

# Структуры и алгоритмы обработки данных

## Литература

1. Алгоритмы: вводные понятия
2. Корректность алгоритма
3. Анализ эффективности алгоритма (начало)

# Литература по алгоритмизации:

1. **Вирт Н.** Алгоритмы и структуры данных. Новая версия для Оберона, 2010.
2. **Кнут Д.** Искусство программирования. Тома 1-4, 1976-2013.
3. **Бхаргава А.** Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих, 2017.
4. **Кормен Т.Х. и др.** Алгоритмы. Построение и анализ, 2013.
5. **Лафоре Р.** Структуры данных и алгоритмы в Java. 2-е изд., 2013.
6. **Макконнелл Дж.** Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., 2018.
7. **Скиена С.** Алгоритмы. Руководство по разработке, 2011.
8. **Хайнеман Д. и др.** Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2017.
9. **Гасфилд Д.** Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология, 2003.

# Литература по C++:

1. **Страуструп Б.** Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
2. **Павловская Т.А.** C/C++. Программирование на языке высокого уровня, 2003.
3. **Прата С.** Язык программирования C++. Лекции и упражнения. - 6-е изд., 2012.
4. **Седжвик Р.** Фундаментальные алгоритмы на C++, 2001-2002
5. **Хортон А.** Visual C++ 2010. Полный курс, 2011.
6. **Шилдт Г.** Полный справочник по C++. 4-е изд., 2006.

# Интернет-ресурсы (общего назначения):

1. **Национальный открытый университет «ИНТУИТ»** [Электронный ресурс]. URL: <https://www.intuit.ru/>
2. **Хабр** [Электронный ресурс]. URL: <https://habr.com/ru/> (
3. **MIT OpenCourseWare** [Электронный ресурс]. URL: <https://ocw.mit.edu>
4. Stepik, Coursera и пр.

# 1. Алгоритмы: вводные понятия

# Алгоритм (лат. algorithmi) –



- Это набор инструкций, описывающих порядок действий **исполнителя**, для достижения определённого результата  
(**неформальное** определение)
- Базисное понятие в математике:
- **Вычисления** (вычислительная задача) – это обработка числовой информации по определённому алгоритму. →



# Исполнитель —



- Это абстрактная или реальная (техническая или биологическая) **система**, способная выполнить действия, предписываемые алгоритмом
- **Неформальный** (знает конечную цель А.) и **формальный**

## Характеристики:

- **Среда** (обстановка) — место действия
- Система допустимых **команд**:
  - Должны быть заданы **условия применимости** (состояние среды)
  - Описаны **результаты** выполнения
- Набор **действий**
- **Отказы** (недопустимое для выполнения команды состояние среды).

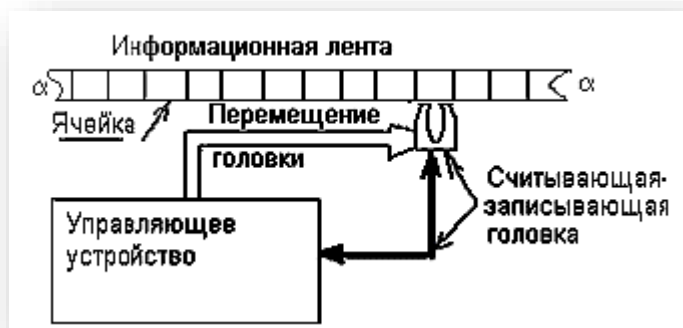


# Теория алгоритмов –



- Наука на стыке математики и информатики об общих свойствах и закономерностях **алгоритмов** и разнообразных формальных моделях их представления
- **Теоретическая основа** вычислительных наук
- Задачи:
  - **Формализация алгоритма** (модели вычислений) →
  - **Формализация задач**
  - Алгоритмическая **неразрешимость**
  - Уровни **сложности** (классификация, анализ, критерии качества).

# Способы формализации алгоритма



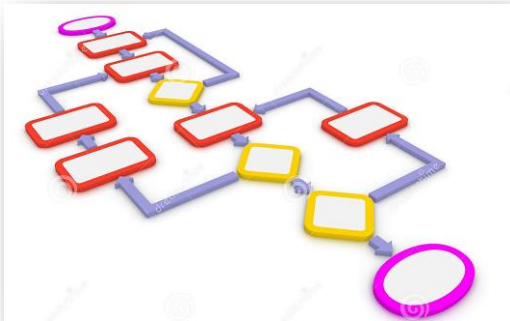
- **Теория автоматов:**
  - **машина Тьюринга** (модель процедурного программирования)
  - **машина Поста;**
- **Рекурсивные функции**  
Гёделя — Эрбрана — Клини
- **Нормальный алгоритм**  
Маркова
- **$\lambda$ -исчисление** Чёрча.

# Виды алгоритмов



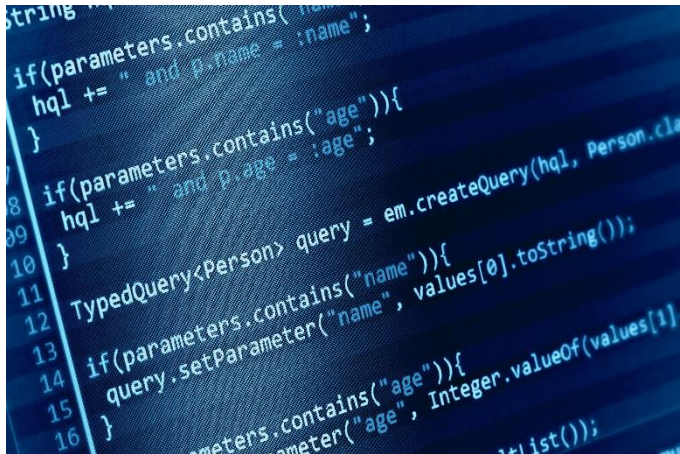
- **Детерминированные** (жёсткие, механические) – единственная и достоверная последовательность инструкций, приводящая к однозначному результату
- **Гибкие:**
  - **Вероятностные** (стохастические):
    - Используют **случайные величины** (ГСЧ),
    - Несколько путей решения, приводящие к **высоко вероятному достижению результата**;
  - **Эвристические** – используют различные разумные соображения без строгих обоснований.

# Свойства алгоритма:



- **Дискретность** – разбиение на конечное количество отдельных шагов
- **Понятность** – включает только команды из набора допустимых команд исполнителя
- **Детерминированность** (определённость) – каждый следующий шаг однозначно определяется состоянием системы – один и тот же ответ для одних и тех же исходных данных
- **Результативность** – всегда приводит к получению определённого результата
- **Массовость** – применимость к множеству наборов начальных данных
- **Завершаемость** (конечность) – результат за конечное время (число шагов).

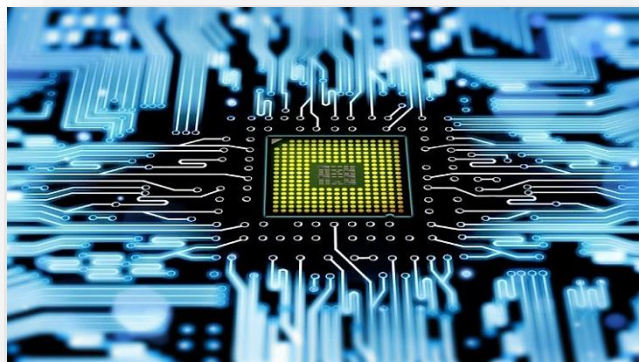
# Способы записи алгоритма



```
String hql = "select p from Person p";
if(parameters.containsKey("name")){
    hql += " and p.name = :name";
}
if(parameters.containsKey("age")){
    hql += " and p.age = :age";
}
TypedQuery<Person> query = em.createQuery(hql, Person.class);
if(parameters.containsKey("name")){
    query.setParameter("name", values[0].toString());
}
if(parameters.containsKey("age")){
    query.setParameter("age", Integer.valueOf(values[1].toString()));
}
return query.getResultList();
```

- **Словесный** (на естественном языке)
- **Формульный**
- **Табличный** (для реляционных задач)
- **Графический** (блок-схемы, UML-диаграммы, ДРАКОН-схемы)
- **Операторный** – из конечного набора допустимых команд исполнителя (ЯП, псевдокод).

# Компьютерная программа –



- Это алгоритм решения **вычислительной задачи** компьютером

- **Исполнитель**

- **Машинная команда:**

- КОп (обяз.часть)
- Адресная часть

Команда 1:	01011000	0110	0000000000001000000000
	операция	операнд 1	операнд 2
Команда 2:	01011010	0110	000000000000100000100
	операция	операнд 1	операнд 2
Команда 3:	01010000	0110	000000000000100001000
	операция	операнд 1	операнд 2

BB 11 01 B9 0D 00 B4 0E 8A 07  
43 CD 10 E2 F9 CD 20 48 65 6C  
6C 6F 2C 20 57 6F 72 6C 64 21

- **Скрипт.**

# Язык программирования –



- Это набор допустимых **операторов, синтаксические и семантические** правила их использования для создания компьютерных программ
- Уровневая классификация:
  - **ЯВУ**
  - **Ассемблеры** -  
машиноориентированные
  - Язык двоичных машинных кодов (**нативный код**)
- **Трансляция:**
  - **Интерпретация**
  - **Компиляция.**

## 2. Корректность алгоритма



# Методы оценки корректности алгоритма



1. Метод **перечисления**
2. Метод **инварианта цикла** →

# Инвариант



- Алгоритм **корректен**, если для каждого ввода результатом его работы является **корректный вывод**
- Методы оценки корректности – на принципах **математической индукции** (путём рассуждений)
- **Инвариант** – это свойство некоторого класса (множества) мат.объектов, остающееся **неизменным** при определённого вида преобразованиях.

# Инвариант цикла —

```
int j = 9;
for(int i=0; i<10; i++)
    j--;
```

## Примеры

### инвариантов:

а)  $i + j == 9$

б)  $i \geq 0 \ \&\& \ i \leq 10$

- Свойство, сохраняемое циклом — это **логическое выражение** (предикат), **истинное** непосредственно **перед** и сразу **после** каждой итерации цикла, зависящее от переменных, **изменяющихся в теле** цикла
- Инвариант цикла  $\neq$  условие цикла
- Инвариант может быть использован для **доказательства корректности** циклического алгоритма без необходимости его непосредственного выполнения (**верификация**)
- Чтобы убедиться, что **оптимизированный цикл** остался корректным, достаточно доказать, что **инвариант цикла не нарушен** и условие завершения цикла **достижимо**.

# Доказательство корректности цикла инвариантом

1. Доказывается, что выражение инварианта истинно перед началом цикла (**инициализация**).
  2. Доказывается, что выражение инварианта сохраняет свою истинность после выполнения тела цикла (**сохранение**). Так, по индукции, доказывается, что по завершении цикла инвариант будет выполняться.
  3. Доказывается, что при истинности инварианта после завершения цикла (**завершение**) переменные примут те значения, которые и требуется получить (что определяется из выражения инварианта и конечных значениях переменных в условии цикла).
  4. Доказывается (возможно, без применения инварианта), что цикл завершится, то есть условие завершения рано или поздно будет выполнено.
- Истинность утверждений на этих этапах однозначно свидетельствует о том, что цикл выполнится за конечное время и даст желаемый результат.

# Схема проверки инварианта цикла

```
// Инвариант цикла должен быть истинным здесь – при инициализации
while ( Условие_выполнения_цикла ) {
    // начало тела цикла
    ...
    // конец тела цикла
    // Инвариант цикла должен быть истинным здесь (после итерации)
}
// Инвариант цикла должен быть истинным здесь – по завершении цикла
}
```

# Пример – алгоритм поиска минимума в массиве

```
Min ← A[1]
for i ← 2 to n do
  if A[i] < Min then
    Min ← A[i]
  endif
od
```

Формулировка  
инварианта:

- В переменной Min записан минимум из первых  $i$  элементов  $[1, i)$  массива.

# Область неопределённости

```
Min ← A[1]
for i ← 2 to n do
  if A[i] < Min then
    Min ← A[i]
  endif
end
```

- Область изменения параметров задачи  $[1, n)$  можно разделить на две части:
  - **исследованную область**, для которой найден Min в  $[1, i)$ ;
  - **область неопределенности**  $[i+1, n)$ .
- Необходимо составлять цикл так, чтобы **на каждой итерации область неопределенности сокращалась**
- В начале первой итерации исследованная область представляет собой единственную точку 1, а область неопределенности составляет  $[2, n)$
- На втором шаге область неопределенности сокращается до  $[3, n)$ , на третьем – до  $[4, n)$  и т.д., пока не превратится в **пустое множество**.

# Пример – алгоритм суммирования элементов массива

```
Sum ← 0  
for i ← 1 to n do  
  Sum ← Sum + A[i]  
od
```

- После каждого шага цикла при любом  $i$  к переменной  $Sum$  добавляется элемент массива  $A[i]$
- После окончания очередного шага цикла **в  $Sum$  накоплена сумма всех элементов массива с номерами от 1 до  $i$**
- Вывод: после завершения цикла ( $i=n$ ), в  $Sum$  будет записана сумма всех элементов массива.



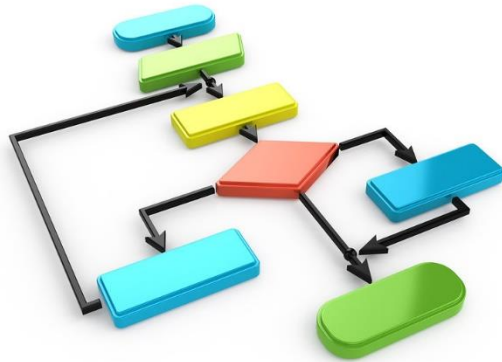
# Пример – сортировка массива пузырьком

```
for i ← 1 to n-1 do
  for j ← n-1 downto i do
    if A[j] > A[j+1] then
      c ← A[j]; A[j] ← A[j+1]; A[j+1] ← c;
    endif
  od
od
```

- На каждом шаге **внешнего цикла** на свое место «всплывает» один элемент массива
- Поэтому **инвариант внешнего цикла**: «После выполнения  $i$ -го шага цикла первые  $i$  элементов массива отсортированы и установлены на свои места»
- Во **внутреннем цикле** очередной «лёгкий» элемент поднимается вверх к началу массива
- Перед первым шагом внутреннего цикла элемент, который будет стоять на  $i$ -м месте в отсортированном массиве, может находиться в любой ячейке от  $A[i]$  до  $A[n]$
- После каждого шага его «зона нахождения» сужается на одну позицию
- **Инвариант внутреннего цикла**: «Элемент на  $i$ -м месте в отсортированном массиве может находиться в любой ячейке от  $A[i]$  до  $A[j]$ »
- Когда в конце этого цикла  $j = i$ , элемент  $A[i]$  встаёт на своё место.

# 3. Анализ эффективности алгоритма

# Анализ алгоритма



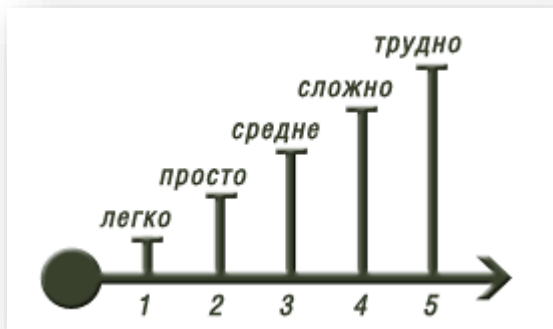
- Позволяет предсказать **требуемые** для его выполнения **ресурсы** (время работы процессора, память и пр.)
- На основе анализа нескольких алгоритмов можно выбрать **наиболее эффективный**. →

# Эффективность алгоритма



- Критерии – **скорость** (время) и **расход памяти** (или других ресурсов – диска, трафик в сети и пр.)
- Алгоритм A1 эффективнее алгоритма A2, если алгоритм A1 выполняется за меньшее время и (или) требует меньше компьютерных ресурсов
- Составляющие эффективности:
  - Время – мера **системной эффективности**
  - Расход памяти – мера **пространственной эффективности**
  - Количество команд относительно количества обрабатываемых данных – мера **вычислительной эффективности**.

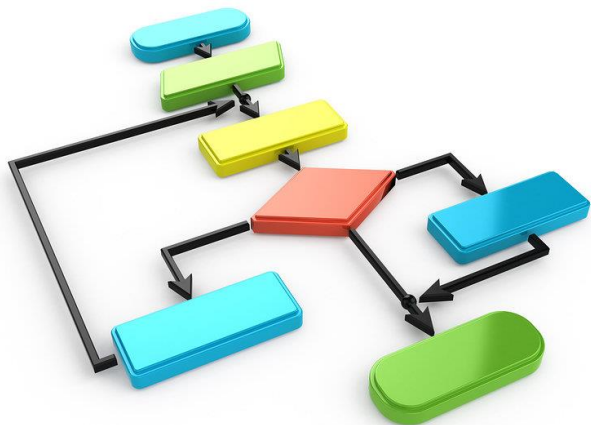
# Сложность алгоритма



**Сложность** как характеристика связана с эффективностью:

- **Эффективный** алгоритм требует **приемлемое** время исполнения и разумную ресурсоемкость
- **Сложность** возрастает при **увеличении** времени исполнения алгоритма и (или) задействованных ресурсов
- Т.о. для одной и той же задачи **более сложный** алгоритм из нескольких характеризуется **меньшей эффективностью**.

# Вычислительная сложность



- Составляющие:

- **Временная сложность** - отражает временные затраты на реализацию алгоритма
- **Емкостная сложность** – отражает объём требующейся алгоритму памяти

- Подходы к оценке:

- **Эмпирический анализ** (экспериментальный, практический):
  - Практический метод →
  - Теоретический метод
- **Асимптотический анализ.**

# Практический метод (1/2)

```
#include <ctime>

const int n=100000000;
double sum(int *x, int n);

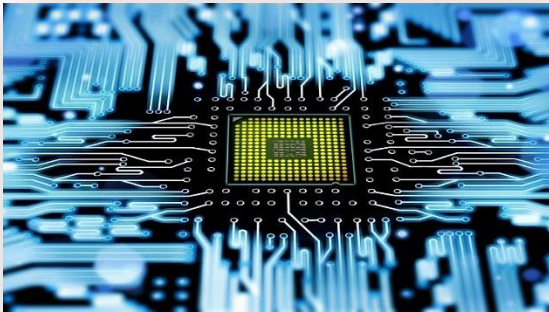
int main(){
int x[n];
srand(time(0));
for(int i=0;i<n;i++)
x[i]=rand();
sum(x,n);
int t2=clock()/CLOCKS_PER_SEC;
cout<<"при n="<<n<<"t= "<<t2<<"\n";
}

double sum(int *x, int n){
int s=0;
for(int i=0;i<n;i++){
s=s+x[i];
}
return s;
}
```

Характеризуется  
**измеримыми**  
**параметрами:**

- Временная сложность –  
**во временных единицах**  
(микро-, милли-, секундах) или  
**количестве тактов**  
процессора
- Емкостная сложность – в  
**битах** (байтах и производных единицах),  
минимальных аппаратных  
требованиях и пр.

# Практический метод (2/2)



Факторы, влияющие на оценку:

- Особенности **аппаратно-программной платформы**:
  - Характеристики **оборудования** (тактовая частота, объём ОЗУ и сверхоперативной памяти, размер файла подкачки)
  - Архитектура **программной среды** (многозадачность, алгоритм работы планировщика задач, особенности ОС)
- **Язык программирования** (транслятор)
- Квалификация (**опыт**) программиста
- В результате – практическая оценка **не является абсолютным показателем эффективности** (сложности).



# Примеры определения практической сложности

**функция clock() модуля ctime  
(реальное время):**

```
#include <ctime>

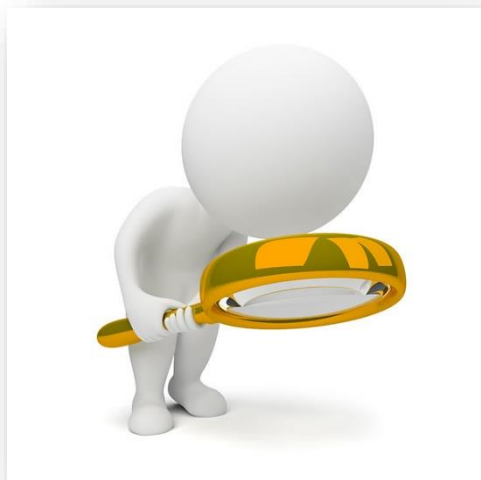
time_t begin = clock();
//вызов функции с алгоритмом...
time_t end = clock();
double time_spent = (double)
(end - begin) / CLOCKS_PER_SEC;
```

**через библиотеку chrono  
(стабильное время):**

```
#include <stdlib>
#include <iostream>
#include <chrono>

auto begin =
chrono::steady_clock::now();
getchar();
auto end =
chrono::steady_clock::now();
auto elapsed_ms =
chrono::duration_cast <
chrono::milliseconds>(end - begin);
cout << "The time: " <<
elapsed_ms.count() << " ms\n";
```

# Теоретический подход (1/2)



- Характеризует алгоритм **без привязки** к конкретному оборудованию, ПО и средствам реализации
- **Временная сложность** – в количестве операций, тактах работы машины Тьюринга и пр.
- **Емкостная сложность** определяется объёмом данных (входных, промежуточных, выходных), числом задействованных ячеек на ленте машины Тьюринга и пр.

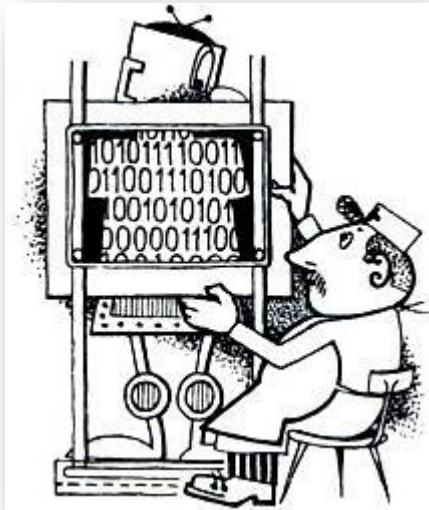
# Теоретический подход (2/2)



Факторы, влияющие на оценку эффективности (сложности):

- Объём входных данных (**размер входа**, размерность задачи) – например, количество элементов в массиве на сортировку или длина строки и пр.
- **Метод решения** – например, тот или иной алгоритм сортировки.

# Модель вычислительной машины



- Идеализированная  
одноядерная  
однопроцессорная **машина** с  
памятью с произвольным  
доступом (RAM)
- **Команды** – арифметические,  
перемещения данных,  
управляющие
- Каждая команда выполняется  
за определённое  
**фиксированное время.** →