Время выполнения перебора: " << result.count() << " ns";  
  
 start = steady\_clock::now();  
 binary\_search(sortedArray, 0, N - 1, x);  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\n Время выполнения бинарного поиска: " << result.count() << " ns";  
 } break;  
  
 case 8: {  
 insert\_sort(sortedArray);  
 cout << "\n Отсортированный массив:" << endl;  
 print\_sortedArray1(sortedArray);  
 int elem1, elem2;  
 cout << "\n\n Введи номера элементов (индексы) массива, которые нужно поменять местами: ";  
 cin >> elem1 >> elem2;  
 start = steady\_clock::now();  
 swap(sortedArray[elem1], sortedArray[elem2]);  
 end = steady\_clock::now();  
 result = duration\_cast<nanoseconds>(end - start);  
 cout << "\n Изменённый массив:" << endl;  
 print\_sortedArray1(sortedArray);  
 cout << "\n Время: " << result.count() << " ns";  
 } break;  
 }  
 cout << "\n\n Продолжить выполнение? (1 - да, 0 - нет)" << endl;  
 cin >> YorN;  
 }  
 return 0;  
}