АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Алгоритмизация

1. Понятие, свойства, логика алгоритма
2. Структуры данных
3. Эффективность алгоритма

1

Понятие, свойства, логика алгоритма

Понятие алгоритма

«**Алгоритм** — это конечный набор правил, который определяет последовательность операций для решения конкретного множества задач и обладает пятью важными чертами: конечность, определенность, ввод, вывод, эффективность». *(**Д. Э. Кнут**)*

«**Алгоритм** — это точное предписание, определяющее вычислительный процесс, идущий от варьируемых исходных данных к искомому результату». *(**А. Марков**).*

Алгоритм — это однозначное описание последовательности выполнения действий из заданного набора, позволяющее получить требуемый результат за конечное число шагов.

Понятие алгоритма

Вспомогательные или **подчиненные алгоритмы** — это готовые алгоритмы, целиком включаемые в состав разрабатываемого алгоритма.

Слово «алгоритм» использовалось в математике для обозначения правил выполнения четырех арифметических действий: сложения, вычитания, умножения и деления.

Пример. Алгоритмы в областях человеческой деятельности
Алгоритм управления производственным процессом.
Алгоритм поиска пути в лабиринте и т. п.

Алгоритм указывает последовательность действий по переработке исходных данных в результаты.

Свойства алгоритма

Массовость. Один и тот же алгоритм применим для работы на разных наборах исходных данных.

Дискретность. Алгоритм представлен в виде конечной последовательности шагов, т. е. имеет дискретную структуру.

Конечность. Выполнение алгоритма заканчивается после выполнения конечного числа шагов.

Определенность. При повторениях алгоритма для одних и тех же исходных данных всегда получается одинаковый результат.

Эффективность. Действия исполнителя на каждом шаге можно выполнить точно и за разумное конечное время.

Результативность. Алгоритм должен завершаться определенными результатами.

Управляющие конструкции языков программирования

Линейная последовательность действий

Условная конструкция (если-то-иначе)

Конструкция повторения (цикл)

Переход

Описание управляющих конструкций языка (операторов) удобнее всего делать с помощью блок-схем.

Управляющие конструкции языка являются структурированными, если эквивалентная им блок-схема имеет один вход и один выход.

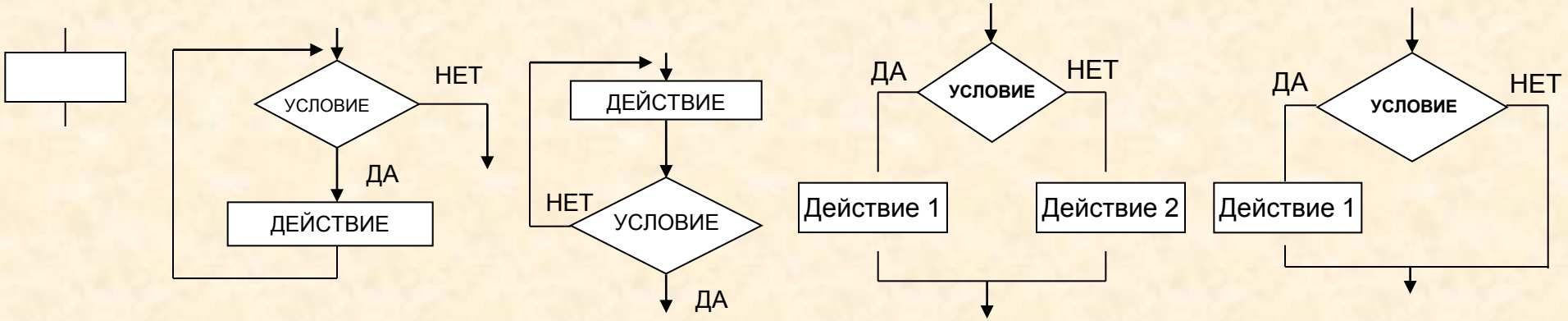
Блок-схема, состоящая из структурированных конструкций, также является структурированной.

Методы проектирования алгоритмов

Нисходящее проектирование

Модульное программирование

Структурное программирование



Логика алгоритма

СТРУКТУРИРОВАННАЯ

- последовательность
- выбор (ветвление)
- повторение (цикл)

АРХИТЕКТУРНО-ОРИЕНТИРОВАННАЯ

- действие
- условие
- переход

РЕКУРСИВНАЯ

- последовательность
- выбор (ветвление)
- рекурсия

2

Структуры данных

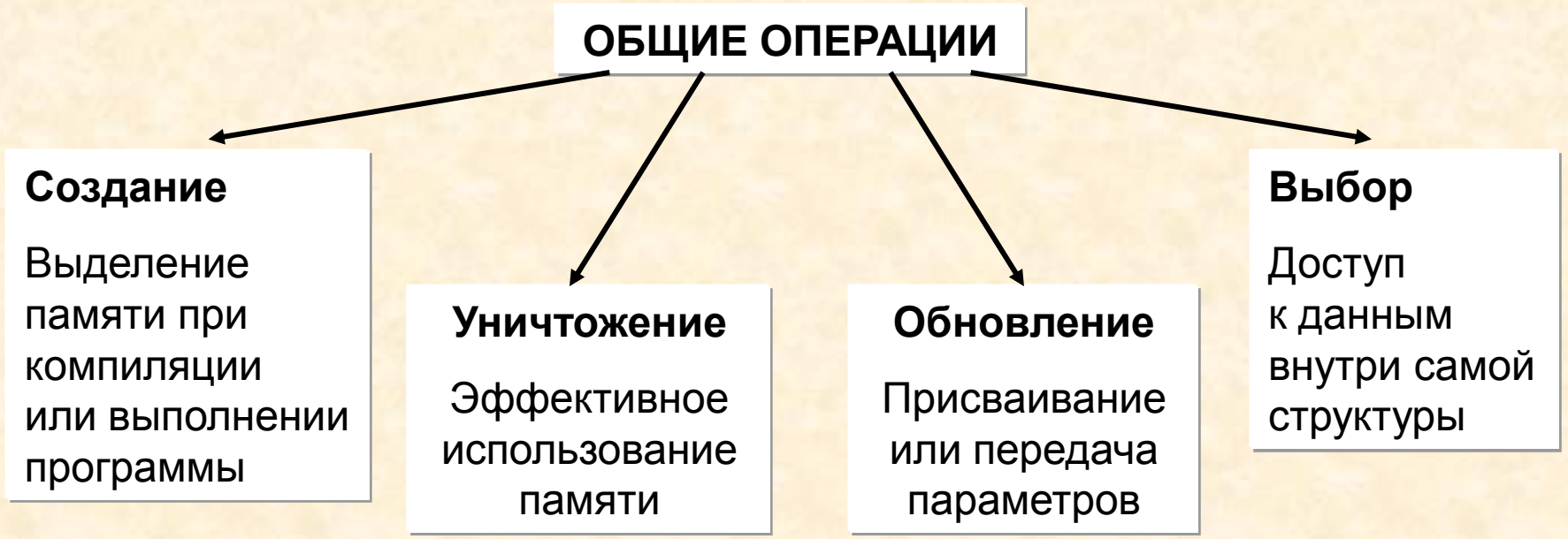
Классификация структур данных

Под структурой данных в общем случае понимают
множество элементов данных и множество связей между ними.

Простые базовые	**Статические**	**Полустатические**	**Динамические**
Числовые	Вектор	Стеки	Списки
Символьные	Массивы	Очереди	Графы
Логические	Множества	Деки	Деревья
Указатели	Записи	Строки	
	Таблицы		

Файловые структуры
соответствуют структурам данных для внешней памяти.

Операции над структурами данных



Структуры данных
Массив

Во всех современных языках программирования, кроме ***Фортрана***, элементы массива располагаются по строкам.

Пример

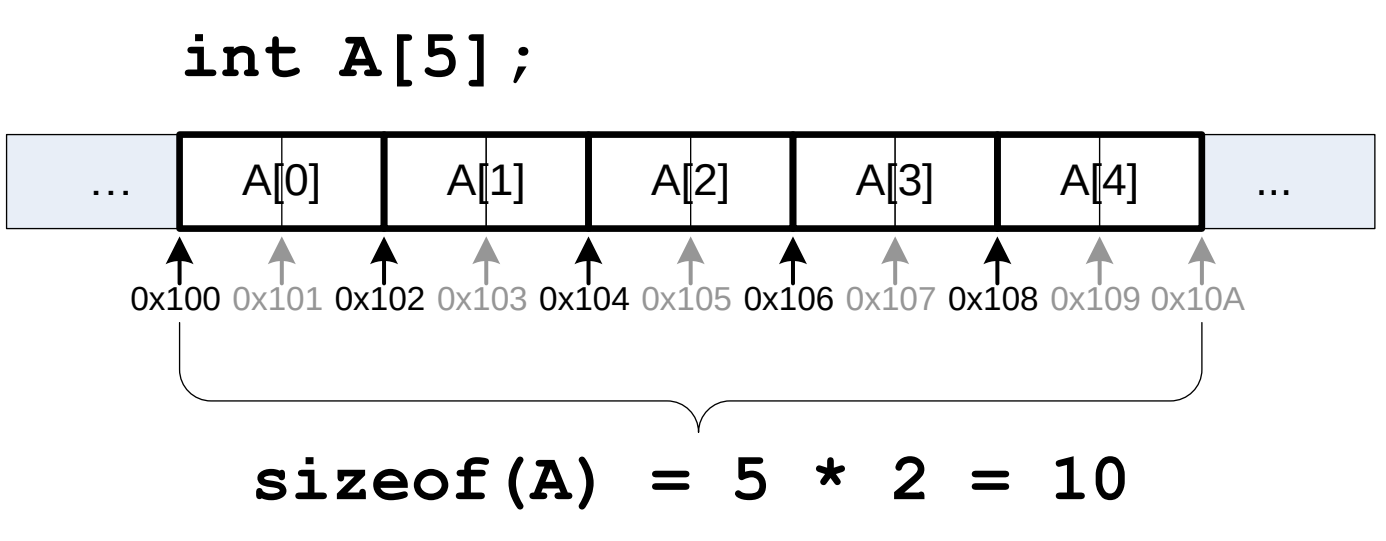
float X[2][3] = {{1., 3.},{6.}};

Элементы **X[1][2], X[2][1], X[2][2]** будут нулевыми.

Расположение массива в памяти

X[0][0] X[0][1] X[0][2] X[1][0] X[1][1] X[1][2]

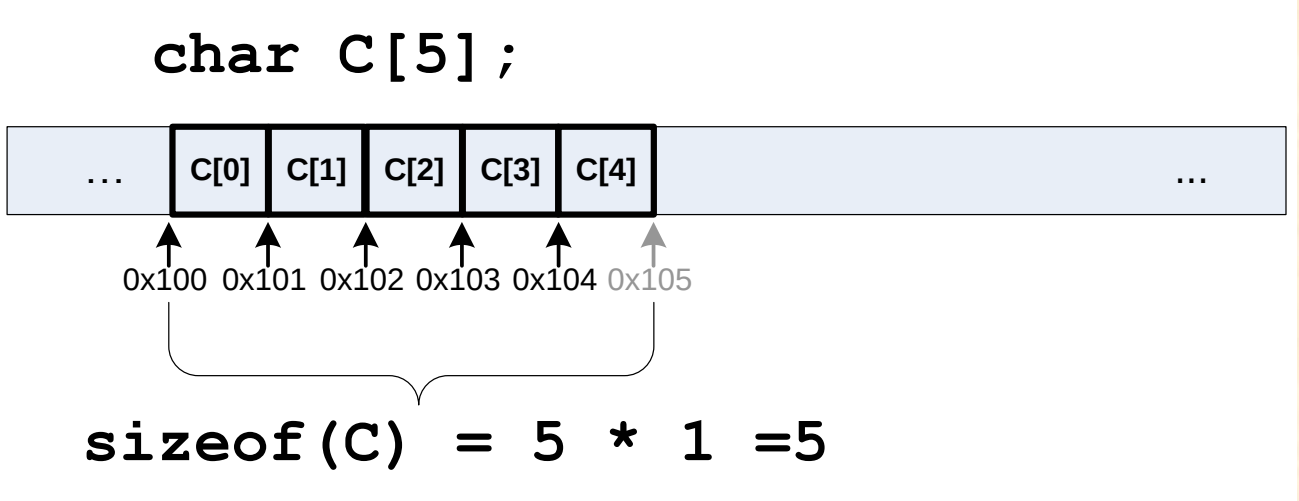
Структуры данных
Порядок хранения в памяти элементов массива



Общее правило большинства алгоритмических языков

Элементы многомерных массивов хранятся в памяти в последовательности, соответствующей более частому изменению младших индексов.

Структуры данных
Массив



Данные для вычисления реального адреса элемента массива в памяти

Базовый адрес массива (адрес начала массива)

Тип элементов массива

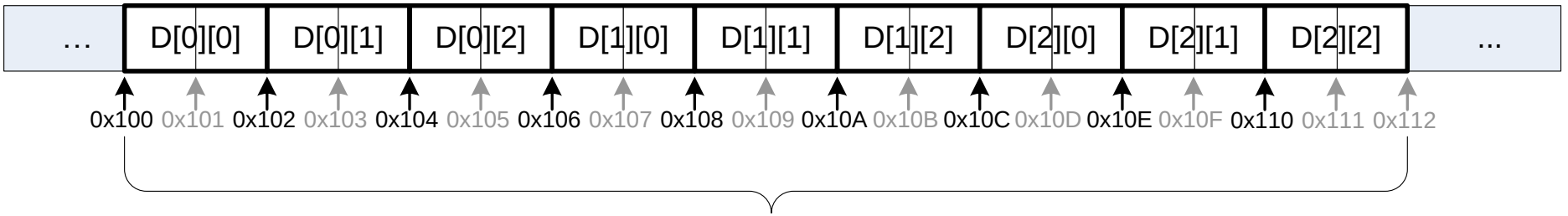
Номер элемента массива

char C[5];

Addr(C[i]) = Addr(C) + i*sizeof(char);

Структуры данных
Массив

```
int D[3][2];
```




```
sizeof(D) = 3 * 2 * 2 = 12
```

int D[R][C];

Addr(D[j][i]) = Addr(D) + (j * C + i) * sizeof(int);

(j * C + i) — порядковый номер элемента в памяти при обходе массива.

3

Эффективность алгоритма

Анализ сложности алгоритма

Временная трудоемкость

Емкостная трудоемкость

Критерии оценки временной трудоемкости
Поведение вычислительного процесса
Скорость реализации алгоритма
Фиксированный набор инструкций

Моделирование "худших" случаев
Проверка поведения алгоритма на входных данных, принимающих граничные значения из разрешенного диапазона.
Тестирование алгоритма на максимально больших по объему входных наборах.

Временная трудоемкость алгоритма

Временная трудоемкость T(n) показывает количество операций, которые алгоритм выполняет при обработке входящих данных объема n.

Задача.
Временная трудоемкость процедуры сортировки данных оценивается как $T(n) = n^2$.
Во сколько раз увеличится время сортировки, если объем данных увеличить в 4 раза?

? $\frac{T(4n)}{T(n)} = 16$

O-нотации

g(n) = O(f(n))

O(f(n)) — O-большое

O-большое *относится к дискретным функциям f(n) натурального n и ко всем функциям g(n), растущим не быстрее f(n).*

Существует такая пара положительных значений M и n_0, что

$|g(n)| \leq M|f(n_0)|$ (для $n \geq n_0$)

o(f(n)) — o-малое

o-малое *относится к функциям, которые растут быстрее f(n)*

Квадратичная трудоемкость

Пример
Независимый выбор пары соседей для первой парты
в классе из n учеников можно осуществить n*(n - 1) способами.

Трудоемкость алгоритма оценивается как $O(n^2)$
(*квадратичная*)

Кубическая трудоемкость

Пример
Для набора из n попарно неравных отрезков
подсчитать количество всевозможных "троек",
из которых получаются невырожденные треугольники.

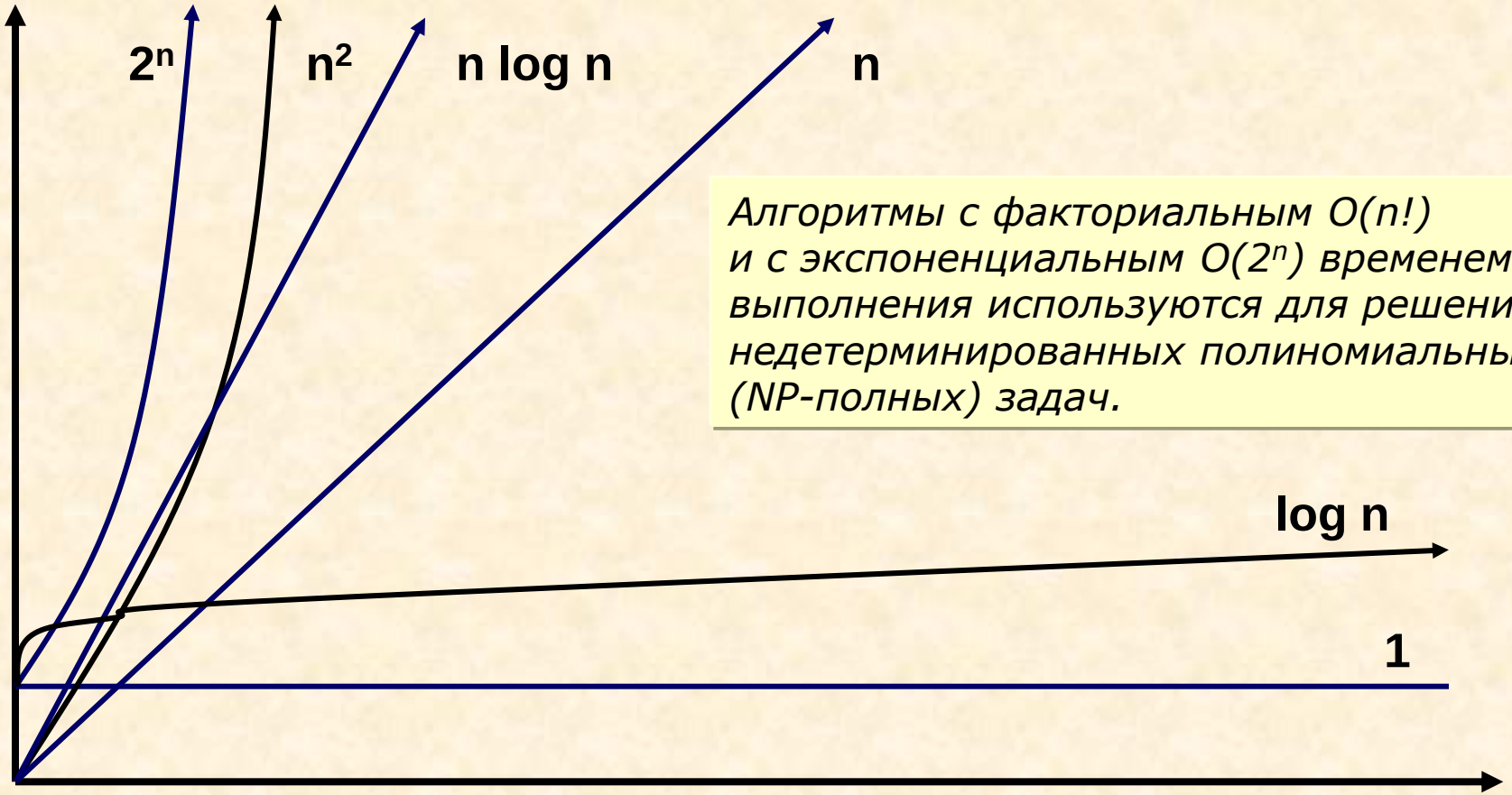
Трудоемкость алгоритма оценивается как $O(n^3)$
(кубическая)

Уточнение оценки

$n(n - 1)(n - 2) = n^3 - 3n^2 + 2n = n^3 + O(n^2)$

$n^3 - 3n^2 + 2n = n^3 - 3n^2 + O(n)$.

Временная трудоемкость алгоритма
Графики порядков роста



Алгоритмы с факториальным O(n!) и с экспоненциальным $O(2^n)$ временем выполнения используются для решения недетерминированных полиномиальных (NP-полных) задач.

Эффективность алгоритма
Рекурсивные алгоритмы

Нерекурсивное/рекурсивное разбиение — это деление задачи на множество идентичных задач меньшей размерности с последующим объединением результата.

Рекурсивные вызовы образуют древовидную структуру. Количество вершин в дереве определяет эффективность алгоритма.

**Эффективность (трудоемкость) таких алгоритмов,
зависит от затрат на разделение
и от пропорций разделяемых частей.**

Логарифмическая зависимость (лучший случай) — деление на равные части.

Линейная зависимость (худший случай) — выделение единственного элемента.

$N \ldots N \log N \ldots N^2$

Эффективность алгоритма
Жадные алгоритмы. Полный перебор

В основе жадного алгоритма лежит разделение, но в каждой точке выбирается **одна из подзадач.**

Принцип жадных алгоритмов

Движение «по линии наименьшего сопротивления» в каждой точке приведет к желаемому результату.

logN … N — *Диапазон трудоемкостей жадного алгоритма*

2^N … N^N … N! — *Диапазон трудоемкостей при полном переборе*