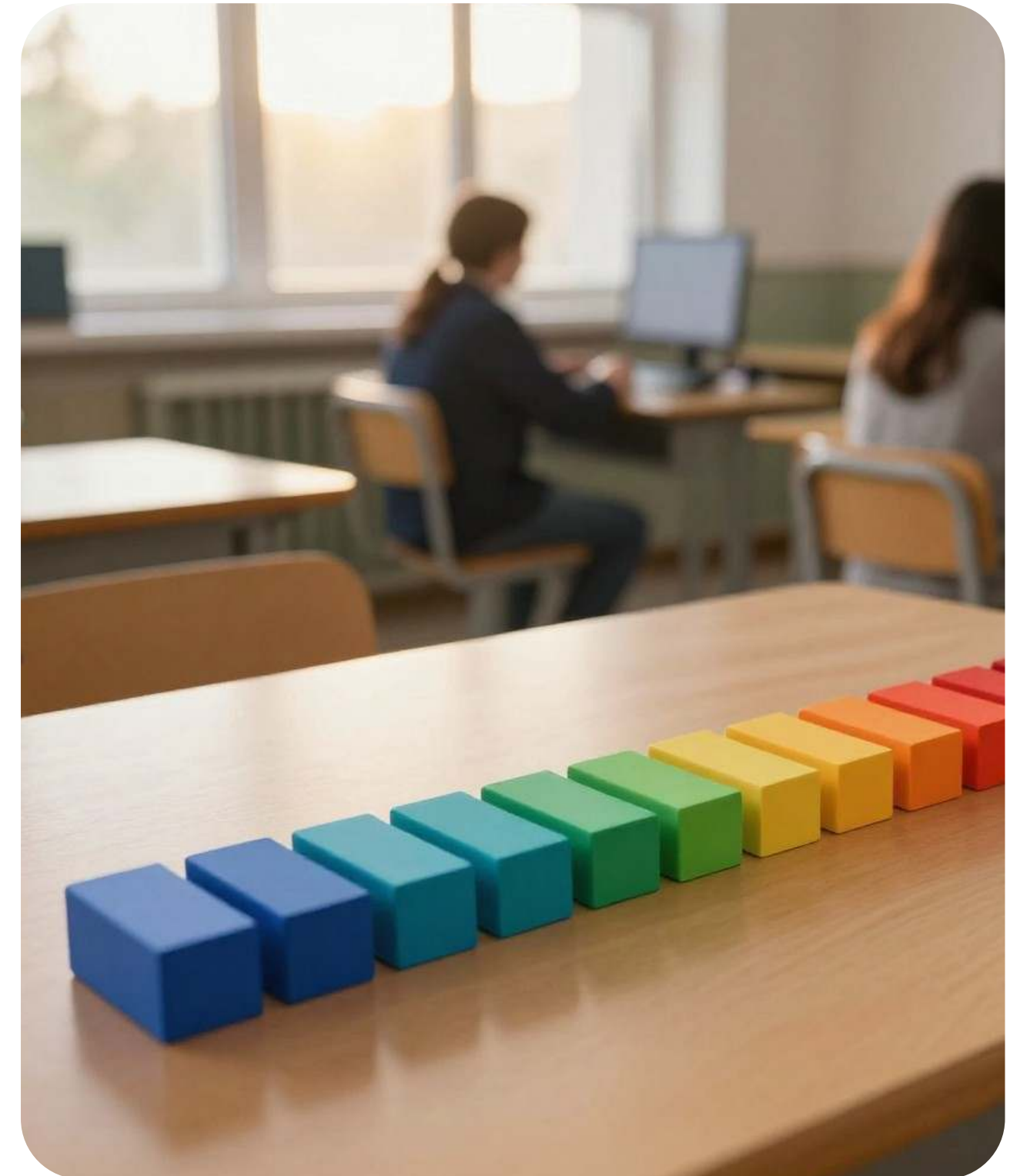




Массивы в информатике: базовая структура данных для упорядоченной информации

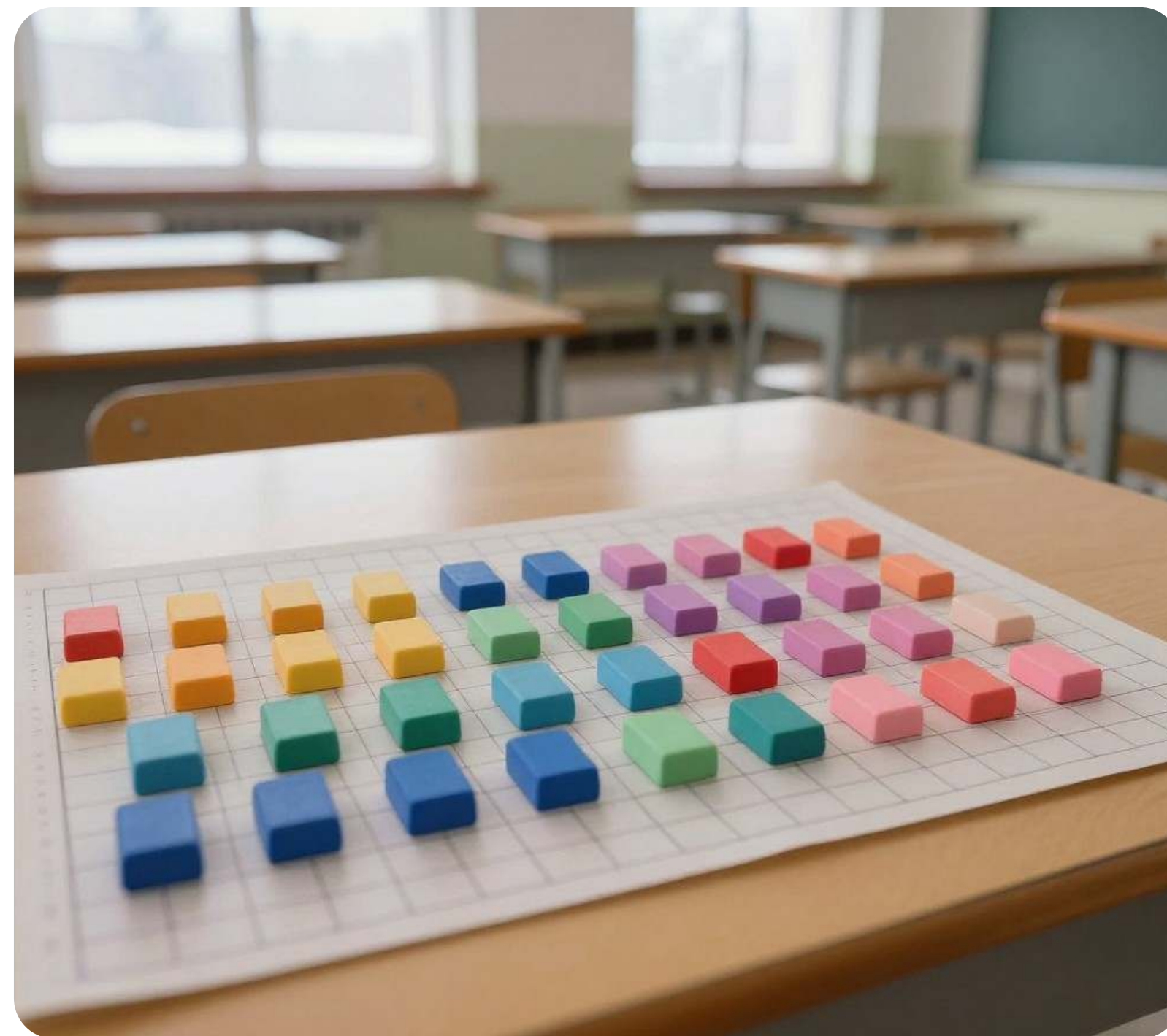
Массив помогает хранить упорядоченные данные одного типа и быстро обращаться к любому элементу по номеру.



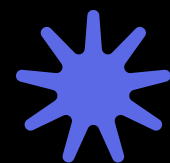
Где массивы встречаются в задачах и жизни



Они нужны везде, где есть последовательности: оценки, температуры по дням, координаты точек, измерения и таблицы значений. В 10 классе важны индексирование с нуля, длина массива и связь с циклами, потому что решения почти всегда — это последовательный проход.

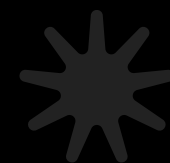


Определение массива и его основные свойства



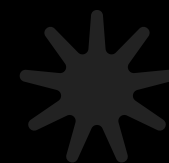
Массив хранит несколько значений одного типа под одним именем. Каждый элемент имеет свой номер — индекс, и доступ к нему выполняется напрямую: например, `a[0]`, `a[1]`, `a[2]`.

01



Размер массива обычно задаётся заранее, поэтому число элементов фиксировано. Это отличает массив от структур, где количество элементов может изменяться в процессе работы программы.

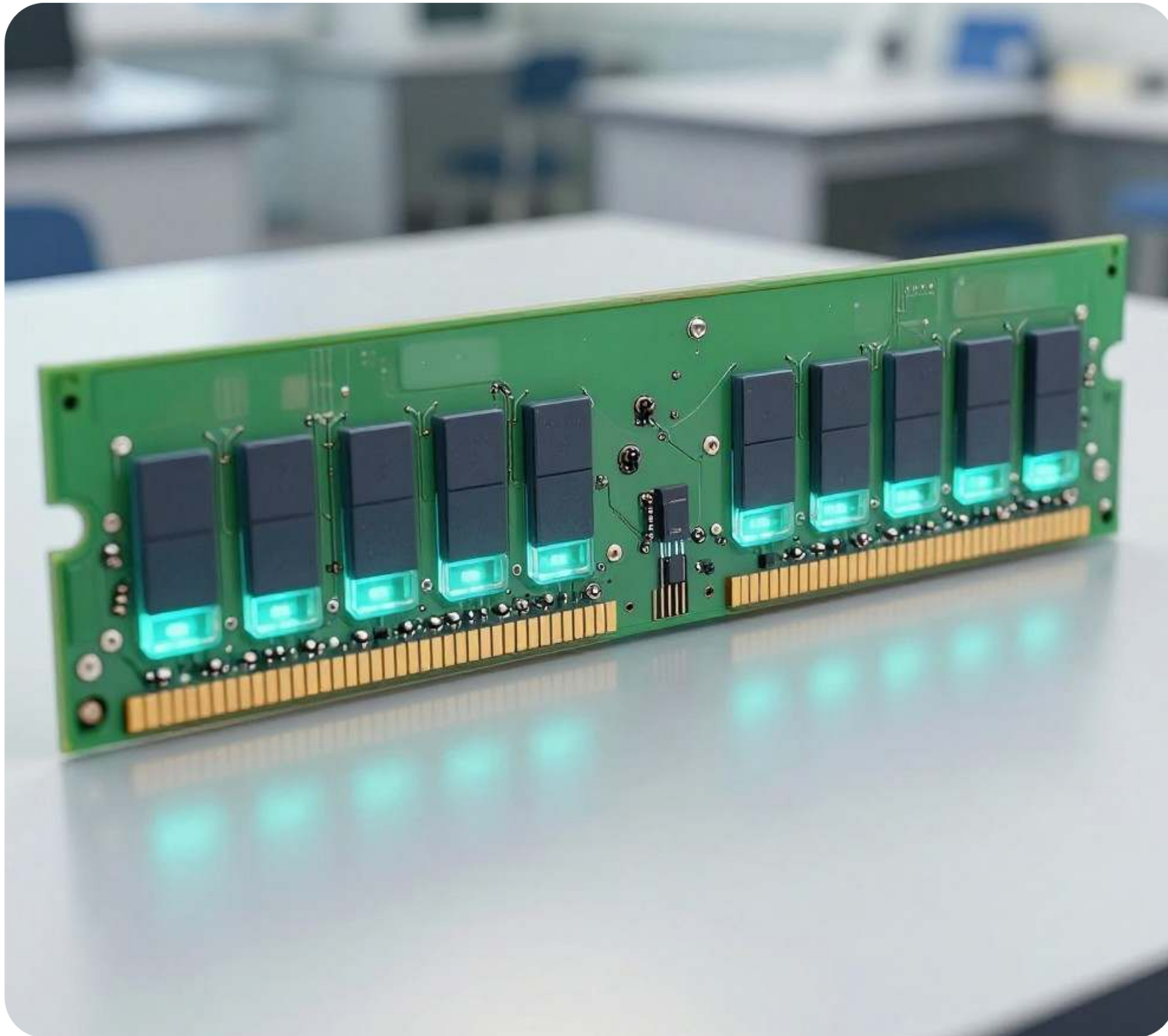
02



Главные свойства массива — упорядоченность элементов, одинаковый тип данных во всех ячейках, индексный доступ и фиксированная длина. Благодаря этому массив удобен для систематической обработки данных.

03

Как массив выглядит в памяти



01

Последовательные ячейки

В памяти массив занимает последовательный участок, который можно представить как цепочку соседних ячеек. Первый элемент лежит в начальном адресе, а следующий — рядом, в порядке следования индексов.

02

Почему индекс работает быстро

Индекс позволяет быстро вычислить смещение до нужного элемента: чтобы обратиться к $a[k]$, не нужно искать его «вручную». Поэтому обращение по номеру происходит очень быстро, а массив хорошо подходит для упорядоченной информации.

Массив, список и строка: сравнение основных признаков

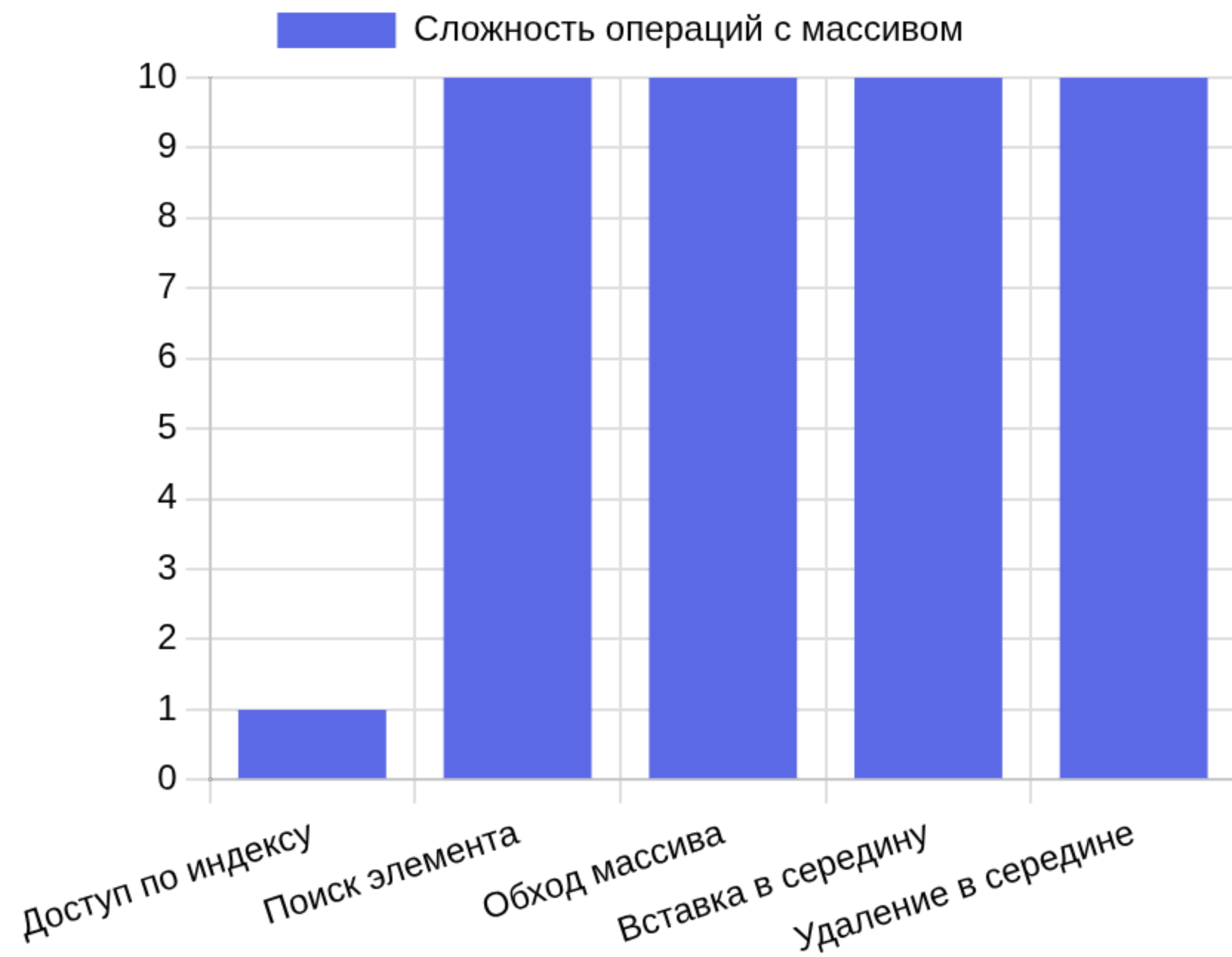
Признак	Массив	Список	Строка
Тип элементов	Один тип	Обычно один тип	Символы
Доступ	По индексу	По ссылкам	По позиции
Длина	Фиксирована	Меняется	Фиксируется структурой
Вставка/удаление	Медленнее в середине	Удобнее в середине	Обычно через создание новой
Типичное применение	Последовательные данные	Частые правки	Текст и шаблоны

Обобщение свойств структур данных в учебном курсе информатики

Выбор структуры зависит от того, какие операции вы выполняете чаще: доступ по номеру, вставки и удаления или работу с последовательностью символов.

Массив выбирают, когда нужен частый быстрый доступ по индексу и последовательная обработка. Список удобнее при частых вставках и удалениях. Строка — частный случай работы с упорядоченной последовательностью символов.

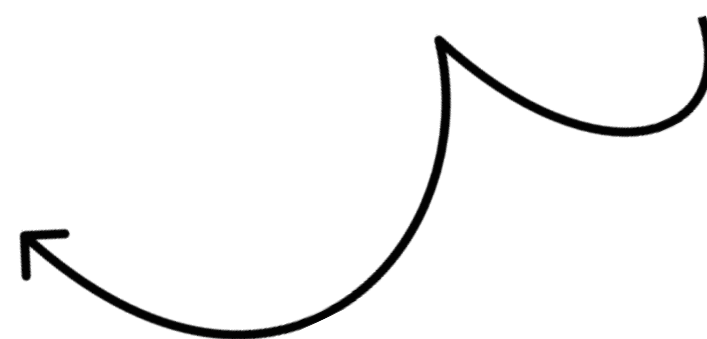
Сложность основных операций с массивом



Быстрый индексный доступ — сильная сторона массива. Но любые вставки или удаления часто требуют сдвига элементов, поэтому они становятся заметно «дороже».

Если задача сводится к многократному чтению элементов по номеру и последовательному проходу, массив эффективен. Когда же нужно много изменений внутри, выигрывают другие структуры.

Учебная оценка асимптотической сложности: $O(1)$ и $O(n)$.



Создание и инициализация массива



Задайте тип и длину

Сначала определяют тип элементов и количество ячеек. Массив целых, вещественных или символов различается по типу данных, а длина задаёт, сколько элементов будет храниться.



Инициализация начальными значениями

Можно создать массив сразу с заданными значениями. Тогда все элементы получают свои данные на старте, что удобно для моделирования сценариев и для последующих вычислений.

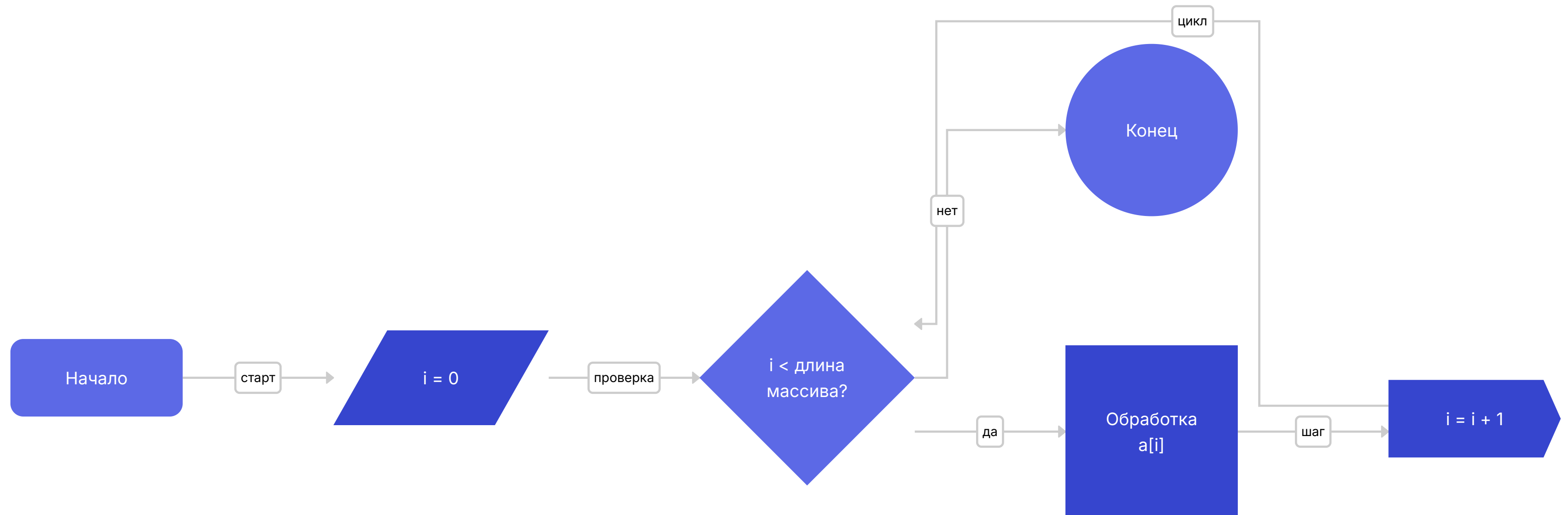


Смысл один, синтаксис разный

В разных языках программирования запись может отличаться, но логика одинакова: сначала создают структуру массива, затем заполняют её данными. Это помогает избежать ошибок с индексацией.

Алгоритм обхода массива по индексам

Последовательный просмотр элементов с использованием длины массива как границы



Поиск элементов в массиве

01

Линейный поиск просматривает элементы по порядку и подходит для любого массива, даже если он не отсортирован. В худшем случае нужно проверить все ячейки, поэтому сложность составляет $O(n)$.

02

Если массив уже отсортирован, появляется возможность использовать бинарный поиск. Он многократно сужает диапазон поиска, поэтому работает за $O(\log n)$ и обычно заметно быстрее при больших объёмах данных.

03

Выбор метода зависит от размера массива и количества запросов: если данных много и поиск выполняется часто, выгодно заранее подготовить упорядоченность. Если же важна простота и сортировка не нужна, берут линейный поиск.

Примеры типовых операций с массивами



Операция	Идея	Типичная сложность
Сумма элементов	Складываем каждый $a[i]$	$O(n)$
Максимум	Запоминаем наибольшее $a[i]$	$O(n)$
Чётные числа	Считаем элементы по условию	$O(n)$
Минусы $\rightarrow$ нули	Меняем $a[i]$, если $a[i] < 0$	$O(n)$
Копирование массива	Переписываем элементы в новый массив	$O(n)$

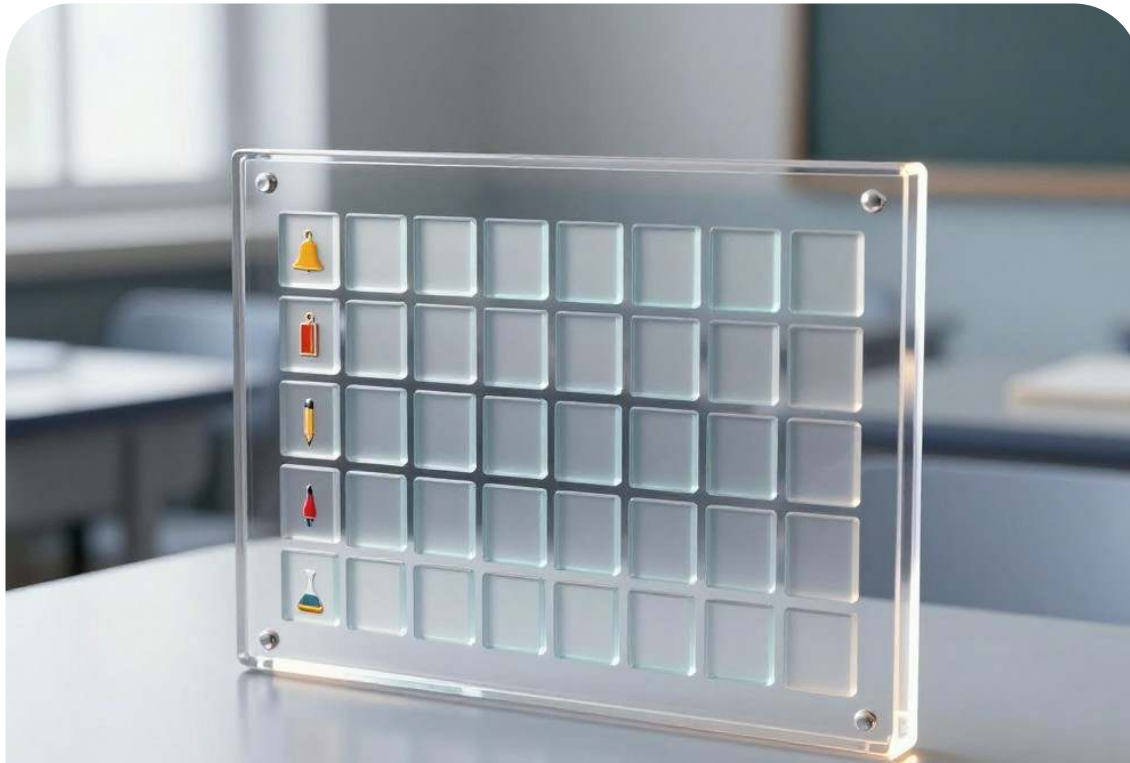
Большинство операций в учебных задачах выполняется через цикл по индексам, потому что требуется последовательно обработать элементы массива и накопить результат.

В большинстве задач требуется один проход по массиву: доступ к элементам по индексу делает решения прямыми и понятными. Поэтому цикл и индекс — центральные инструменты при работе с массивами.

Практические алгоритмы из школьного курса информатики



Двумерный массив как таблица значений



Классное расписание: строки — дни, столбцы — уроки

Представьте школьное расписание: каждый день — это строка, каждый урок — столбец. Элемент `al[i][j]` отвечает за конкретное событие в ячейке «день i — урок j ». Такую структуру удобно заполнять и обновлять, не теряя соответствие между местом и данными.



Таблица оценок: каждая ячейка хранит результат

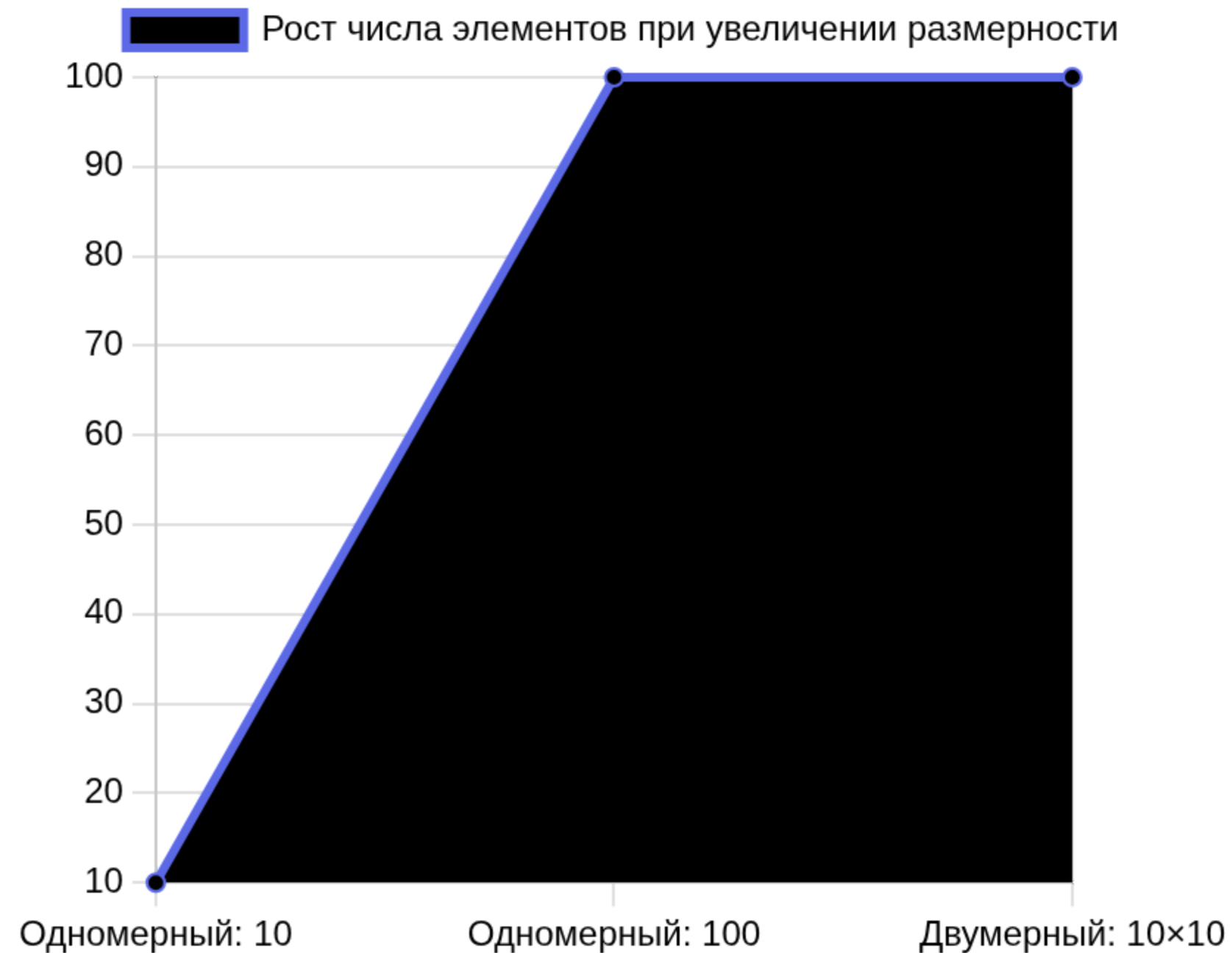
В ведомости видно сразу и ученика, и предмет: строка задаёт ученика, столбец — дисциплину. Двумерный массив позволяет хранить оценки как отдельные значения в своих координатах. Сравнить и подсчитать результаты проще, потому что данные всегда лежат в предсказуемых позициях.



Игровое поле и матрица символов: точка имеет адрес

В игре клетки важны не меньше персонажей. Двумерный массив помогает описывать поле так, будто у каждой клетки есть свой «адрес» `al[i][j]`. Это удобно для отметок, например стены, пустоты или фишки: каждое состояние хранится в соответствующей ячейке.

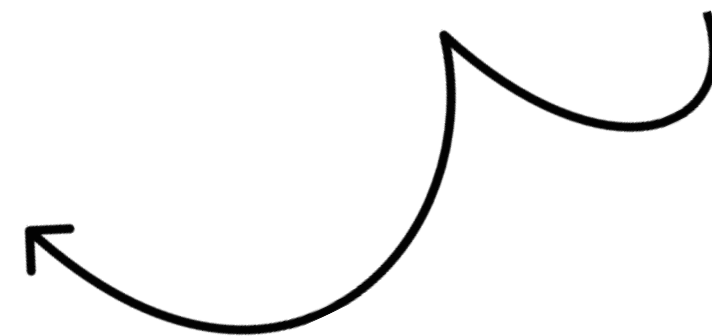
Как растёт объём данных при увеличении размерности



Переход к двумерным структурам часто ускоряет «размножение» ячеек: важны не только размеры, но и то, сколько измерений вы добавляете.

График показывает: при увеличении размеров массива количество хранимых элементов растёт пропорционально числу ячеек. Значит, заранее оценивать объём данных обязательно.

Суммарное число элементов для примеров: 10, 100, 10x10 (=100),
2026-05-13



Типичные ошибки при работе с индексами



01

Самая частая проблема — выход за границы: обращение по индексу меньше нуля или больше последнего допустимого значения. Даже одна лишняя проверка может привести к сбою программы.

02

Путаница между длиной массива и последним индексом: при длине 10 последний индекс равен 9. Если использовать условие $i \leq n$, то последний шаг станет ошибочным.

03

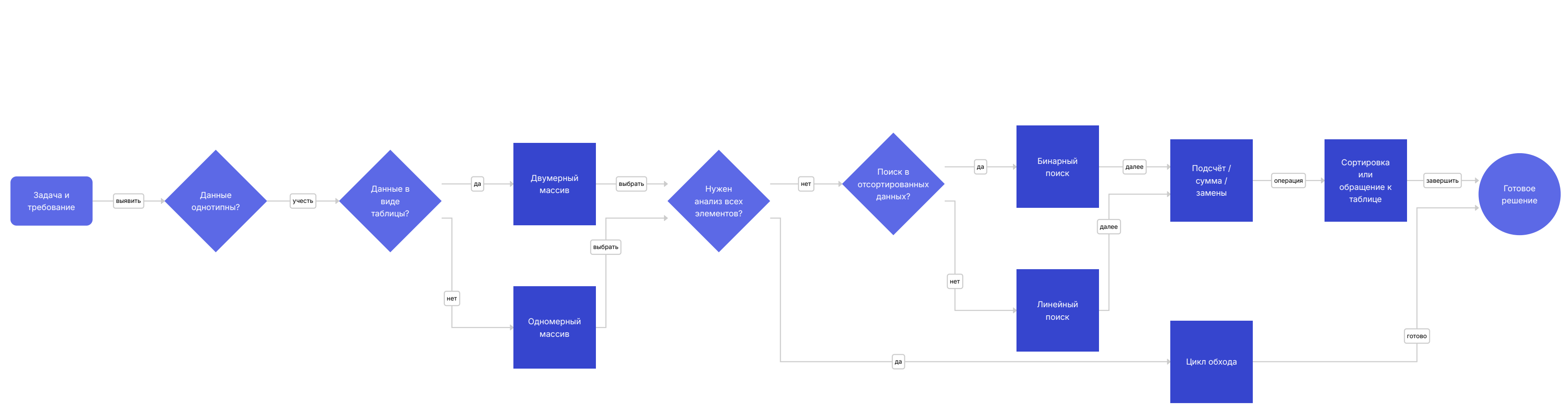
В двумерных массивах легко перепутать строки и столбцы: использовать `a[j][i]` вместо `a[i][j]`. Визуально данные «кажутся на месте», но расчёты дают неверные ответы.

04

Ошибки в вычислениях индексов проявляются не сразу: программа может работать, но результат будет неправильным. Надёжный способ — аккуратные границы цикла и проверка условий внутри него.

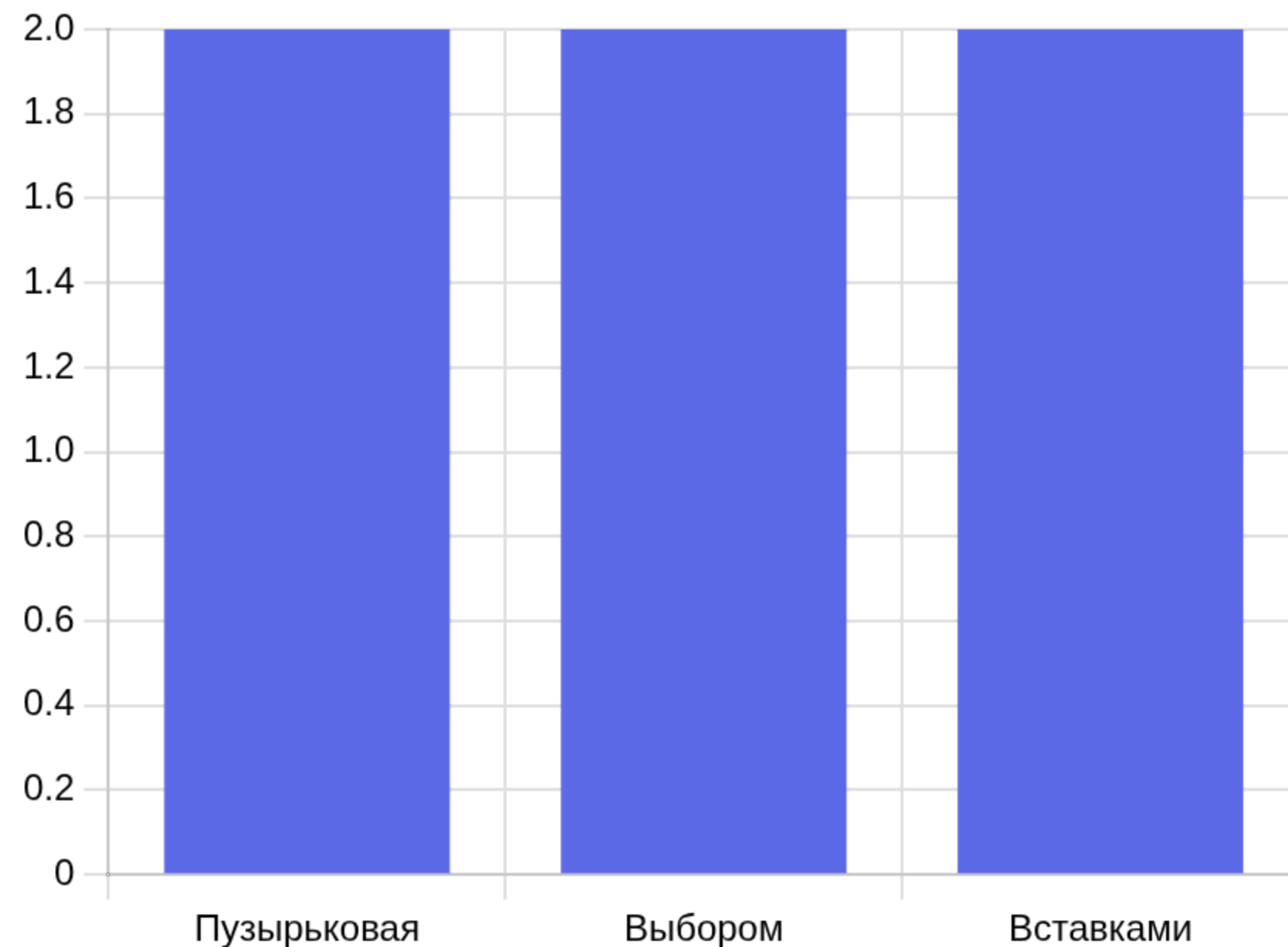
Выбор подхода к задаче с массивом

Связь между формулировкой задачи и структурой/операцией для решения



Сортировка массива и её значение для обработки данных

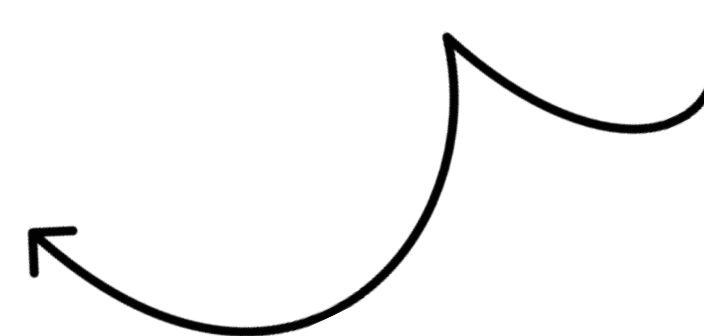
Сравнение простых сортировок на уровне школьных реализаций (число сравнений)



После упорядочивания многие задачи решаются быстрее: например, легче находить максимум, проверять диапазоны и применять эффективный поиск.

Для небольших массивов сортировки на сравнениях удобны и понятны, хотя в худшем случае дают квадратичную сложность. Их польза — в последующей обработке данных.

Обобщённые учебные оценки сложности трёх сортировок в простейших реализациях; без привязки к конкретному языку, 2026-05-13

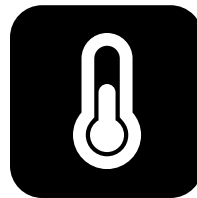


Массивы в реальных информационных задачах



Оценки и статистика

В классных журналах массивы удобно использовать для хранения оценок по предметам и вычисления простых итогов. Данные одного типа системно лежат в ячейках, что упрощает подсчёты.



Температуры и измерения

Список температур за месяц — типичный одномерный случай. Хранение значений в массиве позволяет легко обходить их циклом и искать максимум, минимум или дни с аномалиями.



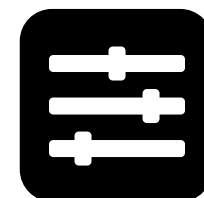
Координаты точек и траектории

Координаты точек на плоскости можно раскладывать по массивам, а для сетки использовать двумерные структуры. Тогда «позиция» каждой точки становится однозначной и проверяемой.



Результаты соревнований

Номера мест и баллы участников удобно представлять массивами: так проще сопоставлять участников и результаты, а затем выполнять сортировку или последовательный анализ.



Пиксели и отсчёты сигнала

В обработке изображений двумерные массивы хранят значения пикселей, а в звуке — отсчёты сигнала во времени. Это показывает, что массивы — базовая модель для данных в приложениях.

Когда массив подходит лучше всего



Тип задачи	Почему удобно	Ограничение
Последовательность значений	Быстрый индексный доступ	Размер нужно учитывать заранее
Подсчёты и обход	Цикл проходит элементы последовательно	Удаление/вставка в середину дороги
Табличные данные	Координаты ячеек понятны	Нужно строго следить за индексами
Повторяющиеся запросы	Можно переиспользовать данные	При частых изменениях структура теряет эффективность

Учебная логика по свойствам массивов; 2026-05-13

Массив эффективен там, где важны однотипность данных, частый доступ по номеру и последовательная обработка. Его слабые стороны проявляются при постоянных изменениях структуры.

Выбор массива оправдан, когда данные стабильны и операции чаще связаны с чтением/перебором, чем с частыми вставками и удалениями.

Массивы как основа других структур данных

Дальше: динамические массивы

Если размер нужно менять, используют динамические массивы. Принцип адресации сохраняется, но добавляется управление увеличением ёмкости и перераспределением.

Таблицы и матрицы

Матрицы расширяют двумерную идею массива: координаты определяют место элемента. Это напрямую связывает массивы с обработкой таблиц и более сложными алгоритмами анализа данных.

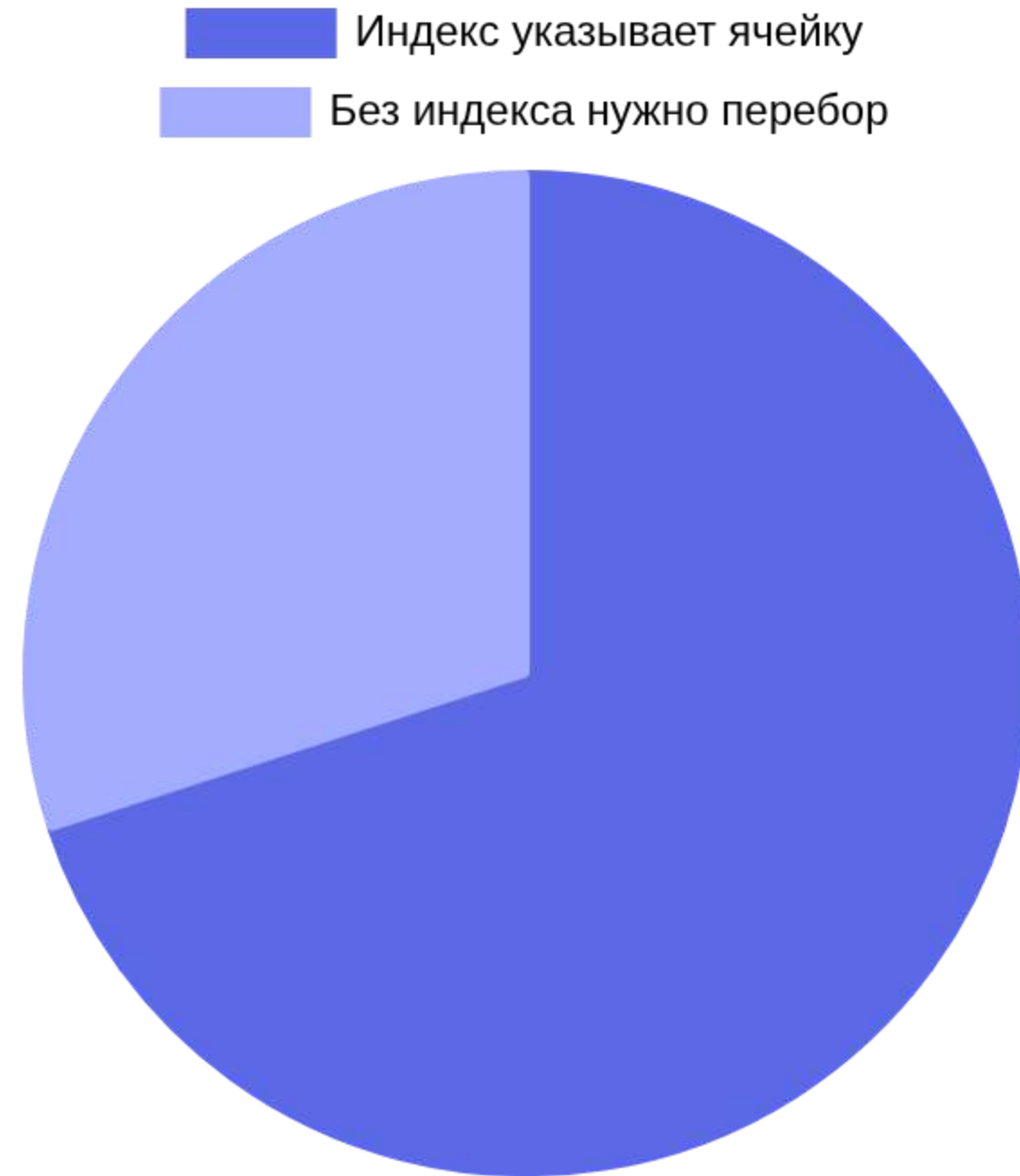
Базовый уровень: фиксированный блок данных

Массив учит мыслить адресацией по индексу: как хранится последовательность и как к ней обращаться. Это фундамент для понимания памяти и организации данных.

Стек и очередь: доступ по правилам

Стек и очередь строятся на механизмах размещения элементов в последовательности. Понимание индексов помогает правильно реализовать операции «последним вошёл — первым вышел» и «первым вошёл — первым вышел».

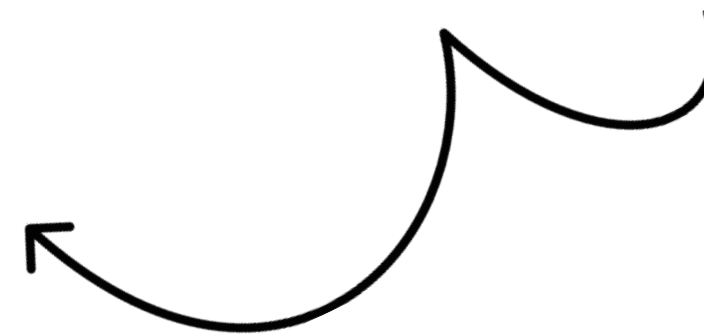
Индексация и быстрый доступ к элементам



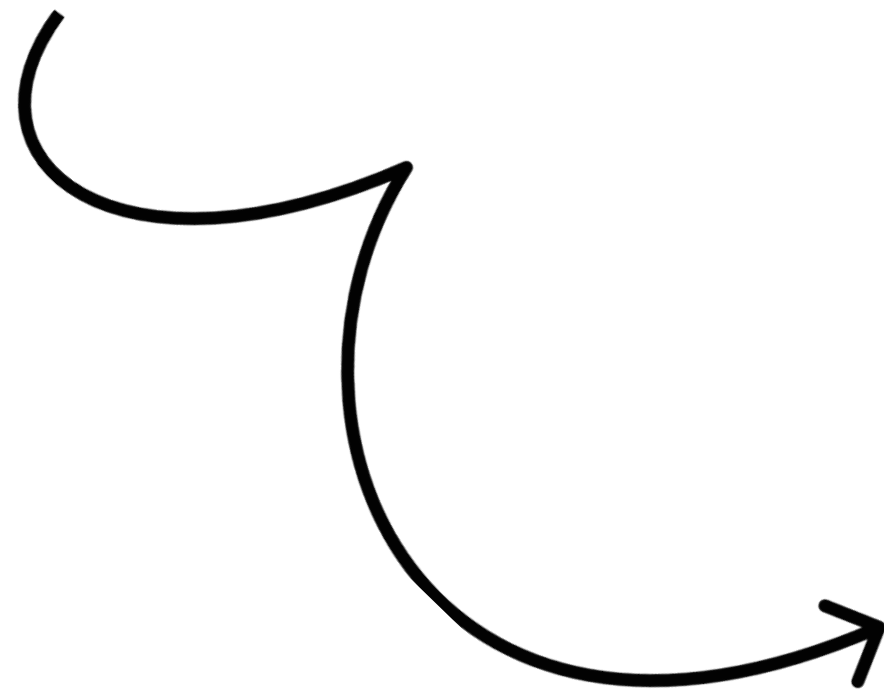
Когда индекс задан, программа сразу находит нужную ячейку, не возвращаясь к началу. Поэтому обращение похоже на «адресный» шаг.

Диаграмма подчёркивает: массив выигрывает из-за прямой адресации. В структурах без индексного доступа приходится тратить время на поиск.

Схема-иллюстрация соотношений на уровне объяснения (без привязки к конкретным замерам), 2026-05-13



Массивы: кратко о главном



Массив — базовая структура для хранения упорядоченной однотипной информации. Его сильные стороны — индексный доступ, фиксированная длина и удобство последовательной обработки циклом. Ограничения — сложность вставки и удаления в середине и необходимость заранее учитывать размер.

