**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(ка) гр. |  | Коврижных А.Н. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цели работы**: разработать алгоритм и написать программу, которая сортирует массив.

**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){Ее и использовал в работе}. 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.

Изображение выглядит как диаграмма, текст, линия, снимок экрана

Автоматически созданное описание

Но стандартный алгоритм пузырьковой сортировки предполагает полный циклический проход по массиву. Если изначально подается упорядоченная последовательность, то работа алгоритма все равно продолжиться. Исправить это можно, добавив условие проверки: если на текущей итерации ни один элемент не изменил свой индекс, то работа алгоритма прекращается.

## 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.

Рассмотрим простой пример: имеется массив из 100 элементов, упорядоченных по возрастанию от 1 до 100. Было загадано какое-то число, необходимо его назвать. Компьютер имеет три ответа на ваше предположение: верно, число больше, число меньше. Сколько попыток нужно, чтобы ответить правильно? Обычный перебор – наихудшая стратегия. Можно назвать правильный ответ лишь с 100-ой попытки.

Но если начать спрашивать с середины, то ситуация кардинально меняется. Если число больше 50, то необходимо делить правую половину, и следующее предположение – 75, если меньше – 25. Так необходимо продолжать до тех пор, пока не будет названо правильное число. Наибольшее число предположений равняется:

�=⌈���2100⌉=7*k*=⌈*log*2​100⌉=7

Этот результат явно лучше простого перебора. Схожий принцип работы имеет алгоритм бинарного поиска.

Бинарный поиск работает только в топ 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.

Изображение выглядит как зарисовка, линия, Шрифт

Автоматически созданное описание

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

Предположим, вы ищете слово в словаре с 240 ООО словами. Как вы думаете, сколько попыток вам понадобится в худшем случае?

При простом поиске может потребоваться 240 ООО попыток, если искомое слово находится на самой последней позиции в книге. С каждым шагом бинарного поиска количество слов сокращается вдвое, пока не останется ТОЛЬКО ОДНО СЛОВО.

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

**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 […] 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 в отсортированном и неотсортированном. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитайте время поиска.

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:**

В функции main создаётся целочисленный массив размерности N = 100. В диапазоне от -99 до 99 с рандомно генерирующими числами. Потом по ссылке переносится в другие функии.

for (int i = 0; i < 100; i++)

{

arr[i] = rand() % 199 - 99;

}

Цикл генерирующий массив, также srand(time(NULL)); - это вызов функции srand() с аргументом time(NULL), который позволяет, каждый раз создавать новый массив.

Также в программе используется библиотека “ #include <chrono> ”, которая позволяет засекать время выполнения цикла.

auto startTime = high\_resolution\_clock::now(); - текущие время

auto endTime = high\_resolution\_clock::now(); - текущие время 2

duration<double> dur = endTime - startTime; - время затраченное на программу

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 1. Генерируемый массив.

Нажав 1, программа перейдет в меню. 2 соберет новый массив.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 2. Меню программы.

Выбрав пункт 8 массив прособирается.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 3. Отсортированный массив.

Нажав 1, программа перейдет к следующему действию (к пункту 2), 2 Вернётся в меню. 3 закончит выполнение программы.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 3. Максимальное и минимальное число. Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 4. Среднее значение.

Если в массиве нет элементов равному среднему значению, программа выдаст, что на рисунке 4.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рис. 5. Минимальное число.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рис.6. Максимальное число.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис 7. Поиск числа (\*)

На рис 6. Показано время поиска числа двумя способами: бинарным и перебором.

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

Рис. 8. Выбор типа массива.

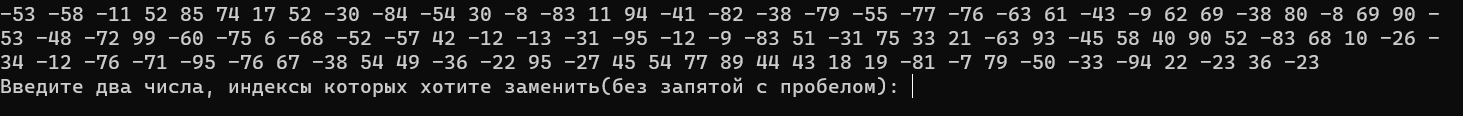


Рис.8.2. Замена числа

Программа предлагает выбрать массив: несортированный и сортированный. Выбрав массив нужно ввести числа, которые надо поменять местами.

**Вывод:**

Я научился создавать массив со случайными числами. Сортировать массив несколькими способами, и использовать библиотеку chrono