Отчет по лабораторной работе №1

Введение в алгоритмы. Сложность. Поиск

Дата: 2025-10-03 **Семестр:** 3 курс 1 полугодие - 5 семестр **Группа:** ПИЖ-б-о-23-2(2) **Дисциплина:** Анализ сложности алгоритмов
Студент: Блинов Виктор Александрович

Цель работы

Освоить понятие вычислительной сложности алгоритма. Получить практические навыки реализации и анализа линейного и бинарного поиска. Экспериментально подтвердить теоретические оценки сложности O(n) и O(log n).

Теоретическая часть

Алгоритмическая сложность характеризует количество ресурсов (времени и памяти), необходимых алгоритму для обработки входных данных размера *n*.

- **Асимптотический анализ** — анализ поведения алгоритма при $n \to \infty$, что позволяет абстрагироваться от аппаратных особенностей и констант.
- **O-нотация** («O-большое») — верхняя асимптотическая оценка роста функции, описывающая наихудший сценарий работы алгоритма.
- **Линейный поиск (Linear Search)** — последовательный перебор всех элементов массива до нахождения целевого значения. Сложность: **O(n)**.
- **Бинарный поиск (Binary Search)** — поиск элемента в отсортированном массиве путём деления интервала поиска пополам. Сложность: **O(log n)**. Требует предварительной сортировки массива.

Практическая часть

Выполненные задачи

- ☑ Реализована функция линейного поиска `linear_search(arr, target)`.
- ☑ Реализована функция бинарного поиска `binary_search(arr, target)`.
- ☑ Проведён теоретический анализ сложности функций.
- ☑ Замерено время выполнения алгоритмов на массивах различных размеров.
- ☑ Результаты визуализированы в виде графиков с линейной и логарифмической шкалой.
- ☑ Добавлены характеристики ПК и проведён анализ расхождений теории и практики.

Ключевые фрагменты кода

```
# Линейный поиск O(n)
def linear_search(arr, target):
```

```
    for i in range(len(arr)):          # O(n)
        if arr[i] == target:           # O(1)
            return i                   # O(1)
    return None                         # O(1)

# Бинарный поиск O(log n)
def binary_search(arr, target):
    left, right = 0, len(arr) - 1      # O(1)
    while left <= right:               # O(log n)
        mid = (left + right) // 2      # O(1)
        if arr[mid] == target:         # O(1)
            return mid                 # O(1)
        elif arr[mid] < target:        # O(1)
            left = mid + 1             # O(1)
        else:
            right = mid - 1            # O(1)
    return None                        # O(1)
```

Результаты выполнения

Характеристики ПК для тестирования

```
Процессор: 12th Gen Intel(R) Core(TM) i5-12450H 2.00 GHz
Оперативная память: 16 GB DDR4
ОС: Windows 11 Pro
Python: 3.10.10
```

Экспериментальные данные

```
Размер массива | Linear (ms) | Binary (ms)
---------------------------------------
n=  1000 | linear=0.01339 ms | binary=0.00092 ms
n=  2000 | linear=0.03028 ms | binary=0.00105 ms
n=  5000 | linear=0.06908 ms | binary=0.00115 ms
n= 10000 | linear=0.14336 ms | binary=0.00124 ms
n= 20000 | linear=0.28706 ms | binary=0.00160 ms
n= 50000 | linear=0.77764 ms | binary=0.00177 ms
n=100000 | linear=1.38712 ms | binary=0.00187 ms
n=200000 | linear=3.13556 ms | binary=0.00213 ms
```

- **time_vs_n_linear.png** — график зависимости времени поиска от размера массива в линейной шкале.
- **time_vs_n_loglog.png** — график зависимости времени поиска в логарифмической шкале (log-log).

Графики показывают линейный рост времени работы линейного поиска и логарифмический — бинарного поиска, что подтверждает теоретическую асимптотику.

Тестирование

- ☑ Поиск проверен на элементах в начале, середине, конце массива и на отсутствующем элементе.
- ☑ Замеры времени проведены на массивах от 1 000 до 200 000 элементов.

Выводы

1. Линейный поиск показал рост времени выполнения, пропорциональный размеру массива, что соответствует O(n).
2. Бинарный поиск масштабируется значительно лучше и демонстрирует рост близкий к O(log n), особенно на больших массивах.
3. Эксперимент подтвердил теоретические оценки сложности; небольшие расхождения связаны с влиянием констант, особенностями Python и архитектуры ПК.

Ответы на контрольные вопросы

1. **Что такое асимптотическая сложность алгоритма и зачем она нужна?**
   Асимптотическая сложность — это оценка количества ресурсов алгоритма в зависимости от размера входных данных *n*. Нужна для сравнения эффективности алгоритмов и выбора оптимальных решений при работе с большими данными.

2. **Разница между O(1), O(n) и O(log n). Примеры:**
   - O(1) — время не зависит от размера входных данных (доступ к элементу массива).
   - O(n) — время растёт линейно (линейный поиск).
   - O(log n) — время растёт логарифмически (бинарный поиск в отсортированном массиве).

3. **Отличие линейного поиска от бинарного. Условия бинарного поиска:**
   Линейный поиск перебирает элементы по одному (O(n)).
   Бинарный поиск делит отсортированный массив пополам на каждом шаге (O(log n)); требует предварительной сортировки массива и индексного доступа.

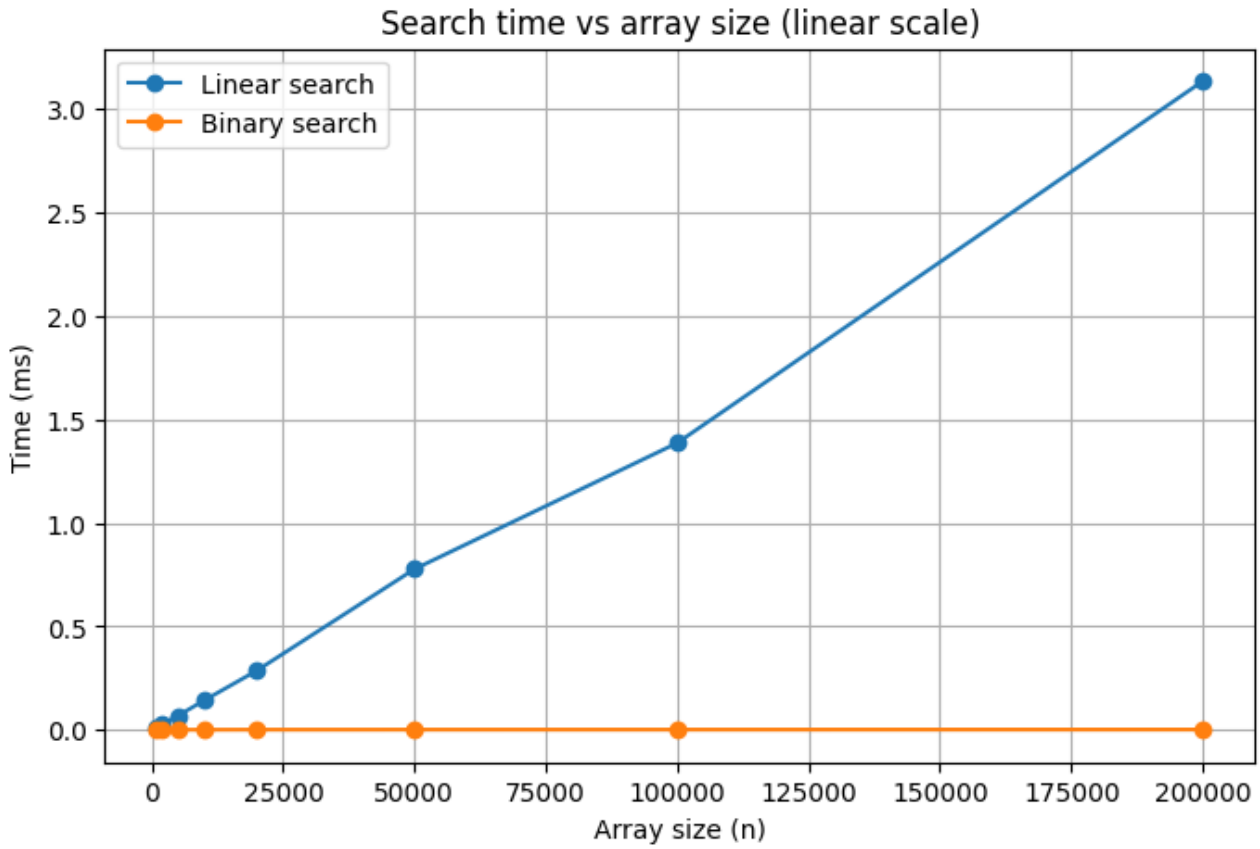
4. **Почему на практике время может отличаться от теоретической оценки:**
   Из-за скрытых констант в O-нотации, оптимизаций Python, особенностей архитектуры ПК (кэш, память), а также характера входных данных (лучший, худший, средний случай).

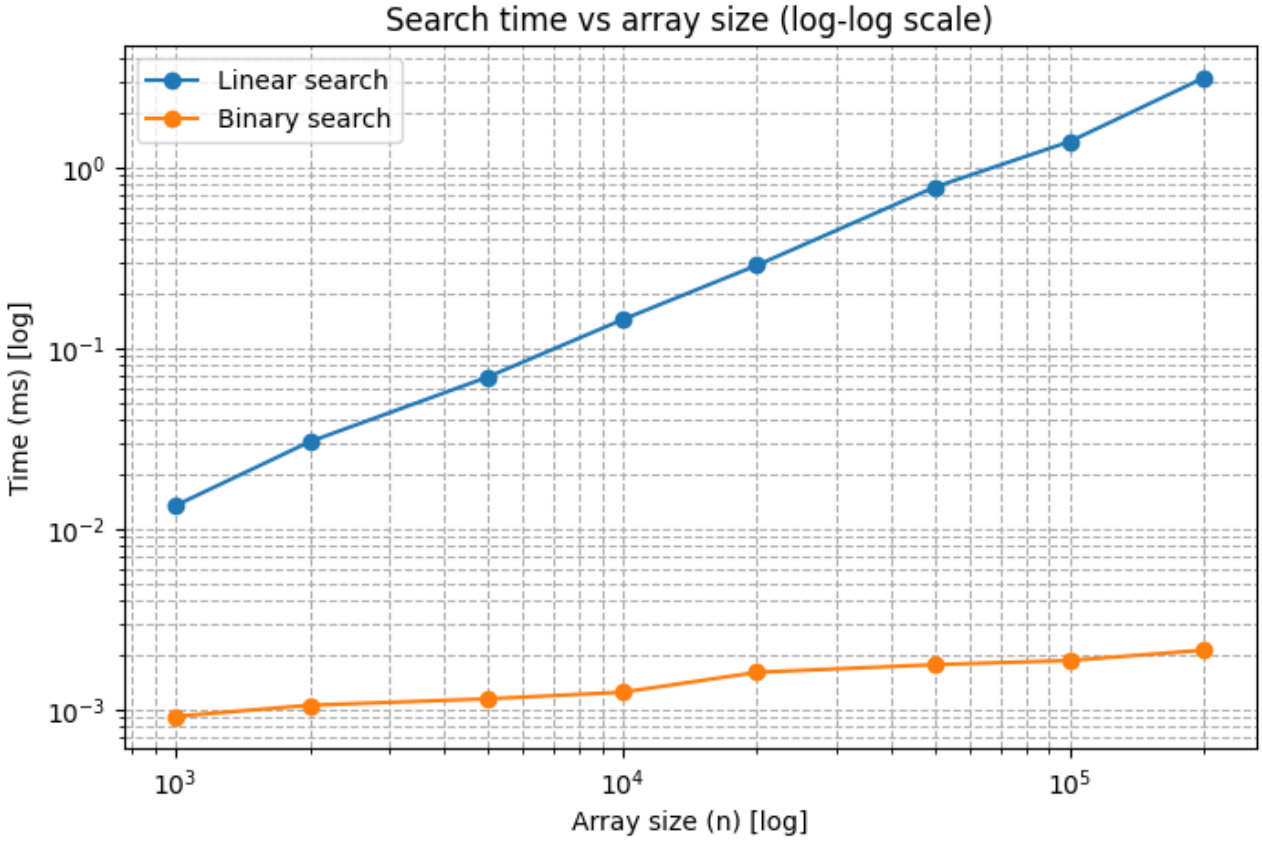
5. **Как экспериментально подтвердить сложность:**
   Провести замеры времени выполнения на массивах разного размера, построить графики зависимости времени от *n* и сравнить их с теоретическими кривыми роста.

Приложения

- Исходный код программы
- Результаты замеров времени (results.txt)

- Графики:


Search time vs array size (linear scale)
Linear search
Binary search
Time (ms)
Array size (n)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
25000
50000
75000
100000
125000
150000
175000
200000


Search time vs array size (log-log scale)
Linear search
Binary search
Time (ms) [log]
Array size (n) [log]
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{3}
10^{4}
10^{5}