**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 | Каплунов 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.**

**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.

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

**Обмен местами элементов массива**

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

**Пузырьковая сортировка массива (bubble sort)**

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.

**Шейкер-сортировка массива (shaker sort)**

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 массива расчёской (comb sort)**

Очевидный недостаток bubble и shaker sort заключается в том, что элементы переставляются максимум на одну позицию.

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.

Оптимально изначально взять расстояние равным длине массива , а далее уменьшать его на определенный коэффициент, который примерно равен 1.247. Когда расстояние станет равно 1, выполняется обычная сортировка пузырьком.

**Сортировка массива вставками (insert sort)**

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.

**Быстрая сортировка массива (quick sort)**

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.

**Бинарный поиск**

Алгоритм бинарного поиска – классический алгоритм поиска в отсортированном массиве, который использует дробление массива на половины. Если элемент, который необходимо найти, присутствует в списке, то бинарный поиск возвращает ту позицию, в которой он был найден.

Бинарный поиск работает только в том случае, если массив отсортирован. Например, если бы искомое минимальное значение стояло не на своем положенном месте, а на месте максимального элемента, то мы бы откинули его на первой же итерации. Сам алгоритм имеет вид:

1)    Определение значения в середине массива (или иной структуры данных). Полученное значение сравнивается с ключом (значением, которое необходимо найти).

2)    Если ключ меньше значения середины, то необходимо осуществлять поиск в первой половине элементов, иначе – во второй.

3)    Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.

4)    Процесс продолжается до тех пор, пока не будет определен элемент, равный значению ключа или не станет пустым интервал для поиска.

Чтобы уменьшить количество шагов поиска, можно сразу смещать границы поиска на элемент, следующий за серединой отрезка.

**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:

* “0” – выполнение цикла завершается.
* “1” – создается массив.

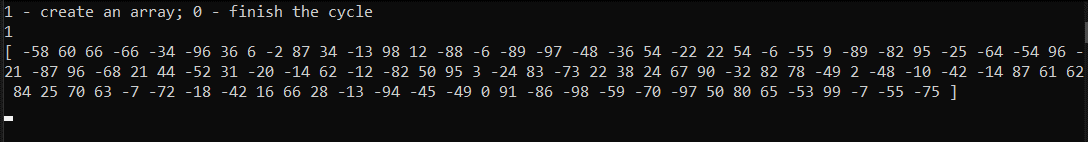


Рисунок 2. Создание массива

1. Далее пользователю будет предложено просмотреть 5 видов сортировок массива:
   * Если пользователь ввёл “0”, то цикл завершится и массив не будет отсортирован.
   * Если “1”, то программа предложит отсортировать массив с помощью пяти на выбор сортировок: bubble sort, shaker sort, insert sort, comb sort, quick sort.
2. После выбора сортировки в терминал будет выведен отсортированный массив и скорость его сортировки:

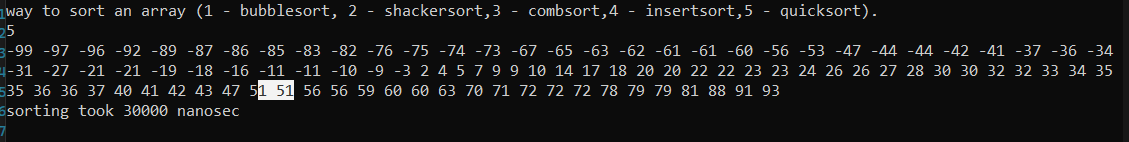


Рисунок 3. Сортировка массива

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

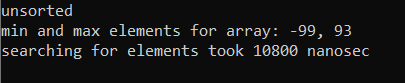


Рисунок 4. Sorted.

* + Неотсортированный массив:

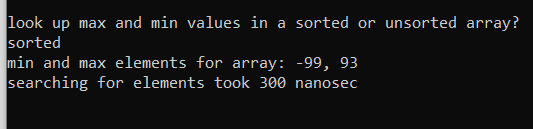


Рисунок 5. Unsorted.

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 и их количество.



Рисунок 6. Avg.

1. Далее пользователю будет предложено ввести число a. После в терминал выведется количество элементов в отсортированном массиве меньше введенного значения.

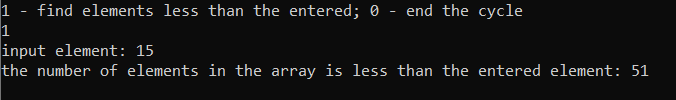


Рисунок 7. Поиск элементов меньше введенного.

1. Далее пользователю будет предложено ввести число a. После в терминал выведется количество элементов в отсортированном массиве больше введенного значения.

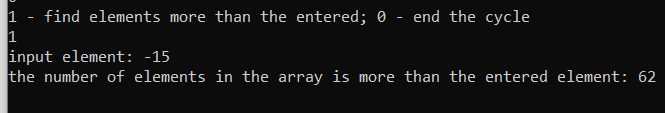
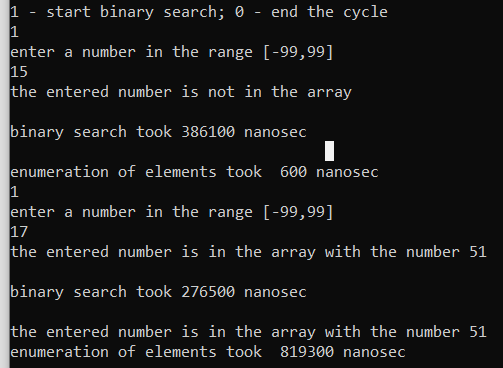
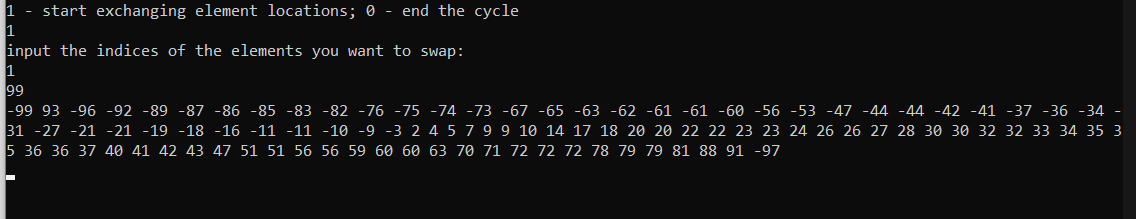


Рисунок 8. Поиск элементов больше введенного.

1. Следующий шаг – бинарный поиск введенного пользователем элемента и поиск перебором, а также сравнение скорости их выполнения. Пользователь вводит число, а программа выводит на экран есть ли оно в массиве и под каким номером оно находится.



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**

#include <iostream>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

using nanoseconds = duration<long long, nano>;

const int N = 100, maxNum = 99, minNum = -99;

void bubblesort(int\* arr, int N) {

auto start = steady\_clock::now();

for (int j = 0; j < N; j++) {

for (int i = 0; i < N - 1; i++) if (arr[i] > arr[i + 1]) swap(arr[i], arr[i + 1]);

}

auto end = steady\_clock::now();

auto elapsed = duration\_cast<nanoseconds>(end - start);

for (int i = 0; i < N; i++) cout << arr[i] << " ";

cout << endl;

cout << "sorting took " << elapsed.count() << " nanosec\n";

}

void shackersort(int\* arr, int N) {

auto start = steady\_clock::now();

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]);

}

}

auto end = steady\_clock::now();

auto elapsed = duration\_cast<nanoseconds>(end - start);

for (int i = 0; i < N; i++) cout << arr[i] << " ";

cout << endl;

cout << "sorting took " << elapsed.count() << " nanosec\n";

}

void combsort(int\* arr, int N) {

auto start = steady\_clock::now();

double S = N - 1, k = 1.3;

int count = 0;

while (S >= 1) {

for (int i = 0; i + S < N; i++) {

if (arr[i] > arr[int(i + S)]) swap(arr[i], arr[int(i + S)]);

}

S /= k;

}

for (int j = 0; j < N; j++) {

for (int i = 0; i < N - 1; i++) if (arr[i] > arr[i + 1]) swap(arr[i], arr[i + 1]);

}

auto end = steady\_clock::now();

auto elapsed = duration\_cast<nanoseconds>(end - start);

for (int i = 0; i < N; i++) cout << arr[i] << " ";

cout << endl;

cout << "sorting took " << elapsed.count() << " nanosec\n";

}

void insertsort(int\* arr, int N) {

auto start = steady\_clock::now();

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;

}

}

auto end = steady\_clock::now();

auto elapsed = duration\_cast<nanoseconds>(end - start);

for (int i = 0; i < N; i++) cout << arr[i] << " ";

cout << endl;

cout << "sorting took " << elapsed.count() << " nanosec\n";

}

void quicksort(int\* arr, int end, int begin, int N) {

int mid;

int b = begin;

int e = end;

mid = arr[(b + e) / 2];

while (b < e)

{

while (arr[b] < mid) b++;

while (arr[e] > mid) e--;

if (b <= e)

{

swap(arr[b], arr[e]);

b++;

e--;

}

}

if (begin < e) quicksort(arr, e, begin, N);

if (b < end) quicksort(arr, end, b, N);

}

void binarySearch(signed\* arr, signed a, signed begin, signed end, int N, int kleft, int kright) {

int kl = kleft, kr = kright;

signed right = end, left = begin;

if (-100 < a && a < 100) {

int mid = (kl + kr) / 2;

if (arr[mid] != a && kr - kl < 2) {

cout << "the entered number is not in the array " << endl;

return;

}

if (arr[mid] == a) {

cout << "the entered number is in the array with the number " << mid << endl;

}

if (arr[mid] > a) {

kr = mid;

right = arr[mid];

binarySearch(arr, a, begin, right, N, kleft, kr);

}

else if (arr[mid] < a) {

kl = mid;

left = arr[mid];

binarySearch(arr, a, left, end, N, kl, kright);

}

}

else cout << "number out of range\n";

}

int main() {

// 1. Целочисленный массив из 100 элементов с диапазоном [-99,99].

int arr[N] = {}, cycleArr;

cout << "1 - create an array; 0 - finish the cycle\n";

cin >> cycleArr;

if (cycleArr == 0) return 0;

while (cycleArr) {

cout << "[ ";

for (int i = 0; i < N; i++) {

arr[i] = minNum + rand() % (maxNum - minNum + 1);

cout << arr[i] << " ";

}

cout << "]" << endl;

cin >> cycleArr;

}

// \*Поиск минимального и максимального элементов в неотсортированном 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);

// 2. Сортировка массива.

int sortName, koef = 0, arrBub[N] = {}, arrShack[N] = {}, arrComb[N] = {}, arrIns[N] = {}, arrQuick[N] = {}, cycleSort, end = N - 1, begin = 0;

cout << "1 - sort an array; 0 - finish the cycle\n";

cin >> cycleSort;

while (cycleSort) {

cout << endl << "way to sort an array (1 - bubblesort, 2 - shackersort,3 - combsort,4 - insertsort,5 - quicksort). \n";

cin >> sortName;

switch (sortName) {

case 1:

for (int i = 0; i < N; i++) {

arrBub[i] = arr[i];

}

bubblesort(arrBub, N);

break;

case 2:

for (int i = 0; i < N; i++) {

arrShack[i] = arr[i];

}

shackersort(arrShack, N);

break;

case 3:

for (int i = 0; i < N; i++) {

arrComb[i] = arr[i];

}

combsort(arrComb, N);

break;

case 4:

for (int i = 0; i < N; i++) {

arrIns[i] = arr[i];

}

insertsort(arrIns, N);

break;

case 5:

for (int i = 0; i < N; i++) {

arrQuick[i] = arr[i];

}

auto start = steady\_clock::now();

quicksort(arrQuick, end, begin, N);

auto end1 = steady\_clock::now();

auto elapsed = duration\_cast<nanoseconds>(end1 - start);

for (int i = 0; i < N; i++)

cout << arrQuick[i] << " ";

cout << endl;

cout << "sorting took " << elapsed.count() << " nanosec\n";

break;

}

cin >> cycleSort;

}

quicksort(arr, end, begin, N);

// 3. Поиск максмимального и минимального элемента.

{

cout << "\nlook up max and min values in a sorted or unsorted array?\n";

string array;

int j = 0;

while (j != 2) {

cin >> array;

if (array == "unsorted") {

cout << "min and max elements for array: " << arrMaxMin[0] << ", " << arrMaxMin[N - 1] << endl; // \*

j += 1;

cout << "searching for elements took " << resultFind.count() << " nanosec\n";

}

else if (array == "sorted") {

auto startFindS = steady\_clock::now();

arr[0], arr[N - 1];

auto endFindS = steady\_clock::now();

auto resultFindS = duration\_cast<nanoseconds>(endFindS - startFindS);

cout << "min and max elements for array: " << arr[0] << ", " << arr[N - 1] << endl;

cout << "searching for elements took " << resultFindS.count() << " nanosec\n";

j += 1;

}

}

}

// 4. Среднее значение минимального и максимального значения. Индексы равные этому значению и их кол-во.

{

int avg = (arr[N - 1] + arr[0]) / 2, k = 0;

cout << endl << "the average of the min and max elements: " << avg << endl;

for (int i = 0; i < N; i++)

if (arr[i] == avg) {

k += 1;

cout << "arr [ " << i << " ] = ";

}

if (k != 0) cout << avg << " ";

cout << "amount of elements: " << k << endl;

}

// 5. Количество элементов в массиве меньше элемента a, введенного пользователем.

int cycle5, a;

cout << "1 - find elements less than the entered; 0 - end the cycle\n";

cin >> cycle5;

while (cycle5) {

int k = 0, i = 0;

cout << "input element: ";

cin >> a;

while (arr[i] < a && i < N) {

i++;

k++;

}

cout << "the number of elements in the array is less than the entered element: " << k << endl;

cin >> cycle5;

}

// 6. Количество элементов в массиве больше элемента b, введенного пользователем.

int cycle6, b;

cout << "1 - find elements more than the entered; 0 - end the cycle\n";

cin >> cycle6;

while (cycle6) {

int k = 0, i = N - 1;

cout << "input element: ";

cin >> b;

while (arr[i] > b) {

i--;

k++;

}

cout << "the number of elements in the array is more than the entered element: " << k << endl;

cin >> cycle6;

}

// 7. Есть ли введенное пользователем число в отсортированном массиве (проверка бинарным поиском и перебором).

int cycle7;

cout << "1 - start binary search; 0 - end the cycle\n";

cin >> cycle7;

while (cycle7) {

// бинарный поиск

int a;

cout << "enter a number in the range [-99,99] \n";

cin >> a;

int kleft = 0, kright = N;

{

signed begin = arr[0], end = arr[N - 1];

auto start = steady\_clock::now();

binarySearch(arr, a, begin, end, N, kleft, kright);

auto end1 = steady\_clock::now();

auto elapsed = duration\_cast<nanoseconds>(end1 - start);

cout << endl;

cout << "binary search took " << elapsed.count() << " nanosec\n";

}

cout << endl;

// перебор

auto start = steady\_clock::now();

for (int i = 0, k = 0; i < N; i++) {

if (arr[i] == a) cout << "the entered number is in the array with the number " << i << endl;

k += 1;

if (k == 0 && i == 99) cout << "the entered number is not in the array\n";

}

auto endTime = steady\_clock::now();

auto elapsed = duration\_cast<nanoseconds>(endTime - start);

cout << "enumeration of elements took " << elapsed.count() << " nanosec\n";

cin >> cycle7;

}

// 8. Обмен мест элементов, индексы которых вводит пользователь.

int cycle8;

cout << "1 - start exchanging element locations; 0 - end the cycle\n";

cin >> cycle8;

while (cycle8) {

int a, b;

cout << "input the indices of the elements you want to swap: \n";

cin >> a >> b;

swap(arr[a], arr[b]);

for (int i = 0; i < N; i++) cout << arr[i] << " ";

cout << endl;

cin >> cycle8;

}

}