**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: Типы сортировки, массивы.

|  |  |  |
| --- | --- | --- |
| Студент гр. 2372 |  | Братко В.В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение различных сортировок, работа с массивами.

**Основные теоретические положения.**

**Понятие массива**

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.

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.

Аналогом одномерного массива из математики может служить последовательность некоторых элементов с одним индексом: a\_i*ai*​ при  i = 0, 1, 2, … n – одномерный вектор. Каждый элемент такой последовательности представляет собой некоторое значение определенного типа данных. Наглядно одномерный массив можно представить как набор пронумерованных ячеек, в каждой из которых содержится определенное значение:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 3.02 | 1.5 | 7.0 | -2.3 | 12.0 |

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.

**Постановка задачи.**

1. Создает целочисленный массив размерности N = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.
2. Отсортировать заданный в пункте 1 массив быстрой 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. Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения. Выводит индексы всех элементов, которые равны этому значению, и их количество.
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. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором.
8. Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

**Выполнение работы.**

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| 1.Заполнение и сортировка массива | |
| Программа создает, сортирует массив, выводит его на экран. Также выводится время в наносекундах, затраченное на сортировку |  |

Продолжение Таблицы

|  |  |
| --- | --- |
| 2. Поиска максимума и минимума | |
| Сравнивается поиск максимума и минимума в отсортированном и неотсортированном массиве. |  |

|  |  |
| --- | --- |
| 3. Подсчет количества элементов меньше или больше заданного | |
| Пользователь число для того, чтобы узнать сколько существует чисел меньше или больше соотвестственно. |  |

|  |  |
| --- | --- |
| 4 Сравнение поиска элемента бинарным поиском и перебором | |
| Пользователь вводит число, которое хочет найти. |  |

|  |  |
| --- | --- |
| 5 Вывод скорости обмена элементов | |
| Пользователь вводит 2 индекса, которые хочет поменять местами |  |

|  |  |
| --- | --- |
| Цикличность программы | |
| Eсли пользователь желает, то он может запустить выполнение программы заново, нажав цифру 1 |  |

Продолжение Таблицы

|  |  |
| --- | --- |
| Проверка на ошибки | |
| Eсли пользователь вводит неправильный тип данных, то программа на дает ему это сделать |  |

|  |  |
| --- | --- |
| Полный вывод программы | |
| Здесь представлен весь вывод консоли. | Вывод в консоли: |

Продолжение Таблицы

**Полный код программы:**

#include<iostream>  
#include<array>  
#include <chrono>  
  
using namespace std;  
const int N = 100;  
  
void printArr(int arr[] , int length){  
 for (int i = 0;i < length; i++){  
 cout << i << "{"<< arr[i] << "}" << " ";  
 }  
 cout << "\n";  
}  
  
int binarySearch(int arr[], int input){  
 int left = 0, right = N-1;  
 while (left<=right){  
 int mid = (right+left)/2;  
 int value = arr[mid];  
 if (input == value){  
 return mid;  
 }else if (input > value){  
 left = mid + 1;  
 }else {  
 right = mid - 1;  
 }  
 }  
 return NULL;  
}  
  
  
  
  
int checkInput(){  
 int input;  
 try {  
 cin >> input;  
 if (cin.fail()) {  
 throw 1;  
 }  
 } catch (int exeption) {  
 cout << "ERROR!!!";  
 exit(0);  
 }  
 cin.sync();  
 return input;  
}  
  
void bubbleSort(int arr[]){  
 bool flag = true;  
 for (int i = 0; i < N - 1 && flag; i++){  
 flag = false;  
 for (int j = 0; j < N - 1 - i; j++){  
 if (arr[j] > arr[j+1]){  
 swap(arr[j], arr[j+1]);  
 flag = true;  
 }  
 }  
 }  
  
}  
  
void shakerSort(int arr[]){  
 bool flag = true;  
 for (int i = 0; i < N - 1 && flag; i++){  
 flag = false;  
 for (int j = i; j < N - i - 1; j++) {  
 if (arr[j] > arr[j + 1] && j + 1 <= N - 1) {  
 swap(arr[j], arr[j + 1]);  
 flag = true;  
 }  
 if (arr[N - 1 - i - j] < arr[N - 2 - i - j] && N - 2 - i - j >= 0){  
 swap(arr[N - 1 - i - j], arr[N - 2 - i - j]);  
 flag = true;  
 }  
 }  
 }  
}  
  
void combSort(int arr[]){  
 float k = 1.247, S = N-1;  
 int count = 0;  
 while (S >= 1){  
 for (int i = 0; S + i < N; i++){  
 if (arr[i] > arr[int(i + S)]){  
 swap(arr[i], arr[int(i + S)]);  
 }  
 }  
 S /= k;  
 }  
  
 while (true){  
 for (int i = 0; i < N - 1; i++){  
 if (arr[i] > arr[i + 1]){  
 swap(arr[i], arr[i + 1]);  
 }  
 else count++;  
 }  
 if (count == N - 1)  
 break;  
 else  
 count = 0;  
 }  
}  
  
void insertSort1(int arr[]){  
 int key, j;  
 for ( int i = 1; i < N; i++){  
 key = arr[i];  
 j = i - 1;  
 while (j >= 0 && arr[j] > key){  
 arr[j + 1] = arr[j];  
 j--;  
 }  
 arr[j + 1] = key;  
 }  
}  
  
void insertSort2(int arr[]){  
 for (int i = 1; i <= N-1; i++){  
 for (int j = i; j > 0;j--){  
 if (arr[j]<arr[j-1]){  
 swap(arr[j-1], arr[j]);  
 } else {  
 continue;  
 }  
 }  
 }  
}  
  
void quickSort(int arr[], int low, int high)  
{  
 int i = low, j = high;  
 int pivot = arr[(j+i)/2];  
 while (i <= j){  
 while (arr[i] < pivot)  
 i++;  
 while (arr[j] > pivot)  
 j--;  
 if (i <= j)  
 {  
 swap(arr[i], arr[j]);  
 i++;  
 j--;  
 }  
 }  
 if (j > low)  
 quickSort(arr, low, j);  
 if (i < high)  
 quickSort(arr, i, high);  
}  
  
void dopExercise(int arrDop[]){  
 int input1 = 1;  
 printArr(arrDop, N);  
 while (input1 == 1){  
 cout << "input the value by which you want to decrease all odd elements: ";  
 int input = checkInput();  
 int count = 0;  
 srand(time(NULL));  
 for (int i = 0; i < N; i++){  
 if (arrDop[i] % 2 != 0){  
 arrDop[i]-=input;  
 arrDop[i] \*= (rand() % 9) + 1;  
 }  
 if (arrDop[i] % 2 == 0 && arrDop[i] % 3 == 0 && arrDop[i] % 4 == 0 && arrDop[i] % 5 == 0 && arrDop[i] % 7 == 0 && arrDop[i] % 9 == 0){  
 count++;  
 }  
 }  
 printArr(arrDop, N);  
 cout << "exist " << count << " element(s) which execute condition\n";  
 cout << "if you want to repeat then input 1: ";  
 input1 = checkInput();  
 }  
}  
//1. Создать целочисленный массив размерности N = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.  
void function1(int arr[], int arr1[], int arrB[], int arrS[], int arrC[], int arrI[]){  
 srand(time(NULL));  
 int random = 0;  
 for (int i = 0;i < N; i++){  
 random = (rand() % 199) - 99;  
 arr[i] = random;  
 arr1[i] = random;  
 arrB[i] = random;  
 arrS[i] = random;  
 arrC[i] = random;  
 arrI[i] = random;  
 }  
}  
//2. Отсортировать заданный в пункте 1 массив (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.  
void function2(int arr[], int arrB[], int arrS[], int arrC[], int arrI[]){  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 bubbleSort(arrB);  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "time spent to bubble sort in nanoseconds: " << end - start << "ns\n";  
  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 shakerSort(arrS);  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "time spent to shaker sort in nanoseconds: " << end - start << "ns\n";  
  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 combSort(arrC);  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "time spent to comb sort in nanoseconds: " << end - start << "ns\n";  
  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 insertSort1(arrI);  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "time spent to insert sort in nanoseconds: " << end - start << "ns\n";  
  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 quickSort(arr, 0, N-1);  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "time spent to quick sort in nanoseconds: " << end - start << "ns\n";  
  
 printArr(arr, N);  
}  
//3. Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.  
void function3(int arr[],int arr1[]){  
 int max, min;  
 //Найти максимальный и минимальный элемент отсортированного массива  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 max = arr[N-1];  
 min = arr[0];  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << max<< " "<< min << " | time spent searching for the minimum and maximum in the sorted array in nanoseconds: " << end - start << "ns\n";  
 //Найти максимальный и минимальный элемент неотсортированного массива  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 max = arr1[0];  
 min = arr1[0];  
 for (int i = 0; i < N; i++){  
 max = arr1[i]>max ? arr1[i] : max;  
 min = arr1[i]<min ? arr1[i] : min;  
 }  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << max<< " "<< min << " | time spent searching for the minimum and maximum in the unsorted array in nanoseconds: " << end - start << "ns\n";  
 cout << "\n";  
}  
//4. Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения. Выводит индексы всех элементов, которые равны этому значению, и их количество.  
void function4(int arr[]){  
 int average = (arr[N-1]+arr[0])/2+(arr[N-1]+arr[0])%2;  
 int count = 0;  
 cout << "index values equal " << average << ": ";  
 //бинарный алгоритм, который идет в 2 стороны  
 if (binarySearch(arr, average) != NULL){  
 count ++;  
 int i = 1;  
 int indexAverage = binarySearch(arr, average);  
 cout << indexAverage << " ";  
 while ((arr[indexAverage-i] == average || arr[indexAverage+i] == average) && indexAverage-i >= 0 && indexAverage+i <= N-1){  
 if (arr[indexAverage-i] == average){  
 count++;  
 cout << indexAverage-i << " ";  
 }  
 if (arr[indexAverage+i] == average){  
 count++;  
 cout << indexAverage+i << " ";  
 }  
 i++;  
 }  
 }  
 cout << "(" << count << ")";  
 cout << "\n";  
}  
//5. Выводит количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем.  
void function5(int arr[]){  
 int input=1;  
 while (input == 1){  
 cout << "Input value: ";  
 int userInput = checkInput();  
 int left = 0, right = N-1;  
 bool flag = false;  
 while (left<=right){  
 int mid = (right+left)/2;  
 int value = arr[mid];  
 if (userInput == value){  
 flag = true;  
 while (arr[mid - 1] == arr[mid] && mid - 1 >= 0){  
 mid--;  
 }  
 cout << "There are " << mid << " element(s) less then " << userInput << "\n";  
 left = right + 1;  
 }else if (userInput > value){  
 left = mid + 1;  
 }else {  
 right = mid - 1;  
 }  
 }  
 if (!flag){  
 cout << "There are " << left << " element(s) less then " << userInput << "\n";  
 }  
 cout << "\n";  
 cout << "if you want to repeat then input 1: ";  
 input = checkInput();  
 }  
  
}  
// 6. Выводит количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем.  
void function6(int arr[]){  
 int input=1;  
 while (input == 1){  
 cout << "Input value: ";  
 int userInput = checkInput();  
 int left = 0, right = N-1;  
 bool flag = false;  
 while (left<=right){  
 int mid = (right+left)/2;  
 int value = arr[mid];  
 if (userInput == value){  
 flag = true;  
 while (arr[mid + 1] == arr[mid] && mid + 1 <= N - 1){  
 mid++;  
 }  
 cout << "There are " << N - 1 - mid << " element(s) more then " << userInput << "\n";  
 left = right + 1;  
 }else if (userInput > value){  
 left = mid + 1;  
 }else {  
 right = mid - 1;  
 }  
 }  
 if (!flag){  
 cout << "There are " << N - 1 - right << " element(s) more then " << userInput << "\n";  
 }  
 cout << "\n";  
 cout << "if you want to repeat then input 1: ";  
 input = checkInput();  
 }  
}  
//7.Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором.  
void function7(int arr[], int arr1[]){  
 int input=1;  
 while (input == 1) {  
 cout << "input value which you want to search: ";  
 int userInput = checkInput();  
 string str = "not exist";  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 int left = 0, right = N-1;  
 while (left <= right){  
 int mid = (right+left)/2;  
 int value = arr[mid];  
 if (value == userInput){  
 str = "exist";  
 }if (value > userInput){  
 right = mid - 1;  
 }else {  
 left = mid + 1;  
 }  
 }  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "input element: " << str << "\n";  
 cout <<"time spent searching element in the sorted array in nanoseconds: " << end - start << "ns\n";  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 for (int i = 0; i < N; i++){  
 if (arr1[i]==userInput){  
 break;  
 }  
 }  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout <<"time spent searching element in the unsorted array in nanoseconds: " << end - start << "ns\n";  
 cout << "\n";  
 cout << "if you want to repeat then input 1: ";  
 input = checkInput();  
 }  
  
}  
//8. Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.  
void function8(int arr[]){  
 int input=1;  
 while (input == 1){  
 cout << "input 2 indexes (both on a new line) which you want to swap: ";  
 int indexI = checkInput();  
 int indexJ = checkInput();  
 if (indexI >= 0 && indexI <= 99 && indexJ >= 0 && indexJ <= 99){  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 swap(arr[indexI], arr[indexJ]);  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout <<"time spent to swap in nanoseconds: " << end - start << "ns\n";  
 }  
 printArr(arr,N);  
 cout << "\n";  
 cout << "if you want to repeat then input 1: ";  
 input = checkInput();  
 }  
}  
  
  
int main(){  
 int arr[N], arr1[N], arrB[N], arrS[N], arrC[N], arrI[N];  
 //arr[N] - quickSort  
 //arr1[N] - unsort  
 //arrB[N] - bubbleSort  
 //arrS[N] - shakerSort  
 //arrC[N] - combSort  
 //arrI[N] - insertSort  
 //1. Создать целочисленный массив размерности N = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.  
 function1(arr,arr1, arrB, arrS, arrC, arrI);  
 //дополнительное задание №9.  
 //int arrDop[N];  
 //function1(arrDop);  
 //dopExercise(arrDop);  
 //2. Отсортировать заданный в пункте 1 массив (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.  
 function2(arr, arrB, arrS, arrC, arrI);  
 cout << "\n";  
 //printArr(arrS, N);  
 //3. Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.  
 function3(arr, arr1);  
 //4. Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения. Выводит индексы всех элементов, которые равны этому значению, и их количество.  
 function4(arr);  
 cout << "\n";  
 //5. Выводит количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем.  
 function5(arr);  
 cout << "\n";  
 // 6. Выводит количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем.  
 function6(arr);  
 cout << "\n";  
 //7.Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором.  
 function7(arr,arr1);  
 cout << "\n";  
 //8. Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.  
 function8(arr);  
 return 0;  
}

**Вывод:**

В процессе написания программы для практической работы было изучено:

1. Какие виды сортировок бывают
2. Как работает быстрая сортировка
3. Как компьютер представляет массивы в памяти и как с ними работает