# 7. Итераторы, выделение памяти new/delete

## Алгоритмы бинарного поиска и метод двух указателей

Программирование и алгоритмизация

Практические занятия

БИВТ-24-17

**Надежда Анисимова**

ms teams m2102039@edu.misis.ru

# Проверка себя

1. Что такое указатель? Зачем нужен nullptr?
2. Чем могут являться * и &?
3. Чем отличаются ссылки и указатели?
4. В чем разница между константным указателем и указателем на константу?

# Выделение динамической памяти

01

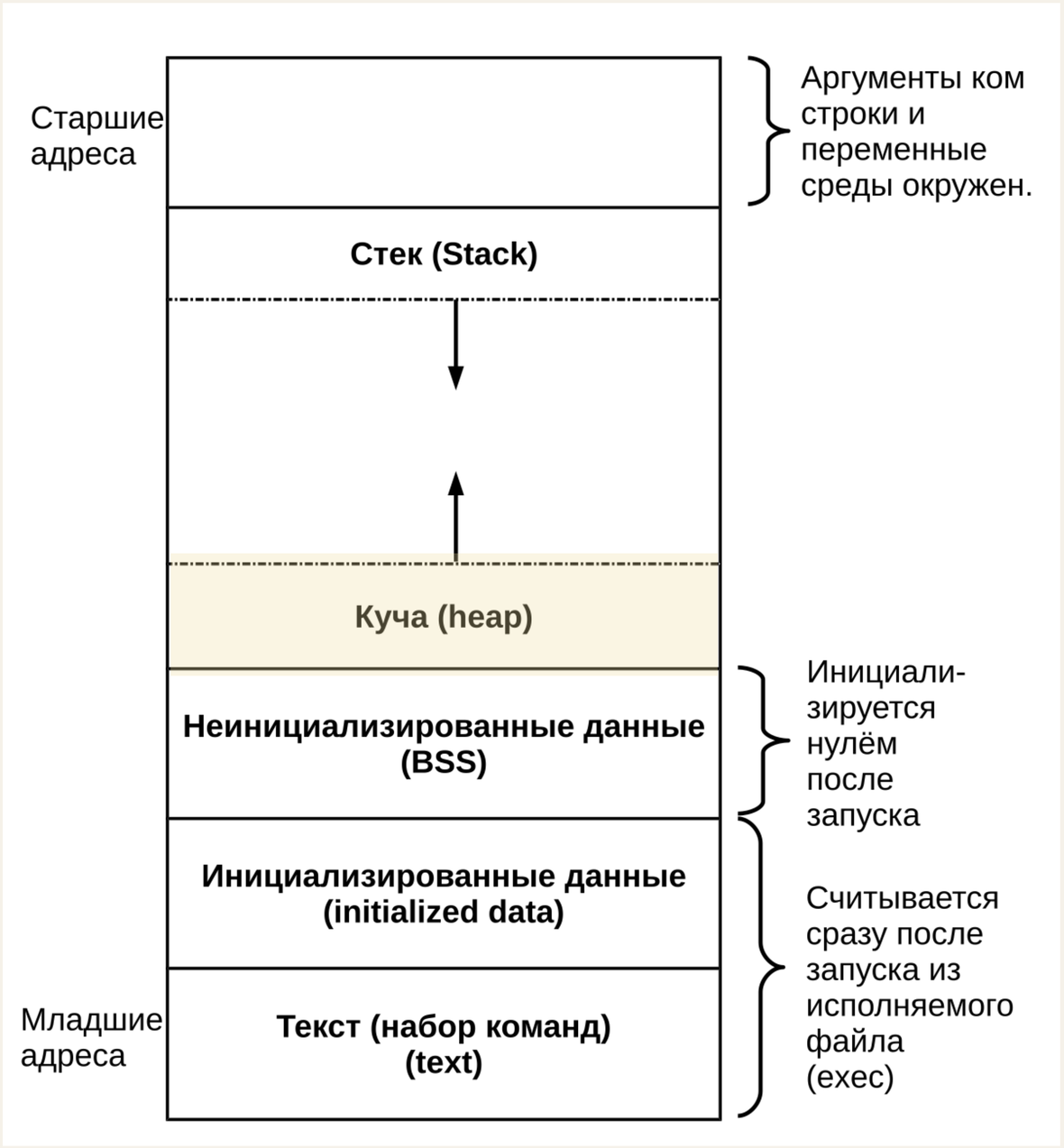
## Вспоминаем

# Куча (heap)

Область памяти, используемая для **динамического выделения памяти**.

Переменные, выделяемые в куче, определяются только во время выполнения программы и могут быть изменены в размере по мере необходимости.

Управление памятью на куче обычно осуществляется функциями, такими как malloc() и free() в C, или **операторами new и delete** в C++



# **new, delete –** операторы для работы с динамической памятью

new – для выделения памяти под указанный тип данных с автоматическим вызовом конструктора;

delete – освобождение памяти с автоматическим вызовом деструктора.

```
int* p = new int; //выделение в куче памяти под int
delete p; //освобождение выделенной памяти
```

**new, delete –** операторы для работы с динамической памятью

new [] – для выделения памяти под указанное число объектов;

delete [] – для освобождения памяти массива объектов.

```
double* arr = new double[10]; //выделение под 10 double и указатель на первый элемент

delete[] arr; // освобождение памяти для всх 10 элементов
```

Чтобы не произошла утечка памяти и другие проблемы:

- Если выделили память с помощь new - нужно освободить в помощью delete
- Если выделили память с помощь new []- нужно освободить в помощью delete[]

Пример итерации по массиву через указатели

```
int* arr = new int[5]{1, 2, 3, 4, 5};

for (int* ptr = arr; ptr != arr + 5; ++ptr) {
   std::cout << *ptr << " ";
}
```

# Итераторы

02

Итератор – объект, который **обобщает концепцию указателей**. Используется для перебора элементов в контейнерах, доступа к элементу контейнера

В отличие от указателей, итераторы – **унифицированный** способ для доступа к элементам различных контейнеров. Кроме того, итераторам доступен дополнительный функционал

```
<type>::iterator it;
```

Тип контейнера

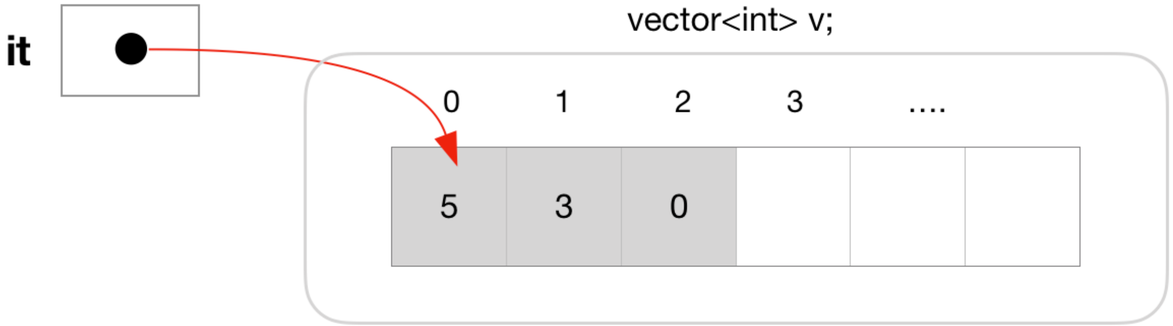
```
vector<int>::iterator it;

auto it;
```

можно не указывать тип, используя auto

# Итераторы, как функции контейнеров

```
auto it = vec.begin();
```

- begin() – на первый элемент
- end() – на следующий после последнего
- rbegin(), rend() – reverse обратном порядке
- cbegin(), cend() – константный




```
vector<int> vec{ 5, 3, 0 };
for (auto it = begin(vec); it != end(vec); it++)
{
  cout << *it << " ";
}
```

# Допустимые операции

- Разыменование `*it = new_value;`
- Инкремент/декремент `it++;` `++it;` `it--;` `--it;`
- Прибавление/вычитание целого числа `it + 5;` `it - 5;`
- Вычитание другого указателя - дистанция `it2 - it1;`
- Сравнение `it1 != it2;` `it1 > it2;`

# Итераторы делятся на категории

- Категории определяются не типом итератора, а допустимыми операциями с ним

| Iterator category | Operations and storage requirement | | | | | | |
|---|---|---|---|---|---|---|---|
| | write | read | increment | | decrement | random access | contiguous storage |
| | | | without multiple passes | with multiple passes | | | |
| *LegacyIterator* | | | Required | | | | |
| *LegacyOutputIterator* | Required | | Required | | | | |
| *LegacyInputIterator* (mutable if supports write operation) | | Required | Required | | | | |
| *LegacyForwardIterator* (also satisfies *LegacyInputIterator*) | | Required | Required | Required | | | |
| *LegacyBidirectionalIterator* (also satisfies *LegacyForwardIterator*) | | Required | Required | Required | Required | | |
| *LegacyRandomAccessIterator* (also satisfies *LegacyBidirectionalIterator*) | | Required | Required | Required | Required | Required | |
| *LegacyContiguousIterator*[1] (also satisfies *LegacyRandomAccessIterator*) | | Required | Required | Required | Required | Required | Required |

- Output Stream
- Input Stream
- forward_list
- list, map, set
- vector, deque, array

https://en.cppreference.com/w/cpp/iterator

В c++20 введены новые понятия **концепций(concept)** и основанная на них новая система итераторов, а также библиотека <ranges>

c++20

- std::input_iterator
- std::output_iterator
- std::forward_iterator
- std::bidirectional_iterator
- std::random_access_iterator

**До**

```
auto it = std::find(numbers.begin(), numbers.end(), 4);
std::sort(numbers.begin(), numbers.end());
```

**После**

```
auto it = std::ranges::find(numbers, 4);
std::ranges::sort(numbers);
```

*концепции - механизм, позволяющий задавать ограничения на шаблоны

Что еще?

- Итераторы-адаптеры

  std::reverse_interator - перебор в обратном порядке

  back_insert_iterator - позволяет в конец контейнера добавлять элемент

- Функции

```
auto vi = v.begin();
std::advance(vi, 2);
```

увеличение итератора на n

```
std::distance(v.begin(), v.end());
```

расстояние между итераторами

и другие

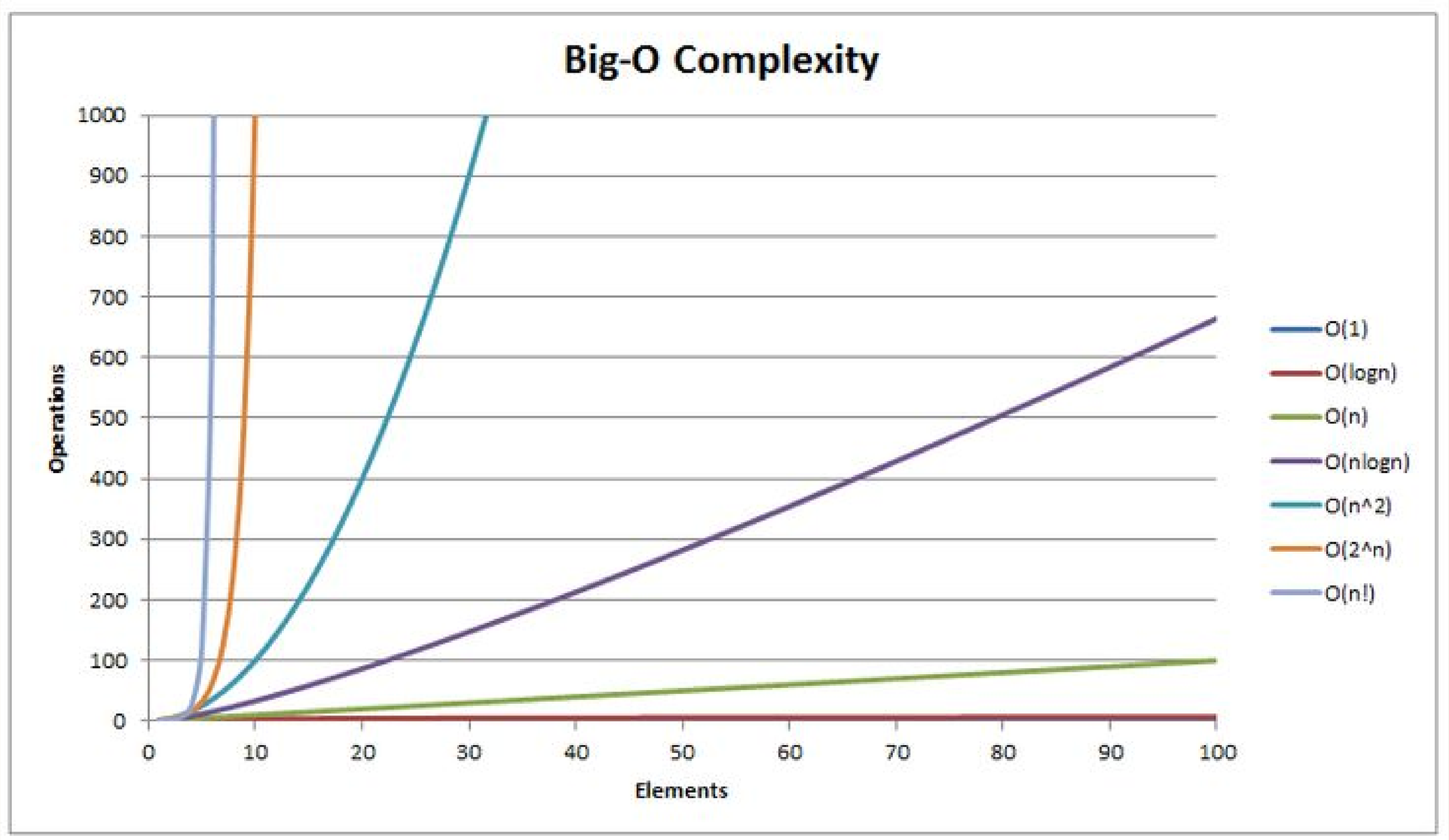
# Алгоритмы

03

**Алгоритм —** это конечная последовательность инструкций или шагов, которые выполняются для решения задачи или достижения определённой цели.

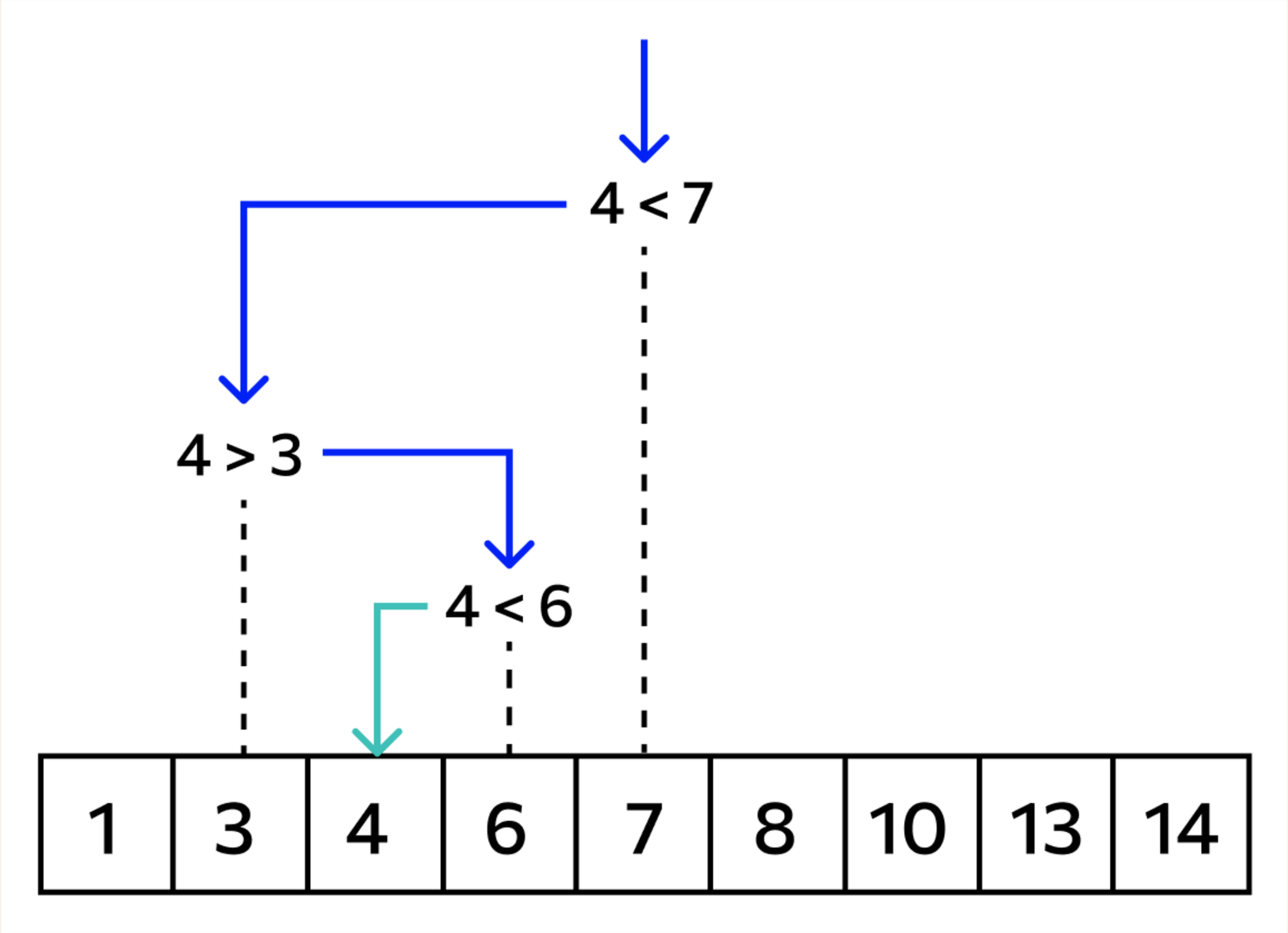
**Сложность алгоритмов** — это характеристика, которая определяет, сколько ресурсов (времени и памяти) требуется для выполнения алгоритма в зависимости от размера входных данных.

- O(1) — константная сложность (время выполнения не зависит от размера данных).
- O(n)— линейная сложность (время выполнения растёт линейно с увеличением размера данных).
- O(n^2)— квадратичная сложность (время выполнения пропорционально квадрату размера данных).


Big-O Complexity
Operations
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
Elements
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

**Бинарный поиск** — процесс нахождения индекса элемента с целевым значением в отсортированном массиве путем его дробления на половину на каждом шаге новой итерации.

1. Сортируем массив данных.
2. Делим его пополам и находим середину.
3. Сравниваем срединный элемент с заданным искомым элементом.
4. Если искомое число больше среднего — продолжаем поиск в правой части массива повторяя п3, иначе в левой




Бинарный поиск:

Сложность **O(log N)**

Простой перебор:

Сложность O(N)

## Применение

**Поиск по массиву:**

- Стандартно – найти элемент в массиве или индексы левого крайнего и правого вхождений
- Поиск локального максимума – двигаемся в сторону возрастания, ибо там в любом случае максимум: либо пик, либо край массива

**Поиск по ответу:**

- Можно поиском перебирать прямую вещественных чисел, например искать приближенное значение
- Можно перебирать поиском тоже некий диапазон чисел, среди которых (если знать какое условие проверять) найти ответ к задаче

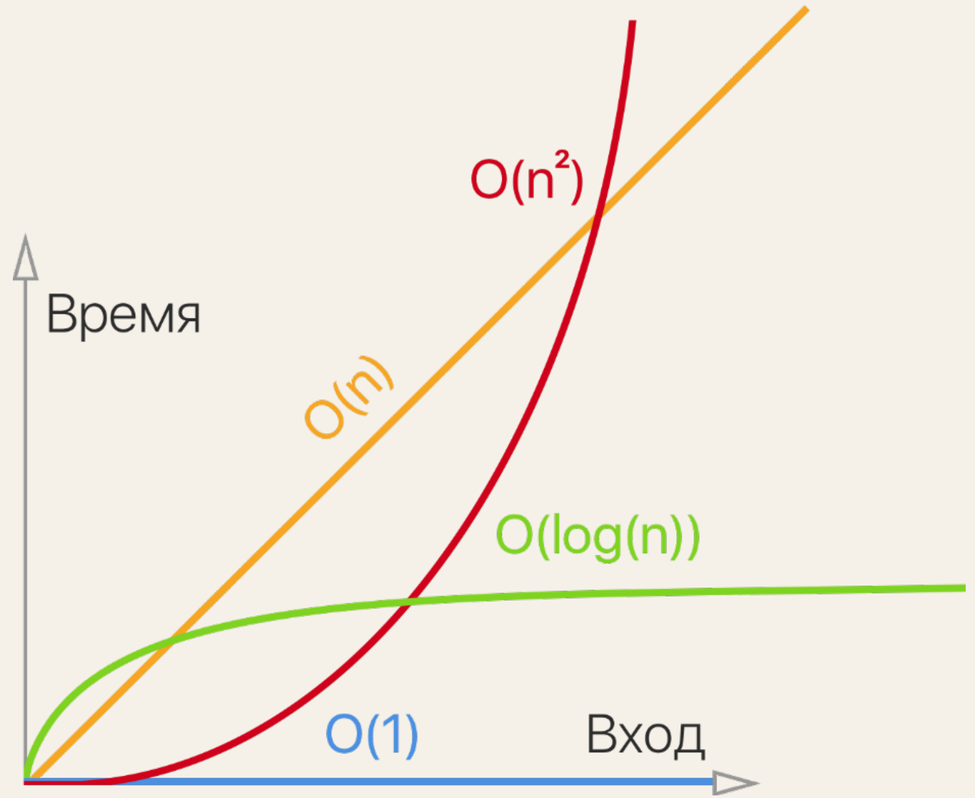
**Метод двух указателей** — это алгоритмический подход, в котором используются два индекса (указателя) для решения задач на массивах или строках. Эти указатели перемещаются по структуре данных **с разной скоростью или в разных направлениях,** что позволяет эффективно искать, сравнивать или преобразовывать данные.

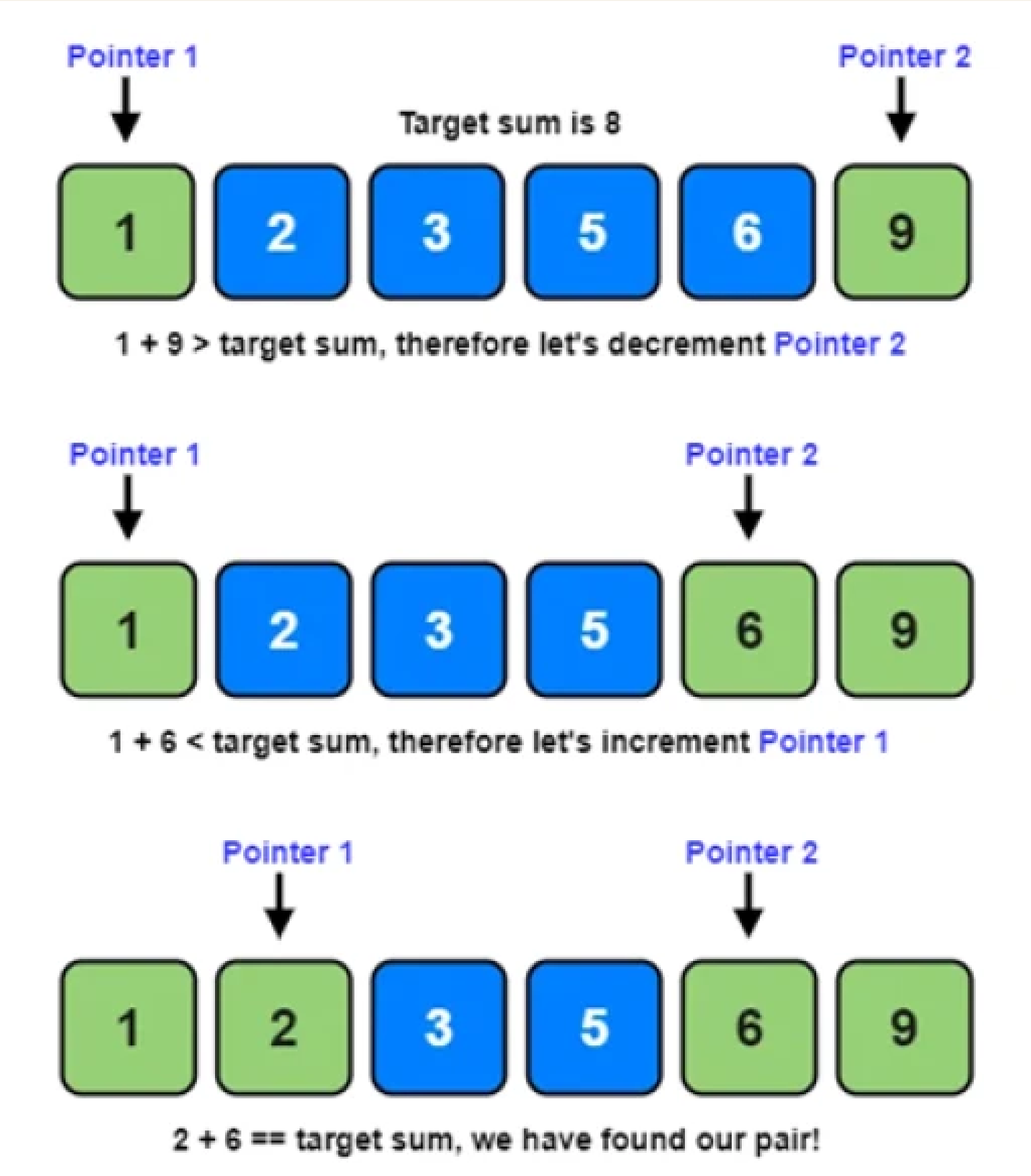
Метод двух указателей:

O(N) – достаточно один раз пройтись

Простой перебор:

O(N^2) – чтобы перебрать все пары






- Ставите два указателя
- Рассчитываете какое-то условие
- Согласно условию, двигаете какой-либо указатель

## Применение

**Идеи:**

- Указатели ставятся на начало и конец массива и движутся к центру
- Указатели ставятся на один конец и движутся с разной скоростью
- Указатели ставятся в один конец, но в разные последовательности

**Примеры:**

- Поиск пары элементов с заданной суммой в отсортированном массиве.
- Проверка, является ли строка палиндромом.
- Слияние отсортированных массивов.

# Домашнее задание

# Контест ДЗ4

**Три задачи на бинарный поиск**
**Три задачи на метод двух указателей**