**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 СТАТИЧЕСКИЕ МАССИВЫ**"**

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

Санкт-Петербург

2021

**Цель работы.**

Научиться работать с массивами, сортировками массивов.

**Основные теоретические положения.**

**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.

**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.

**Comb sort** (сортировка расческой) – ещё одна модификация сортировки пузырьком. 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.

**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.

**Быстрая сортировка (quick sort)** – одна из самых быстрых сортировок. Эта сортировка по сути является существенно улучшенной версией алгоритма пузырьковой сортировки.

Общая идея алгоритма состоит в том, что сначала выбирается из массива элемент, который называется опорным. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Затем необходимо сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: меньше опорного, раны опорному и больше опорного. Для меньших и больших значений необходимо выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

**Постановка задачи.**

Необходимо реализовать 5 видов сортировок в виде функций, создать массив на 100 элементов, заполнить его случайными числами и выполнить манипуляции с массивом по заданиям.

|  |  |
| --- | --- |
| * Номер варианта | * Название сортировки |
| * 1 | * Bubble sort (пузырьковая сортировка) |
| * 2 | * Shaker sort (шейкер-сортировка) |
| * 3 | * Comb sort (сортировка расчёской) |
| * 4 | * Insert sort (сортировка вставками) |
| * 5 (\*) | * Quick sort (быстрая сортировка) |

**Выполнение работы.**

Код программы представлен в приложении А.

1. Программа создает массив на 100 элементов и заполняет его случайными числами в заданном диапазоне.
2. Пользователю предлагается на выбор 1 из 5 сортировок, после выбора пользователя происходит сортировка массива и его вывод на экран. Так же пользователь увидит сколько времени было затрачено на сортировку конкретным способом.
3. Далее программа выполняет все остальные задания практической работы (запрашивает нужные для задания данные у пользователя, обрабатывает их и выводит на экран)

**Выводы.**

**Код написан, и работает корректно.**

Приложение А

рабочий код

|  |
| --- |
| #include <iostream> |
|  | #include <ctime> |
|  | #include <chrono> |
|  | #include <math.h> |
|  | #include <Windows.h> |
|  | #include <algorithm> |
|  | HANDLE hConsole; |
|  | using namespace std; |
|  |  |
|  | void arr\_out(int arr[]){ //функция вывода массива |
|  | for(int i = 0; i < 100; i++){ |
|  | cout << arr[i] << ' '; |
|  | } |
|  | cout << endl << endl; |
|  | } |
|  |  |
|  | void arr\_sorting\_bubble(int arr[], int n){ |
|  | bool bub = true; |
|  | while(bub){ |
|  | bub = false; |
|  | for (int i = 0; i < n ; i++){ |
|  | if (arr[i] > arr[i + 1]) { |
|  | swap(arr[i],arr[i + 1]); |
|  | bub = true; |
|  | } |
|  | } |
|  | n--; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | void arr\_sorting\_shaker(int arr[],int end, int begin){ |
|  | bool shaker = true; |
|  | while(shaker){ |
|  | shaker = false; |
|  | for (int i = begin; i < end; i++){ |
|  | if (arr[i] > arr[i + 1]){ |
|  | swap(arr[i],arr[i + 1]); |
|  | shaker = true; |
|  | } |
|  | } |
|  | end--; |
|  | for(int j = end ; j > begin; j--){ |
|  | if (arr[j] < arr[j - 1]){ |
|  | swap(arr[j], arr[j - 1]); |
|  | shaker = true; |
|  | } |
|  | } |
|  | begin++; |
|  | } |
|  | } |
|  |  |
|  | void arr\_sorting\_comb (int arr[], int end){ |
|  | int s = end; |
|  | while (s >= 1){ |
|  | for (int i = 0; i + s <= end; i++){ |
|  | if (arr[i] > arr[i + s]) swap(arr[i], arr[i + s]); |
|  | } |
|  | s -= 1; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | void arr\_sorting\_insert(int arr[], int n){ |
|  | for (int i = 1; i <= n; i++){ |
|  | for (int j = i; j > 0; j--){ |
|  | if (arr[j] < arr[j - 1]) swap(arr[j], arr[j - 1]); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void arr\_sorting\_quick(int arr[],int end, int begin){ // функция сортировки массива |
|  | int l = begin, r = end, mid = arr[(r + l) / 2]; |
|  | while (l < r){ |
|  | for(;arr[l] < mid; l++); |
|  | for(;arr[r] > mid; r--); |
|  | if (l <= r){ |
|  | swap(arr[l], arr[r]); |
|  | l++; |
|  | r--; |
|  | } |
|  | } |
|  | if (begin < r) arr\_sorting\_quick(arr, r, begin); |
|  | if (l < end) arr\_sorting\_quick(arr, end, l); |
|  | } |
|  |  |
|  | void max\_min\_value(int arr[]){ // функция для поиска максимального и минимального значения в массиве |
|  | int max\_value = -1000, min\_value = 1000; |
|  | auto start11 = chrono::high\_resolution\_clock::now(); |
|  | for (int i = 0; i < 100; i++){ |
|  | if (arr[i] > max\_value){ |
|  | max\_value = arr[i]; |
|  | } |
|  | if (arr[i] < min\_value){ |
|  | min\_value = arr[i]; |
|  | } |
|  | } |
|  | auto finish11 = chrono::high\_resolution\_clock::now(); |
|  | chrono :: duration <float> duration1 = finish11 - start11; |
|  | cout.precision(8); |
|  | cout << "Minimum value in the array = "<< min\_value << '\n' << "Maximum value in the array = " << max\_value <<'\n'; |
|  | cout << "Time to search for items in array = " << duration1.count() << " s"<< '\n'; |
|  | } |
|  |  |
|  | void mean(int arr[], int a, int b){ // функция для поиска среднего значения массива |
|  | int mean\_value = round((a + b)/2), count = 0; |
|  | SetConsoleTextAttribute(hConsole, 11); |
|  | cout << '\n' << "Mean = " << mean\_value <<'\n' << "Indexes of elements equal to the mean :" << '\n'; |
|  | for(int i = 0; i < 100; i++){ |
|  | if(arr[i] == mean\_value){ |
|  | cout << i <<" "; |
|  | count ++; |
|  | } |
|  | } |
|  | cout << '\n' << "Amount such of elements : " << count <<'\n'; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | } |
|  |  |
|  | void arr\_exchange(int arr[], int a, int b ){ // функция для поиска времени перестановок |
|  | arr\_out(arr); |
|  | auto start\_4 = chrono::high\_resolution\_clock::now(); |
|  | swap(arr[a],arr[b]); |
|  | auto finish\_4 = chrono::high\_resolution\_clock::now(); |
|  | chrono :: duration <float> duration\_4 = finish\_4 - start\_4; |
|  | arr\_out(arr); |
|  | cout << "Time to exchange = " << duration\_4.count() << " s" <<'\n'; |
|  | } |
|  |  |
|  | #pragma clang diagnostic push |
|  | #pragma ide diagnostic ignored "cert-msc51-cpp" |
|  | int main() { |
|  | hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE); |
|  | cout.setf(ios\_base::fixed); |
|  | srand(time(nullptr)); |
|  | int arr[100],arr\_1[100], first\_value, last\_value, key\_sort; |
|  | for(int &i : arr){ |
|  | i = -99 + rand() % 199; // заполнение массива рандомными числами |
|  | } |
|  | copy\_n(arr, 100, arr\_1); // создание копии массива |
|  |  |
|  | SetConsoleTextAttribute(hConsole, 12); |
|  | cout <<'\t' <<"Unsorted array :" << '\n'; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  |  |
|  | arr\_out(arr); |
|  |  |
|  | max\_min\_value(arr\_1); |
|  |  |
|  | SetConsoleTextAttribute(hConsole, 14); |
|  | cout << "Choose sorting type :"<< '\n'; |
|  | SetConsoleTextAttribute(hConsole, 10); |
|  | repeat: |
|  | cout <<"1) Bubble sort" <<'\n'<< "2) Shaker sort" <<'\n'<<"3) Comb sort" <<'\n'<<"4) Insert sort" <<'\n'<<"5) Quick sort" <<'\n'<< "enter sort number"<<'\n'; |
|  | cin >> key\_sort; |
|  | switch (key\_sort) { |
|  | case 1: |
|  | { |
|  | auto start\_1\_1 = chrono::high\_resolution\_clock::now(); |
|  | arr\_sorting\_bubble(arr, 99); |
|  | auto finish\_1\_1 = chrono::high\_resolution\_clock::now(); |
|  | chrono :: duration <float> duration\_1\_1 = finish\_1\_1 - start\_1\_1; |
|  | SetConsoleTextAttribute(hConsole, 13); |
|  | cout << "Time spent sorting : " << duration\_1\_1.count() << " s" <<endl; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | break; |
|  | } |
|  | case 2: |
|  | { |
|  | auto start\_1\_2 = chrono::high\_resolution\_clock::now(); |
|  | arr\_sorting\_shaker(arr, 99, 0); |
|  | auto finish\_1\_2 = chrono::high\_resolution\_clock::now(); |
|  | chrono :: duration <float> duration\_1\_2 = finish\_1\_2 - start\_1\_2; |
|  | SetConsoleTextAttribute(hConsole, 13); |
|  | cout << "Time spent sorting : " << duration\_1\_2.count() << " s" <<endl; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | break; |
|  | } |
|  | case 3: |
|  | { |
|  | auto start\_1\_3 = chrono::high\_resolution\_clock::now(); |
|  | arr\_sorting\_comb(arr, 99); |
|  | auto finish\_1\_3 = chrono::high\_resolution\_clock::now(); |
|  | chrono :: duration <float> duration\_1\_3 = finish\_1\_3 - start\_1\_3; |
|  | SetConsoleTextAttribute(hConsole, 13); |
|  | cout << "Time spent sorting : " << duration\_1\_3.count() << " s" <<endl; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | break; |
|  | } |
|  | case 4: |
|  | { |
|  | auto start\_1\_4 = chrono::high\_resolution\_clock::now(); |
|  | arr\_sorting\_insert(arr, 99); |
|  | auto finish\_1\_4 = chrono::high\_resolution\_clock::now(); |
|  | chrono :: duration <float> duration\_1\_4 = finish\_1\_4 - start\_1\_4; |
|  | SetConsoleTextAttribute(hConsole, 13); |
|  | cout << "Time spent sorting : " << duration\_1\_4.count() << " s" <<endl; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | break; |
|  | } |
|  | case 5: |
|  | { |
|  | auto start\_1\_5 = chrono::high\_resolution\_clock::now(); |
|  | arr\_sorting\_quick(arr, 99,0); |
|  | auto finish\_1\_5 = chrono::high\_resolution\_clock::now(); |
|  | chrono :: duration <float> duration\_1\_5 = finish\_1\_5 - start\_1\_5; |
|  | SetConsoleTextAttribute(hConsole, 13); |
|  | cout << "Time spent sorting : " << duration\_1\_5.count() << " s" <<endl; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | break; |
|  | } |
|  | default : |
|  | goto repeat; |
|  | } |
|  |  |
|  | SetConsoleTextAttribute(hConsole, 10); |
|  | cout <<'\t' <<"Sorted array :" << '\n'; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  |  |
|  | arr\_out(arr); |
|  | auto start\_0 = chrono :: high\_resolution\_clock :: now(); |
|  | first\_value = arr[0]; |
|  | last\_value = arr[99]; |
|  | auto finish\_0 = chrono :: high\_resolution\_clock :: now(); |
|  | chrono :: duration <float> duration\_0 = finish\_0 - start\_0; |
|  | cout << "Minimum value in the array = "<< first\_value << '\n' << "Maximum value in the array = " << last\_value <<'\n'; |
|  | cout << "Time to search for items in array = " << duration\_0.count() << " s"<< '\n'; |
|  |  |
|  | mean(arr\_1, first\_value, last\_value); |
|  |  |
|  | int a, b, count = 0; |
|  | bool b1 = true; |
|  | while(b1){ |
|  | SetConsoleTextAttribute(hConsole, 14); |
|  | cout << '\t' <<"Enter the number a (task 5)"<< '\n'; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | cin >> a; |
|  | if (cin){ |
|  | for(int i = 0; i < 100; i++){ |
|  | if (arr[i] < a){ |
|  | count++; |
|  | } |
|  | } |
|  | cout << "Amount of elements which are less " << a << " = " << count <<'\n'; |
|  | count = 0; |
|  | } |
|  | else b1 = false; |
|  | } |
|  | count = 0; |
|  | cin.clear(); |
|  | while(cin.get() != '\n'); |
|  | b1 = true; |
|  | while(b1){ |
|  | SetConsoleTextAttribute(hConsole, 14); |
|  | cout << '\t' <<"Enter the number b (task 6)"<< '\n'; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | cin >> b; |
|  | if (cin){ |
|  | for(int i = 0; i < 100; i++){ |
|  | if (arr[i] > b){ |
|  | count++; |
|  | } |
|  | } |
|  | cout << "Amount of elements which are less " << b << " = " << count <<'\n'; |
|  | count = 0; |
|  | } |
|  | else b1 = false; |
|  | } |
|  | cin.clear(); |
|  | while(cin.get() != '\n'); |
|  | b1 = true; |
|  | while(b1){ |
|  | int a\_1, l = 0, r = 99, mid = 0; |
|  | bool b\_1 = false; |
|  | SetConsoleTextAttribute(hConsole, 14); |
|  | cout << '\t' << "Enter the number (task 7)"<< '\n'; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | cin >> a\_1; |
|  | if(cin){ |
|  | auto start\_2 = chrono ::high\_resolution\_clock :: now(); |
|  | while((not(b\_1)) && (l <= r)){ |
|  | mid = (l + r) / 2; |
|  | if (arr[mid] == a\_1) b\_1 = true; |
|  | if (arr[mid] > a\_1) r = mid - 1; |
|  | else l = mid + 1; |
|  | } |
|  | if (b\_1){ |
|  | SetConsoleTextAttribute(hConsole,10); |
|  | cout <<"Binary search :" <<'\t'<< "Value found"<< '\n'; |
|  | SetConsoleTextAttribute(hConsole,15); |
|  | } |
|  | else { |
|  | SetConsoleTextAttribute(hConsole,12); |
|  | cout <<"Binary search :" << '\t' << "Value not found"<< '\n'; |
|  | SetConsoleTextAttribute(hConsole,15); |
|  | } |
|  | auto finish\_2 = chrono ::high\_resolution\_clock :: now(); |
|  | chrono :: duration <float> duration\_2 = finish\_2 - start\_2; |
|  | SetConsoleTextAttribute(hConsole,9); |
|  | cout <<"Binary search time = " << duration\_2.count() << " s" << '\n'; |
|  | SetConsoleTextAttribute(hConsole,15); |
|  | b\_1 = false; |
|  | auto start\_3 = chrono ::high\_resolution\_clock :: now(); |
|  | for(int i = 0; i < 100;i ++){ |
|  | if (arr[i] == a\_1){ |
|  | b\_1 = true; |
|  | } |
|  | } |
|  | if (b\_1){ |
|  | SetConsoleTextAttribute(hConsole,10); |
|  | cout << "Enumiration search :" <<" Value found"<< '\n'; |
|  | SetConsoleTextAttribute(hConsole,15); |
|  | } |
|  | else { |
|  | SetConsoleTextAttribute(hConsole, 12); |
|  | cout << "Enumiration search :" << " Value not found"<< '\n'; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | } |
|  | auto finish\_3 = chrono ::high\_resolution\_clock :: now(); |
|  | chrono :: duration <float> duration\_3 = finish\_3 - start\_3; |
|  | SetConsoleTextAttribute(hConsole,9); |
|  | cout <<"Enumeration time = " << duration\_3.count() << " s" << '\n'; |
|  | SetConsoleTextAttribute(hConsole,15); |
|  | } |
|  | else b1 = false; |
|  | } |
|  | cin.clear(); |
|  | while(cin.get() != '\n'); |
|  | b1 = true; |
|  | while(b1){ |
|  | int i1, i2; |
|  | SetConsoleTextAttribute(hConsole, 14); |
|  | cout << '\t' << "Enter numbers (task 8)"<< '\n'; |
|  | SetConsoleTextAttribute(hConsole, 15); |
|  | cin >> i1 >> i2; |
|  | if(cin) arr\_exchange(arr, i1, i2); |
|  | else b1 = false; |
|  | } |
|  | return 0; |
|  | } |
|  | #pragma clang diagnostic pop |