УРОК 7

Временная сложность алгоритмов.
Алгоритмы поиска и сортировки

Концепция Big O

Умение видеть и исправлять неоптимальный код

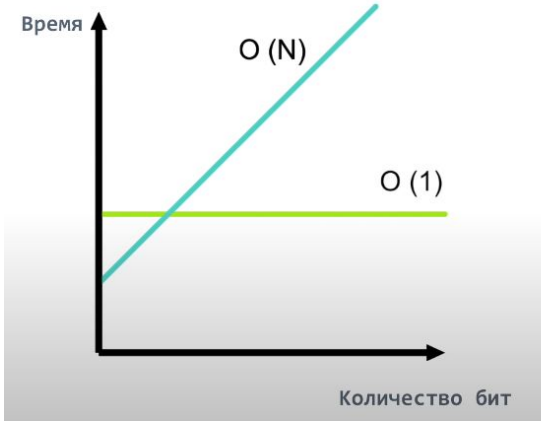
Ни один серьезный проект, как и ни одно серьезное собеседование не могут обойтись без вопросов о Big O

Непонимание Big O ведет к серьезной потери производительности ваших алгоритмов

Измерение скорости алгоритмов

Скорость алгоритма измеряется не в секундах, а в приросте количества операций при наихудшем сценарии.

Big O показывает зависимость между входными параметрами функции и количеством операций, которые выполнит процессор .



Передача файлов по сети: O(N)

Больше байт – дольше передавать

Перенос носителя на самолете: O(1)

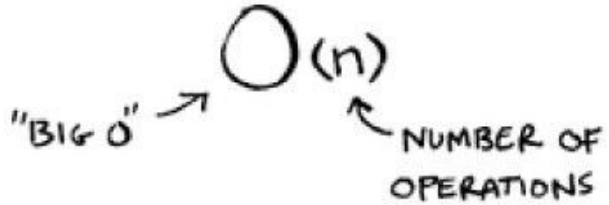
Размер файла не важен – скорость самолета не изменится

Оценка скорости выполнения

```
int pairSumSequence(int n) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    sum += pairSum(i, i + 1);
  }
  return sum;
}
int pairSum(int a, int b) {
  return a + b;
}
```

O(n)

O(1)



Отбрасывание констант и неважная сложность

$N = \infty$, $N + 1 = \infty$, $2N = \infty$

Big O описывает **только** скорость роста, поэтому мы отбрасываем константы при оценке сложности, а также части несущие меньшую нагрузку

$O(100N) = O(N)$

$O(N^2 + N) = O(N^2)$

$O(N + \log N) = O(N)$

$O(N^2 + B) = O(N^2 + B)$

Как просчитывать Big O (сложение или умножение)

```
for (int a : arrA) {
  print(a);
}
for (int b : arrB) {
  print(b);
}
```

Сложение O(A + B)

Последовательность действий – **сложение**

```
for (int a : arrA) {
  for (int b : arrB) {
     print(a + "," + b);
  }
}
```

Умножение O(A * B)

Выполнить что-то N раз, пока делаешь что-то свое – **умножение**

Для алгоритма, где на каждой итерации берется половина элементов - сложность будет включать O(Log N)

Типичные примеры «O-большого»

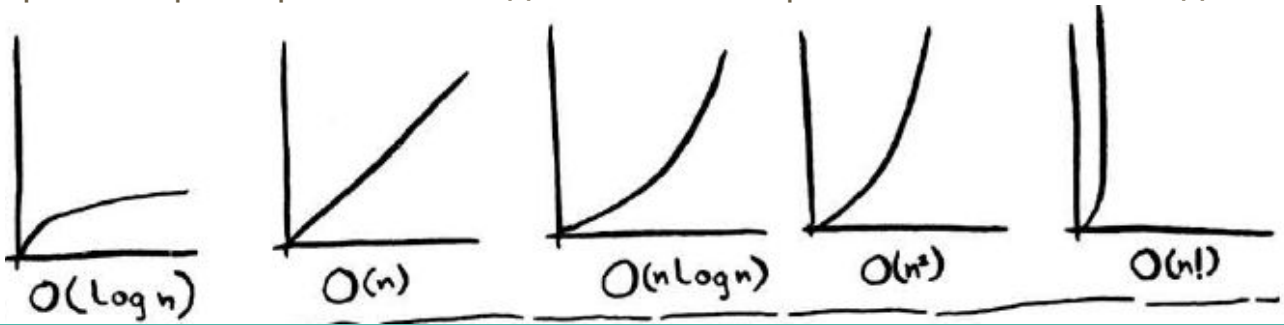
O(log n) — логарифмическое время. Пример: двоичный поиск.

O(n) — линейное время. Пример: простой поиск.

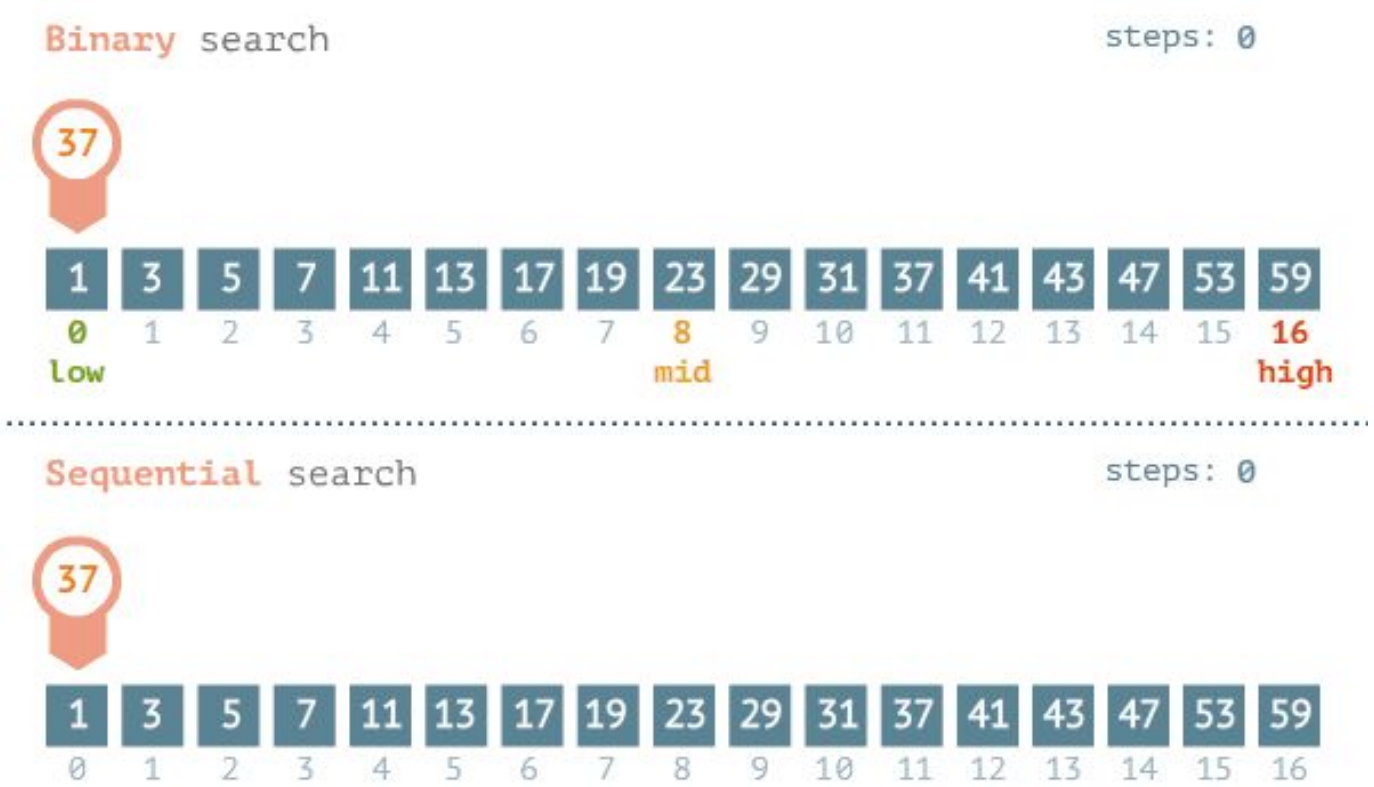
O(n * log n). Пример: быстрый алгоритм сортировки, такой как quicksort (быстрая сортировка).

$O(n^2)$ — квадратичное время. Пример: медленный алгоритм сортировки, такой как сортировка выбором.

O(n!) — факториальное время. Пример: очень медленный алгоритм, такой как в задаче коммивояжера.



Линейный поиск vs Бинарный поиск



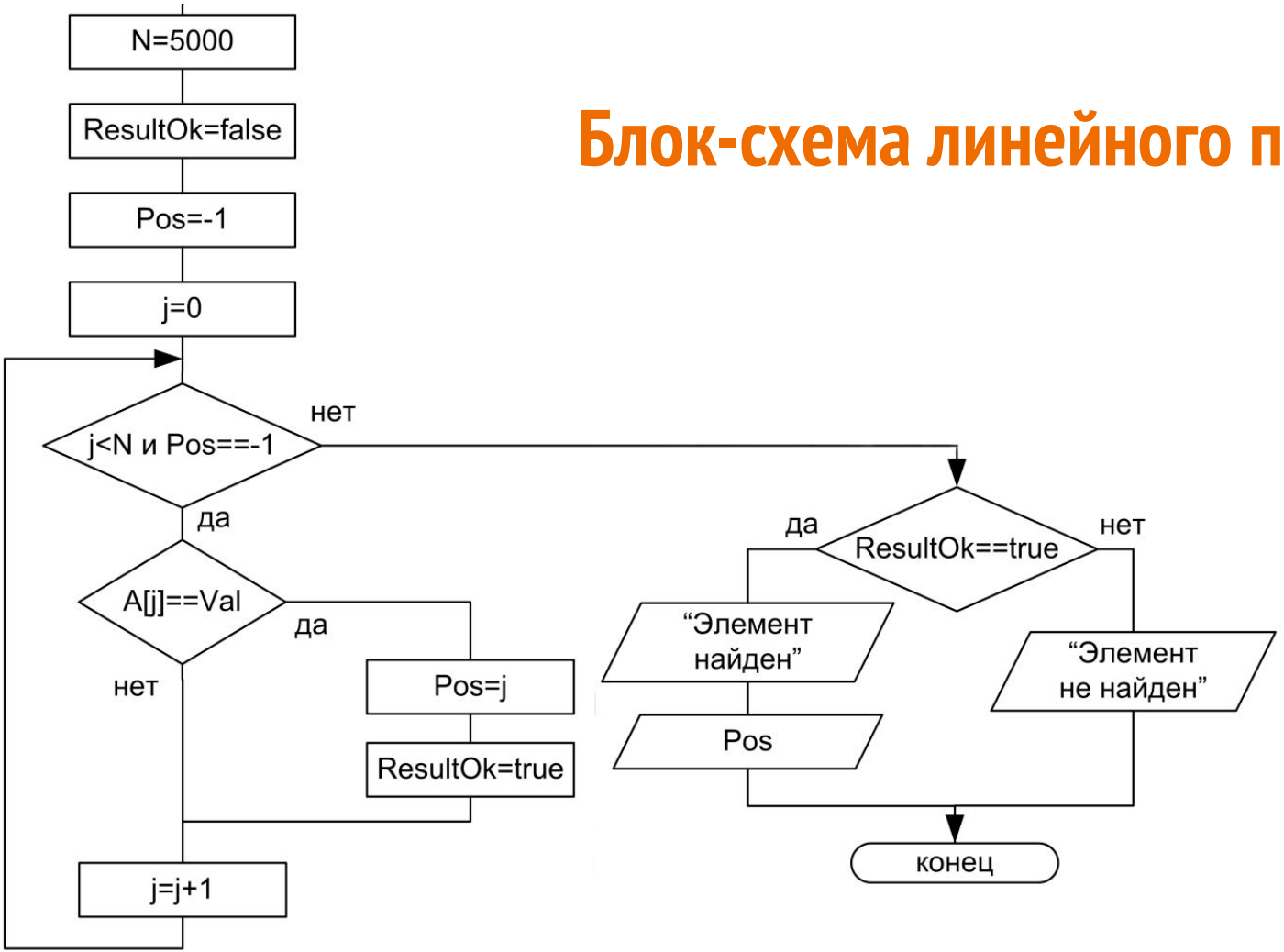
Отличия бинарного и линейного поиска

Временная сложность линейного поиска составляет O (N), в то время как двоичный поиск имеет O ($\log_2$ N).

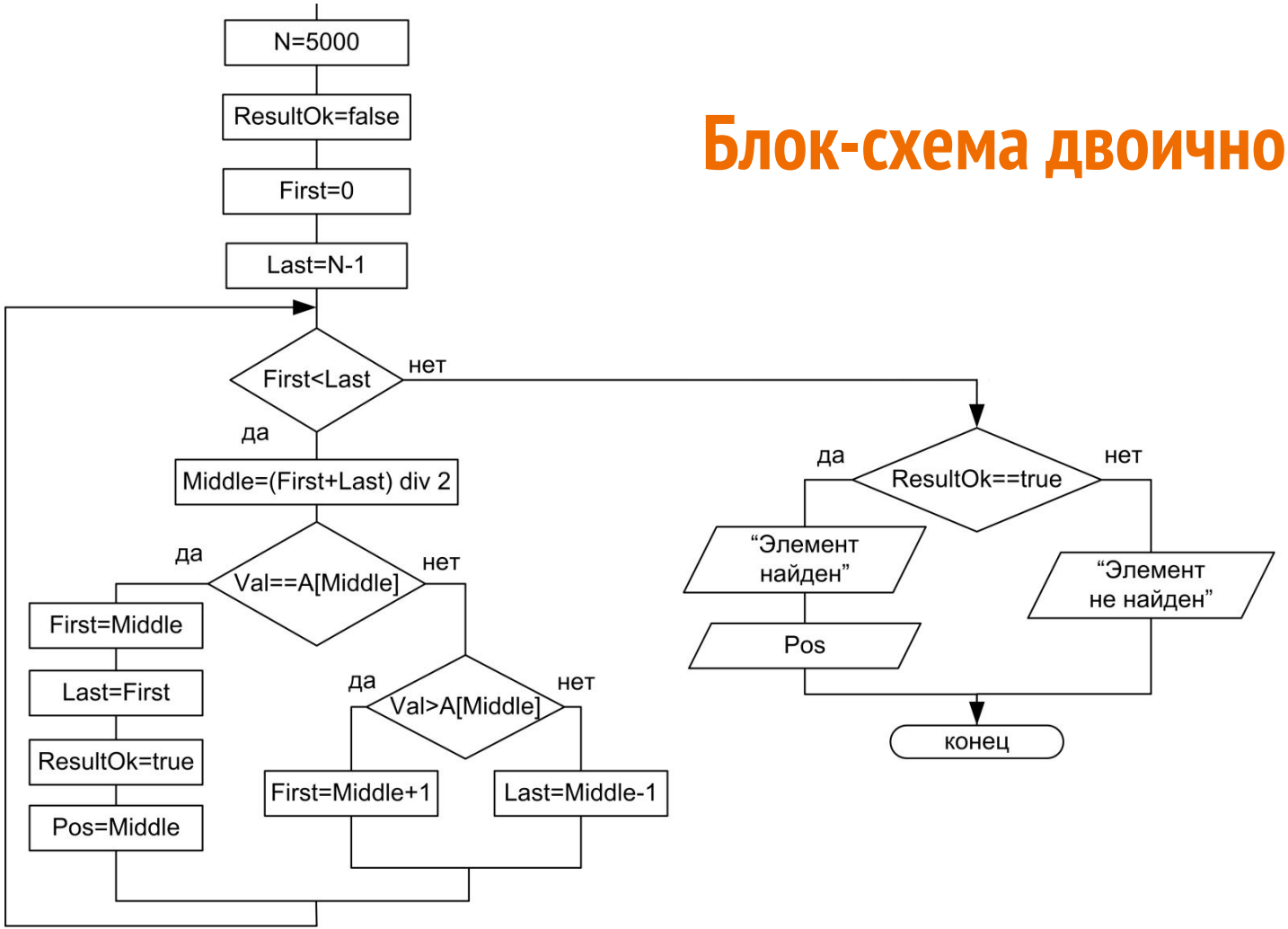
Линейный поиск может быть реализован как в индексируемых наборах, так и в связанном списке, тогда как бинарный поиск не может быть реализован непосредственно в связанном списке.

Как мы знаем, для бинарного поиска требуется отсортированный массив, что является причиной, по которой требуется обработка для вставки в нужное место для поддержания отсортированного списка. Напротив, линейный поиск не требует отсортированных элементов, поэтому элементы легко вставляются в конец списка.

Блок-схема линейного поиска



Блок-схема двоичного поиска



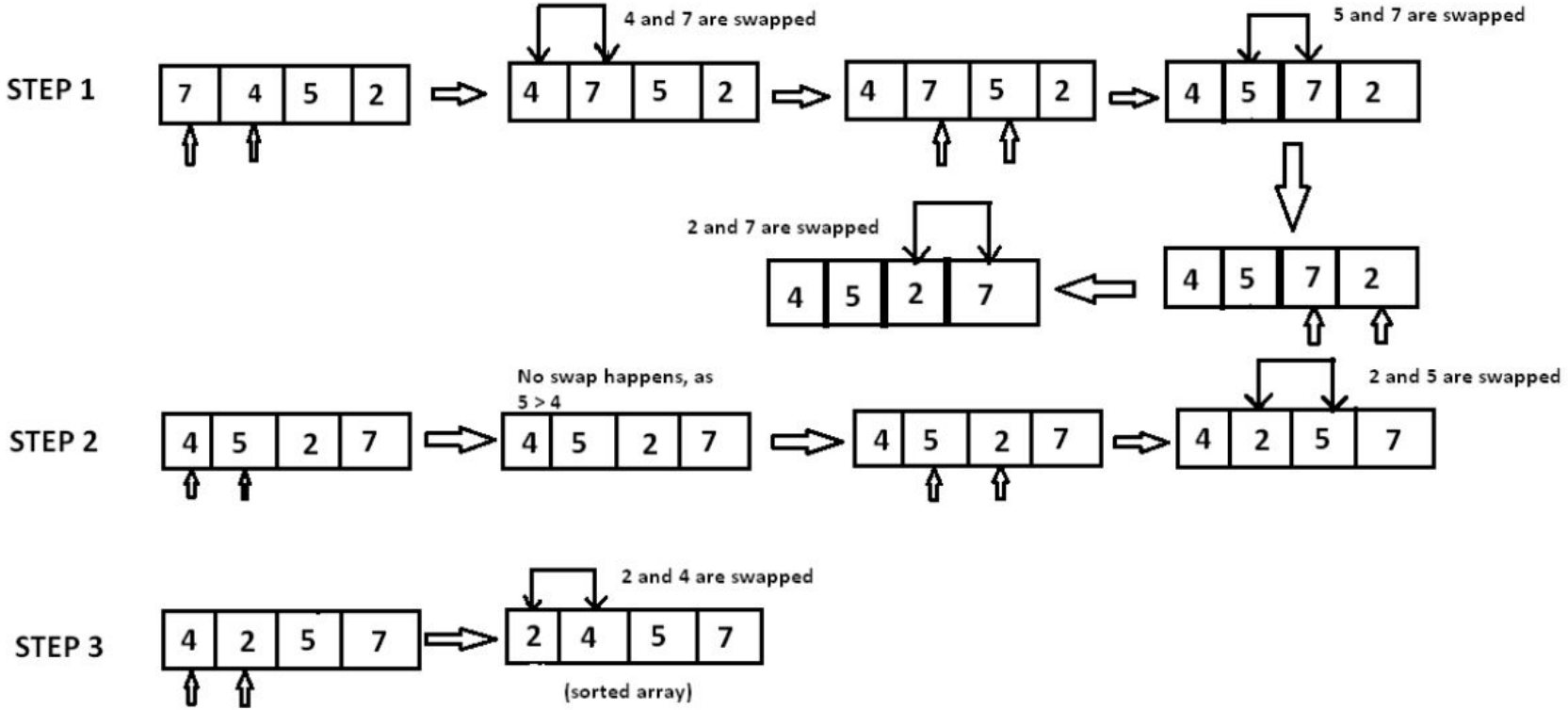
Bubble sort (Сортировка пузырьком)

Данный алгоритм меняет местами два соседних элемента, если первый элемент массива больше второго.

Так происходит до тех пор, пока алгоритм не поменяет местами все не отсортированные элементы.

Сложность данного алгоритма сортировки равна $O(n^2)$. **Почему $O(n^2)$?**

Bubble sort (Сортировка пузырьком)



Selection sort (Сортировка выбором)

Суть алгоритма заключается в проходе по массиву от начала до конца в поиске минимального элемента массива и перемещении его в начало.

Алгоритм сортировки выбором легко объясняется, но медленно работает.

Сложность такого алгоритма $O(n^2)$.

Selection sort (Сортировка выбором)