# Лабораторная работа №2
# Оценка сложности алгоритмов поиска

## 1Цель работы

1.1 Научиться реализовывать и оценивать сложность алгоритмов поиска элементов массивов на C#.

## 2Литература

2.1 Фленов, М. Е. Библия C#. – 3 изд. – Санкт-Петербург: БХВ-Петербург, 2016. – URL: https://ibooks.ru/bookshelf/353561/reading. – Режим доступа: для зарегистрир. пользователей. – Текст : электронный.

## 3Подготовка к работе

3.1 Повторить теоретический материал (см. п.2).

3.2 Изучить описание лабораторной работы.

3.3 Создать в папке C:\Temp папку с названием группы ISPP-05 для хранения создаваемых приложений.

## 4Основное оборудование

4.1 Персональный компьютер.

## 5Задание

5.1 Реализовать и проверить алгоритм линейного поиска элемента в массиве.

При нахождении элемента возвращается его позиция в структуре данных. Если элемент не найден, возвращать -1. Схема алгоритма – на рисунке 1.

5.2 Реализовать и проверить алгоритм двоичного поиска элемента в отсортированном массиве. Если элемент не найден, возвращать -1. Схема алгоритма – на рисунке 2.

5.3 Реализовать и проверить алгоритм поиска прыжками элемента в отсортированном массиве. Если элемент не найден, возвращать -1.

## 6Порядок выполнения работы

6.1 Запустить MS Visual Studio и создать консольное приложение C# (Console Application) с названием LabWork2.

6.2 Выполнить все задания из п.5 в проекте LabWork2. Каждое задание должно быть в своем проекте с названием Task1, Task2 и т.д.

При выполнении заданий использовать минимально возможное количество команд и переменных и выполнять форматирование и рефакторинг кода.

6.3 Ответить на контрольные вопросы.

## 7Содержание отчета

7.1 Титульный лист

7.2 Цель работы

7.3 Ответы на контрольные вопросы

7.4 Вывод

**8Контрольные вопросы**

8.1 Что такое «алгоритм сортировки»?

8.2 Какие виды поиска элементов массивов существуют?

8.3 В чем особенность алгоритма линейного поиска и какова его временная сложность?

8.4 В чем особенность алгоритма двоичного поиска и какова его временная сложность?

8.5 В чем особенность алгоритма поиска прыжками и какова его временная сложность?

**9Приложение**

9.1 Массивы в C#

Стандартный способ сортировки (вызов у класса Array метода Sort)

```
int[] numbers = new int[] { 97, 45, 32, 65, 83, 23, 15 };
Array.Sort(numbers);
```

9.2 Поиск

Поиск – это задача нахождения индекса, по которому в массиве располагается некоторый заданный элемент.

Алгоритм поиска — это алгоритм нахождения индекса заданного значения в массиве.

Примеры алгоритмов поиска:

- **Линейный поиск** – простейший алгоритм поиска. Это метод полного перебора, и он уступает другим алгоритмам. Алгоритм ищет элемент в заданной структуре данных, пока не достигнет конца структуры. При нахождении элемента возвращается его позиция в структуре данных. Если элемент не найден, возвращаем -1.

*Временная сложность:*

Для получения позиции искомого элемента перебирается набор из N элементов. В худшем сценарии для этого алгоритма искомый элемент оказывается последним в массиве. В этом случае потребуется N итераций для нахождения элемента. Следовательно, временная сложность линейного поиска равна O(N).

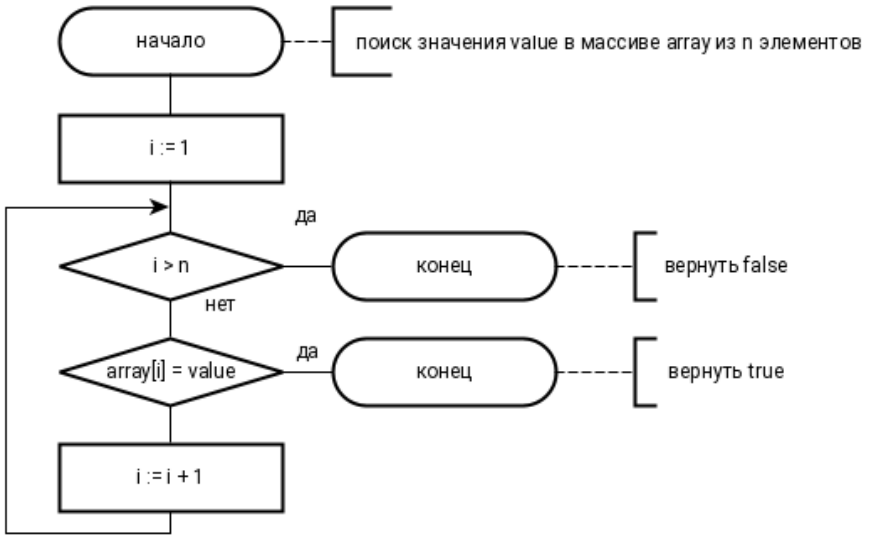


Рисунок 1 — Схема алгоритма линейного поиска

- **Двоичный поиск** — заключается в том, что на каждом шаге множество объектов делится на две части и в работе остаётся та часть множества, где находится искомый объект. Или же, в зависимости от постановки задачи, мы можем остановить процесс, когда мы получим первый или же последний индекс вхождения элемента. Последнее условие — это левосторонний/правосторонний двоичный поиск. Такой поиск требует отсортированной коллекции.

Алгоритм делит входную коллекцию на равные половины, и с каждой итерацией сравнивает целевой элемент с элементом в середине (middle). Поиск заканчивается при нахождении элемента. Иначе продолжаем искать элемент, разделяя и выбирая соответствующий раздел массива. Целевой элемент сравнивается со средним.

Поиск заканчивается, когда firstIndex или left (указатель) достигает lastIndex или right (последнего элемента). Значит, мы проверили весь массив и не нашли элемента.

*Временная сложность* алгоритма двоичного поиска равна O(log (N)) из-за деления массива пополам. Она превосходит O(N) линейного алгоритма.

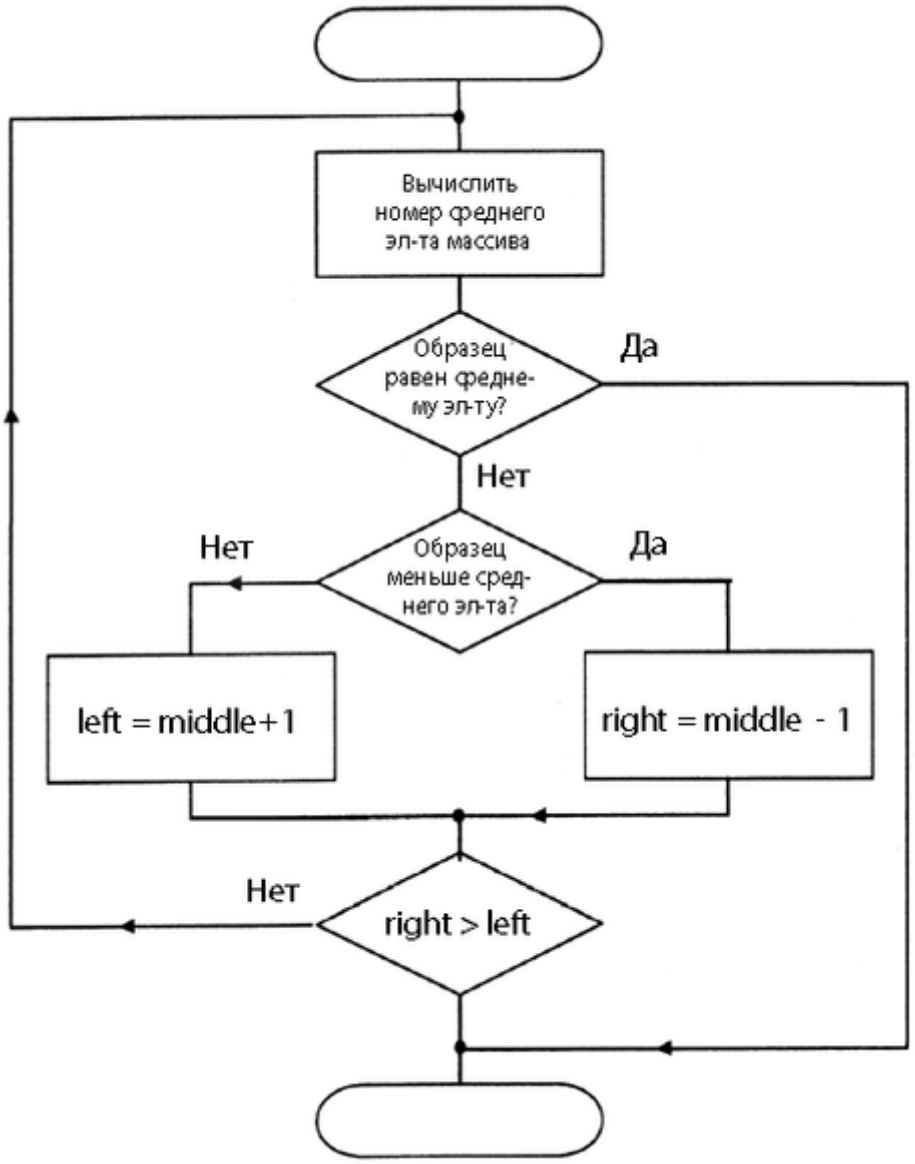


Рисунок 2 — Схема алгоритма двоичного поиска

- **Алгоритм поиска прыжками** — похож на двоичный поиск, но движение только вперёд. Такой поиск требует отсортированной коллекции.

Мы прыгаем вперёд на интервал sqrt(arraylength), пока не достигнем элемента большего, чем текущий элемент или конца массива. При каждом прыжке записывается предыдущий шаг.

Прыжки прекращаются, когда найден элемент больше искомого. Затем запускаем линейный поиск между предыдущим и текущим шагами.

Это уменьшает поле поиска и делает линейный поиск жизнеспособным вариантом.

Мы начинаем с jumpStep размером с корень квадратный от длины массива и продолжаем прыгать вперёд с тем же размером, пока не найдём элемент, который будет таким же или больше искомого элемента.

Сначала проверяется элемент integers[jumpStep], затем integers[2jumpStep], integers[3jumpStep] и так далее. Проверенный элемент сохраняется в переменной previousStep.

Когда найдено значение, при котором

integers[previousStep] < elementToSearch < integers[jumpStep],

производится линейный поиск между integers[previousStep] и integers[jumpStep] или элементом большим, чем elementToSearch.

*Временная сложность:*

Поскольку в каждой итерации мы перепрыгиваем на шаг, равный sqrt(arraylength), временная сложность этого поиска составляет O(sqrt(N)).