**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. |  | Полуянов В. Н. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение одномерных статических массивов и основных алгоритмов работы с ними. Написание программы, заполняющий массив размера, указанного пользователем, случайными числами и сортирующей его пятью разными алгоритмами. Программа должна так же искать различную информацию о массиве, такую как: максимальный и минимальный элементы, их среднее значение и индексы элементов, равные ему; содержится ли в массиве число, введённое пользователем с клавиатуры. Для всех функций необходимо программно рассчитать затраченное время.

**Основные теоретические положения.**

Сортировка пузырьком заключается в попарном сравнении элементов и последующем их обмене. 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 гарантирует, что в конце массива находится максимальный элемент. Хуже всего алгоритм работает, когда на вход подается массив, отсортированный в обратную сторону, быстрее всего – с отсортированным массивом на входе.

Шейкер-сортировка – модификация пузырьковой сортировки. Принцип аналогичен, но имеется существенное отличие. Как только максимальный элемент становится на свое место, алгоритм запускает сортировку в обратную сторону. После выполнения первой итерации минимальный и максимальный элементы будут в начале и конце массива соответственно. За счёт того, что сортировка работает в обе стороны, массив сортируется на порядок быстрее. Но в работе с большими массивами преимущество шейкер-сортировки уменьшается из-за использования двух циклов.

Сортировка вставками. Элементы массива просматриваются по одному, и каждый новый элемент размещается в подходящее место среди ранее упорядоченных элементов. Сортировка вставками отделяет отсортированную часть массива. Лучше всего работает при обработке почти отсортированных массивов. В массивах с длиной меньше 10 она работает быстрее других.

Сортировка расческой – модификация сортировки пузырьком. В этой сортировке переставляются элементы, стоящие на расстоянии. Алгоритм разработан для случаев, когда минимальные элементы стоят слишком далеко от начала массива или максимальные слишком близко к его началу. Эта сортировка неустойчива.

Быстрая сортировка, разработанная в 1960 году английским ученым Чарльзом Хоаром, отличается от других операцией разбиения массива на две части относительно опорного элемента. Вне зависимости от выбора опорного элемента, массив будет отсортирован, но лучшей ситуацией считается та, в которой по сторонам от опорного элемента оказывается примерно равное количество элементов.

**Постановка задачи.**

Разработать алгоритм и написать программу, которая:

1. Создает целочисленный массив размерности N и заполняет его случайными числами в диапазоне от -99 до 99 включительно.
2. Сортирует полученный массив по возрастанию пятью разными сортировками (bubble sort, shaker sort, comb sort, insert sort, quick sort) и определяет время, затраченное на сортировку.
3. Находит максимальный и минимальный элемент массива и определяет время, затраченное на поиск этих элементов в отсортированном и неотсортированном массивах.
4. Находит среднее значение максимального и минимального значения. Выводит индексы элементов, которые равны этому значению, и определяет время, затраченное на поиск.
5. Выводит количество элементов в отсортированном массиве, которые меньше числа, инициализируемого пользователем.
6. Выводит количество элементов в отсортированном массиве, которые больше числа, инициализируемого пользователем.
7. Находит число, заданное пользователем, методом бинарного поиска и методом перебора значений и определяет время, затраченное на поиск.
8. Меняет местами элементы массива, индексы которых вводит пользователь и определяет время обмена.

**Выполнение работы.**

Код программы представлен в приложении А.

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



Рисунок 1. Запуск программы

1. Следующий шаг зависит от введенной команды, если пользователь ввёл:
   1. “0”, то выполнение программы завершается.
   2. “1”, то массив перезаполняется (рис. 2)

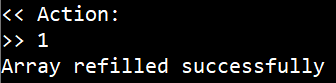


Рисунок 2. Задача 1

* 1. “2”, то выполняется сортировка, указанная в программе, а также время, затраченное на сортировку (рис. 3).

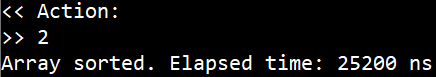


Рисунок 3. Задача 2

* 1. “3”, то при наличии отсортированного массива выполняется поиск максимального и минимального элементов в отсортированном и неотсортированном массивах. Выводится время, затраченное на поиски. (рис. 4).

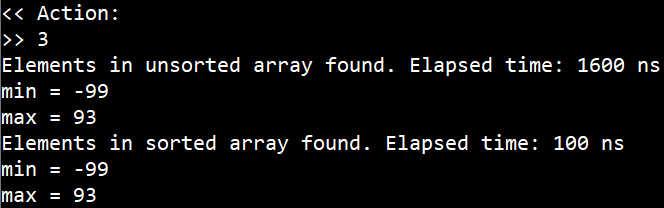


Рисунок 4. Задача 3

* 1. “4”, то выполняется поиск чисел, равных среднему значению минимального и максимального элементов массива. Выводятся индексы элементов и время, затраченное на поиск (рис. 5).

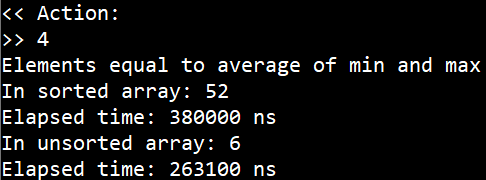


Рисунок 5. Задача 4

* 1. “5”, то в массиве ищется количество чисел, значение которых меньше числа, введенного пользователем (рис. 6).

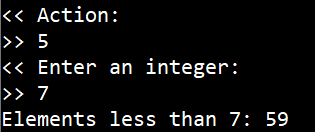


Рисунок 6. Задача 5

* 1. “6”, то в массиве ищется количество чисел, значение которых больше числа, введенного пользователем (рис. 7).

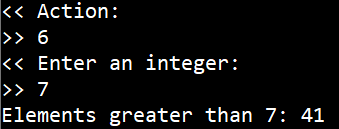


Рисунок 7. Задача 6

* 1. “7”, то выполняется поиск заданного пользователем числа методами бинарного поиска и перебора значений. Выводится время поиска (рис. 8).

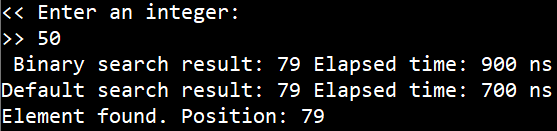


Рисунок 8. Задача 7

* 1. “8”, то выполняется обмен местами чисел, индексы которых вводит пользователь. Подсчитывается время обмена (рис. 9).

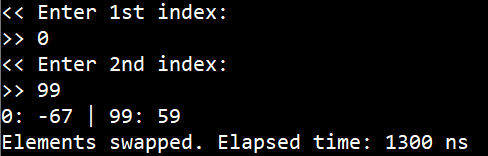


Рисунок 9. Задача 8

* 1. “h”, то выводится меню доступных команд (рис. 10).

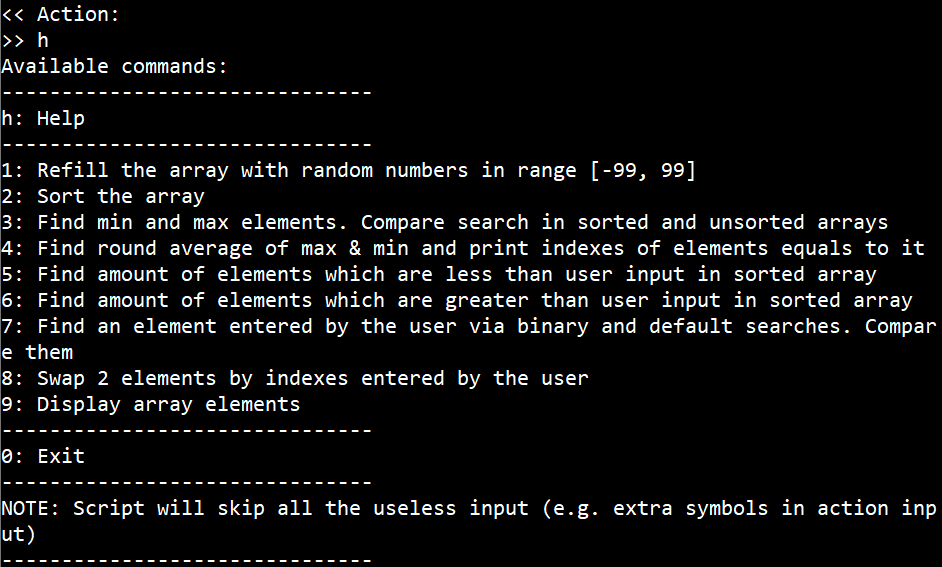


Рисунок 10. Меню

* 1. Любой символ или набор символов, отличный от существующий команд, то будет выведено сообщение об ошибке (рис. 7).



Рисунок 7. Неизвестная команда

**Выводы.**

В ходе работы были изучены одномерные статические массивы и методы работы с ними. Изучены алгоритмы сортировки массивов, алгоритм бинарного поиска.

Приложение А

рабочий код

#include <iostream>  
#include <iomanip>  
#include <cstdlib>  
#include <ctime>  
#include <chrono>  
  
  
void bubbleSort(int arr[], int length) {  
 for (int i = length - 1; i > 0; i--) {  
 for (int j = 0; j < i; j++) {  
 if (arr[j] > arr[j + 1]) {  
 std::swap(arr[j], arr[j + 1]);  
 }  
 }  
 }  
}  
  
  
void shakerSort(int arr[], int length) {  
 for (int i = length - 1; i > 0; i--) {  
 for (int j = 0; j < i; j++) {  
 if (arr[j] > arr[j + 1]) {  
 std::swap(arr[j], arr[j + 1]);  
 }  
 }  
 for (int j = i - 1; j > length - i; j--) {  
 if (arr[j] < arr[j - 1]) {  
 std::swap(arr[j], arr[j - 1]);  
 }  
 }  
 }  
}  
  
  
void combSort(int arr[], int length) {  
 int gap = length;  
  
 while (gap != 1) {  
 gap /= 1.247;  
 if (gap < 1) {  
 gap = 1;  
 }  
  
 for (int i = 0; i < length - gap; i++) {  
 if (arr[i] > arr[i + gap]) {  
 std::swap(arr[i], arr[i + gap]);  
 }  
 }  
 }  
}  
  
  
void insertSort(int arr[], int length) {  
 for (int i = 1; i < length; i++) {  
 for (int j = i; j > 0 && arr[j - 1] > arr[j]; j--) {  
 std::swap(arr[j - 1], arr[j]);  
 }  
 }  
}  
  
  
void quickSort(int arr[], int first, int last) {  
 int mid = arr[(first + last) / 2];  
 int tmpLast = last, tmpFirst = first;  
 while (first < last) {  
 while (arr[first] < mid) first++;  
 while (arr[last] > mid) last--;  
 if (first <= last) {  
 std::swap(arr[first], arr[last]);  
 first++;  
 last--;  
 }  
 }  
 if (tmpFirst < last) quickSort(arr, tmpFirst, last);  
 if (first < tmpLast) quickSort(arr, first, tmpLast);  
}  
  
  
int binarySearch(int arr[], int left, int right, int key) {  
 if (left <= right) {  
 int mid = left + (right - left) / 2;  
 if ((mid == 0 || key > arr[mid - 1]) && arr[mid] == key) {  
 return mid;  
 } else if (key > arr[mid]) {  
 return binarySearch(arr, mid + 1, right, key);  
 } else {  
 return binarySearch(arr, left, mid - 1, key);  
 }  
 }  
 return -1;  
}  
  
  
void printArray(int arr[], int length) {  
 *// Just print array* for (int i = 0; i < length; i++) {  
 std::cout << arr[i] << ' ';  
 }  
 putchar('\n');  
}  
  
  
void fillArray(int arr[], int length) {  
 *// Fill array with pseudo-random nums in range [-99, 99]* std::srand(time(**NULL**));  
 for (int i = 0; i < length; i++) {  
 arr[i] = -99 + (std::rand() % 199);  
 }  
}  
  
  
int main() {  
  
 std::cout.setf(std::ios::fixed);  
 std::cout << "Enter 'h' to get list of commands\n";  
  
 *// Array initialization* const int N = 100;  
 int array[N], unsortedArray[N];  
 bool isArraySorted = false;  
 fillArray(array, N);  
 std::copy(std::begin(array), std::end(array), std::begin(unsortedArray));  
  
 *// Main loop* while (true) {  
  
 *// Get command from user* char userAction;  
 std::cout << "<< Action: \n>> ";  
 std::cin.sync();  
 std::cin >> userAction;  
  
 *// Exit* if (userAction == '0') {  
 break;  
 }  
  
 switch (userAction) {  
  
 *// Refill the array* case '1': {  
 fillArray(array, N);  
 std::cout << "Array refilled successfully\n";  
 std::copy(std::begin(array), std::end(array), std::begin(unsortedArray));  
 isArraySorted = false;  
 break;  
 }  
  
 *// Sort the array* case '2': {  
 auto start = std::chrono::steady\_clock::now();  
 quickSort(array, 0, N-1);  
 auto end = std::chrono::steady\_clock::now();  
 auto elapsed\_us = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count();  
 std::cout << "Array sorted. Elapsed time: " << elapsed\_us << " ns\n";  
 isArraySorted = true;  
 break;  
 }  
  
 *// Find max & min elements in sorted/unsorted array* case '3': {  
 *// UnsortedError* if (!isArraySorted) {  
 std::cout << "Array isn't sorted. Use command '2' to sort it\n";  
 break;  
 }  
  
 int min = 100, max = -100;  
  
 *// Min & max in unsorted array* auto start = std::chrono::steady\_clock::now();  
 for (int i = 0; i < N; i++) {  
 if (unsortedArray[i] < min) {  
 min = unsortedArray[i];  
 }  
 else if (unsortedArray[i] > max) {  
 max = unsortedArray[i];  
 }  
 }  
 auto end = std::chrono::steady\_clock::now();  
 auto elapsed\_us = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count();  
 std::cout << "Elements in unsorted array found. Elapsed time: " << elapsed\_us << " ns\n";  
 std::cout << "min = " << min << "\nmax = " << max << std::endl;  
  
 *// Min & max in sorted array* start = std::chrono::steady\_clock::now();  
 min = array[0];  
 max = array[N-1];  
 end = std::chrono::steady\_clock::now();  
 elapsed\_us = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count();  
 std::cout << "Elements in sorted array found. Elapsed time: " << elapsed\_us << " ns\n";  
 std::cout << "min = " << min << "\nmax = " << max << std::endl;  
  
 break;  
 }  
  
 *// Find avg(max, min) and print indexes of elements equals to it* case '4': {  
 *// In sorted & in unsorted? wdim?  
  
 // UnsortedError* if (!isArraySorted) {  
 std::cout << "Array isn't sorted. Use command '2' to sort it\n";  
 break;  
 }  
  
 *// Average of max and min elements* int avg = (array[N-1] + array[0]) / 2;  
  
 *// Find indexes of elements which equals to average* std::cout << "Elements equal to average of min and max\n";  
  
 *// In sorted array using binary search to find entry* std::cout << "In sorted array: ";  
 auto start = std::chrono::steady\_clock::now();  
 for (int i = binarySearch(array, 0, N, avg); i < N; i++) {  
 if (array[i] != avg) break;  
 std::cout << i << ' ';  
 }  
 auto end = std::chrono::steady\_clock::now();  
 auto elapsed\_us = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count();  
 std::cout << "\nElapsed time: " << elapsed\_us << " ns\n";  
  
 *// In unsorted array* std::cout << "In unsorted array: ";  
 start = std::chrono::steady\_clock::now();  
 for (int i = 0; i < N; i++) {  
 if (unsortedArray[i] == avg) std::cout << i << ' ';  
 }  
 end = std::chrono::steady\_clock::now();  
 elapsed\_us = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count();  
 std::cout << "\nElapsed time: " << elapsed\_us << " ns\n";  
  
 break;  
 }  
  
 *// Find amount of elements which less than user\_input in sorted array* case '5': {  
 *// UnsortedError* if (!isArraySorted) {  
 std::cout << "Array isn't sorted. Use command '2' to sort it\n";  
 break;  
 }  
  
 *// User input* int userInput;  
 std::cout << "<< Enter an integer:\n>> ";  
 std::cin >> userInput;  
 std::cin.sync();  
 if (std::cin.fail()) {  
 std::cout << "TypeError: invalid literal for int with base 10.\n";  
 std::cin.clear();  
 break;  
 }  
  
 *// Search & display* int amount = 0;  
 for (int i = 0; i < N; i++) {  
 amount += array[i] < userInput ? 1 : 0;  
 }  
 std::cout << "Elements less than " << userInput << ": " << amount << std::endl;  
 break;  
 }  
  
 *// Find amount of elements which greater than user\_input in sorted array* case '6': {  
 *// UnsortedError* if (!isArraySorted) {  
 std::cout << "Array isn't sorted. Use command '2' to sort it\n";  
 break;  
 }  
  
 *// User input* int userInput;  
 std::cout << "<< Enter an integer:\n>> ";  
 std::cin >> userInput;  
 std::cin.sync();  
 if (std::cin.fail()) {  
 std::cout << "TypeError: invalid literal for int with base 10.\n";  
 std::cin.clear();  
 break;  
 }  
  
 *// Search & display* int amount = 0;  
 for (int i = N-1; i >= 0; i--) {  
 amount += array[i] > userInput ? 1 : 0;  
 }  
 std::cout << "Elements greater than " << userInput << ": " << amount << std::endl;  
 break;  
 }  
  
 *// Compare binary and default searches (searching for user input). Print time diff* case '7': {  
 *// UnsortedError* if (!isArraySorted) {  
 std::cout << "Array isn't sorted. Use command '2' to sort it\n";  
 break;  
 }  
  
 *// Input* int userInput;  
 std::cout << "<< Enter an integer:\n>> ";  
 std::cin >> userInput;  
 std::cin.sync();  
 if (std::cin.fail()) {  
 std::cout << "TypeError: invalid literal for int with base 10.\n";  
 std::cin.clear();  
 break;  
 }  
 if (userInput > N-1) {  
 std::cout << "IndexError: list index out of range.\n";  
 break;  
 }  
  
 int result;  
  
 *// Binary search* auto start = std::chrono::steady\_clock::now();  
 result = binarySearch(array, 0, N, userInput);  
 auto end = std::chrono::steady\_clock::now();  
 auto elapsed\_us = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count();  
 std::cout << " Binary search result: " << result << " Elapsed time: " << elapsed\_us << " ns\n";  
  
 *// Default search* start = std::chrono::steady\_clock::now();  
 result = -1;  
 for (int i = 0; i < N; i++) {  
 if (array[i] == userInput) {  
 result = i;  
 break;  
 }  
 }  
 end = std::chrono::steady\_clock::now();  
 elapsed\_us = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count();  
 std::cout << "Default search result: " << result << " Elapsed time: " << elapsed\_us << " ns\n";  
  
 *// Output* if (result != -1) {  
 std::cout << "Element found. Position: " << result << std::endl;  
 } else {  
 std::cout << "Element not found.\n";  
 }  
  
 break;  
 }  
  
 *// Swap by indexes entered by the user. Print the time* case '8': {  
 *// Input. This one is kinda huge* int index1, index2;  
 std::cout << "<< Enter 1st index:\n>> ";  
 std::cin >> index1;  
 std::cin.sync();  
 if (std::cin.fail()) {  
 std::cout << "TypeError: invalid literal for int with base 10.\n";  
 std::cin.clear();  
 break;  
 }  
 if (index1 > N-1) {  
 std::cout << "IndexError: list index out of range.\n";  
 break;  
 }  
 std::cout << "<< Enter 2nd index:\n>> ";  
 std::cin >> index2;  
 std::cin.sync();  
 if (std::cin.fail()) {  
 std::cout << "TypeError: invalid literal for int with base 10.\n";  
 std::cin.clear();  
 break;  
 }  
 if (index2 > N-1) {  
 std::cout << "IndexError: list index out of range.\n";  
 break;  
 }  
  
 *// Swap* auto start = std::chrono::steady\_clock::now();  
 std::swap(unsortedArray[index1], unsortedArray[index2]);  
 auto end = std::chrono::steady\_clock::now();  
 auto elapsed\_us = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count();  
 std::cout << index1 << ": " << unsortedArray[index2] << " | ";  
 std::cout << index2 << ": " << unsortedArray[index1] << std::endl;  
 std::cout << "Elements swapped. Elapsed time: " << elapsed\_us << " ns\n";  
 break;  
 }  
  
 *// Display array elements* case '9': {  
 std::cout << "Unsorted array:\n";  
 printArray(unsortedArray, N);  
 if (isArraySorted) {  
 std::cout << "Sorted array:\n";  
 printArray(array, N);  
 }  
 break;  
 }  
  
 *// Help* case 'h': {  
 std::cout << "Available commands:\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "h: Help\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "1: Refill the array with random numbers in range [-99, 99]\n";  
 std::cout << "2: Sort the array\n";  
 std::cout << "3: Find min and max elements. Compare search in sorted and unsorted arrays\n";  
 std::cout << "4: Find round average of max & min and print indexes of elements equals to it\n";  
 std::cout << "5: Find amount of elements which are less than user input in sorted array\n";  
 std::cout << "6: Find amount of elements which are greater than user input in sorted array\n";  
 std::cout << "7: Find an element entered by the user via binary and default searches. Compare them\n";  
 std::cout << "8: Swap 2 elements by indexes entered by the user\n";  
 std::cout << "9: Display array elements\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "0: Exit\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "NOTE: Script will skip all the useless input (e.g. extra symbols in action input)\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << std::setfill(' ');  
 break;  
 }  
  
 *// Unknown command error* default: {  
 std::cout << "RuntimeError: unknown command\n";  
 }  
 }  
 }  
  
 return 0;  
}