Алгоритмы и их сложность

Введение

Определение 1. ***Алгоритм*** *– точное предписание, которое задает вычислительный процесс, начинающийся с произвольного* ***исходного данного*** *и направленный на получение полностью определенного этим исходным данным* ***результата****.*

Алгоритм *– это четкое описание по выполнению некоторого* ***процесса обработки данных****, который через разумное конечное число шагов приводит к решению задачи данного типа для любых допустимых вариантов исходных данных****.***

Определение 2. ***Данные*** *– это информация (числа, факты, характеристики явлений и пр.), представленная в формализованном (конкретном) виде****.***

Для решения любой задачи необходимо:

1. Ввести исходные данные.

2. Преобразовать исходные данные в результаты (выходные данные).

3. Вывести результаты.

Определение 3. ***Теория алгоритмов*** *— наука, изучающая общие* ***свойства*** *и* ***закономерности*** *алгоритмов и разнообразные формальные модели их представления.*

Задачи теории алгоритмов *:*

- *доказательство алгоритмической неразрешимости задач*
- *анализ сложности алгоритмов*
- *классификация алгоритмов*
- *разработка критериев оценки качества алгоритмов*

  и другие.

Пример алгоритма числовой обработки данных:

Вычислить значение

$$Y = X^2 + 1$$

Последовательность действий для выполнения алгоритма:

- *Ввести в компьютер значение* ***X****;*
- *Возвести его значение во вторую степень и получить значение* X^2*;*
- *Прибавить к полученному значению единицу:* $X^2 + 1$ *и присвоить его переменной* ***Y****;*
- *Вывести на экран, полученное значение Y.*

Вычислить сумму *

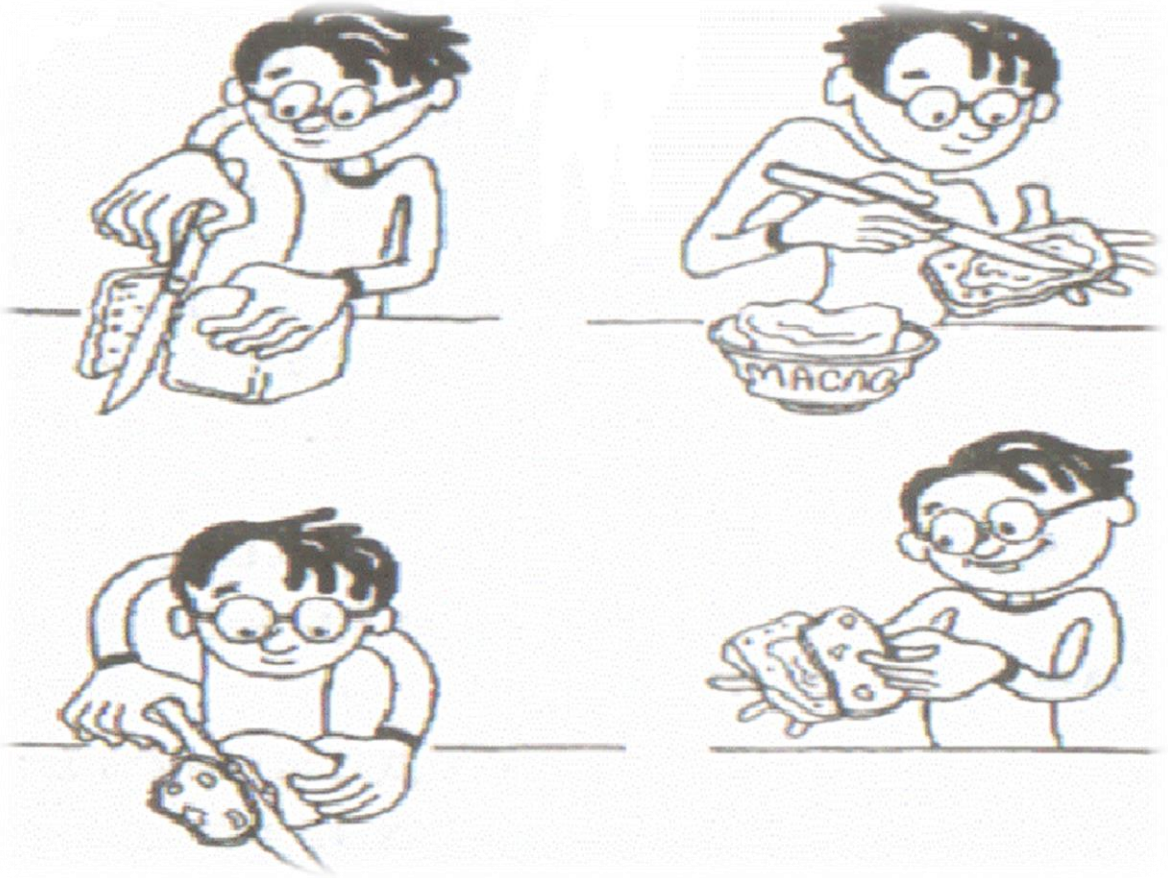
$$1 + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{n!}$$

Последовательность действий для выполнения алгоритма:

- *Ввести в компьютере значение n*
- *1. Считаем каждый член суммы независимо, затем суммируем. Время выполнения* $O(n^2)$
- *2. k-ый член суммы получаем из (k-1)-го делением на к;*

  Время выполнения $O(n)$

2ой вариант более предпочтительный.



Пример алгоритма в быту. Как сделать бутерброд

- *Взять хлеб. Отрезать кусок.*
- *Взять масло. Намазать на отрезанный кусок хлеба.*
- *Взять колбасу. Отрезать кусок.*
- *Положить колбасу на хлеб.*

Свойства алгоритмов:

1. ***Дискретность (прерывность)*** *- алгоритм должен представлять процесс решения задачи как последовательное выполнение простых шагов.*
2. ***Определенность*** *- каждое правило алгоритма должно быть четким, однозначным и не оставлять места для вариаций.*
3. ***Результативность (конечность)*** *- алгоритм должен приводить к решению задачи за конечное число шагов.*
4. ***Детерминированность****. После каждого шага необходимо указывать, какой шаг выполняется следующим, либо давать команду остановки*
5. ***Массовость*** *- алгоритм решения задачи разрабатывается в общем виде и должен быть применим для некоторого класса задач, различающихся только входными данными. При этом входные данные могут выбираться из некоторой области, которая называется* ***областью применимости алгоритма.***

Средства описания алгоритма

Формы представления алгоритмов:

*1. **Словесная** (записи на естественном языке);*

*2. **Графическая** (изображения из графических символов);*

*3. **Псевдокоды** (описания алгоритмов на условном алгоритмическом языке, включающие в себя элементы языка программирования, фразы естественного языка, общепринятые математические обозначения и др.);*

*4. **Программная** (тексты на языках программирования)*

Словесная форма представления алгоритмов.

Словесный способ *записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.*

Может быть описана в устной форме, письменной форме на естественном языке, в письменной форме на формальном языке.

Например*. Записать алгоритм нахождения наибольшего общего делителя двух натуральных чисел.*

1. *Задать два числа;*	*5 и 10*
2. *Если числа равны, то взять любое из них в качестве ответа и остановиться, иначе продолжить выполнение алгоритма;*	*5≠10*
3. *Определить большее из чисел;*	*10*
4. *Заменить большее из чисел разностью большего и меньшего из чисел;*	*10 -> 5*
5. *Повторить алгоритм с шага 2.*	
Результат:	*5*

Словесный способ *имеет ряд недостатков:*

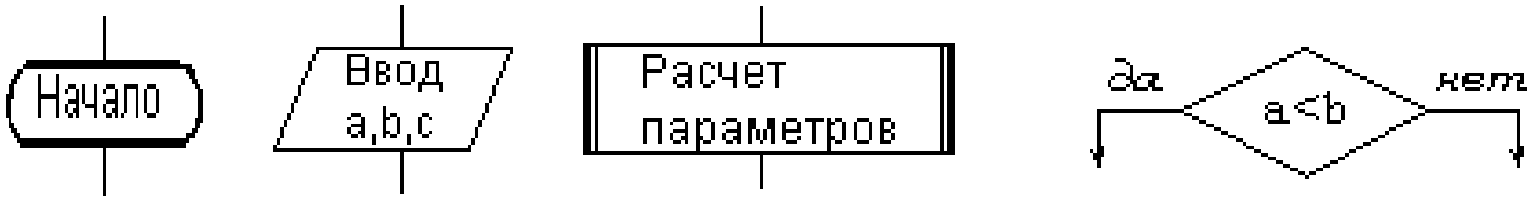
- *такие описания строго не формализуемы;*
- *допускают неоднозначность толкования отдельных предписаний;*
- *страдают многословностью записей.*

Графическая форма представления алгоритмов.

Графический способ *представления алгоритмов является более компактным и наглядным по сравнению со словесным. Его графическое представление называется схемой алгоритма или блок-схемой***.**

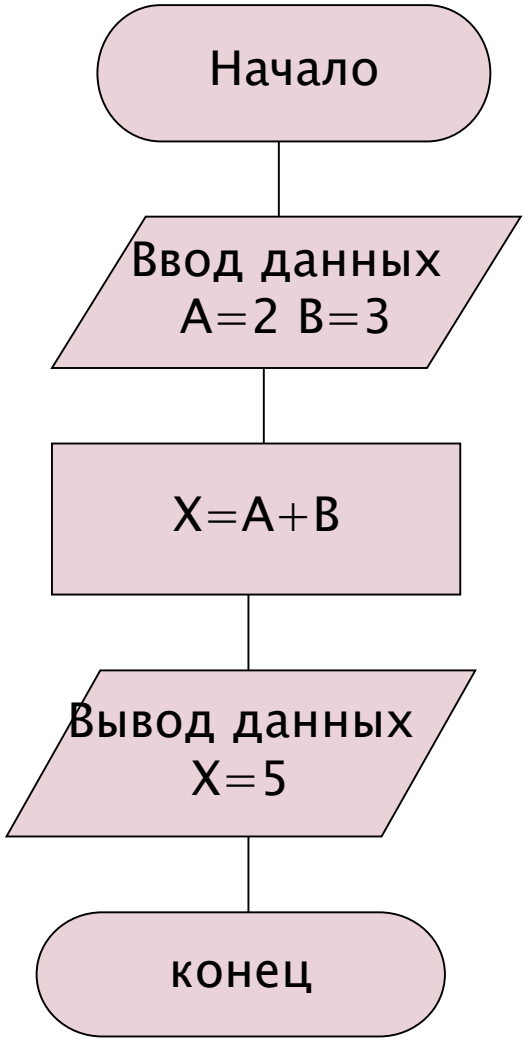
Схемой алгоритма*– графическое изображение алгоритма в виде схемы связанных между собой с помощью стрелок блоков, каждый из которых соответствует одному шагу алгоритма.*

В схеме алгоритма каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа.



Линейный алгоритм

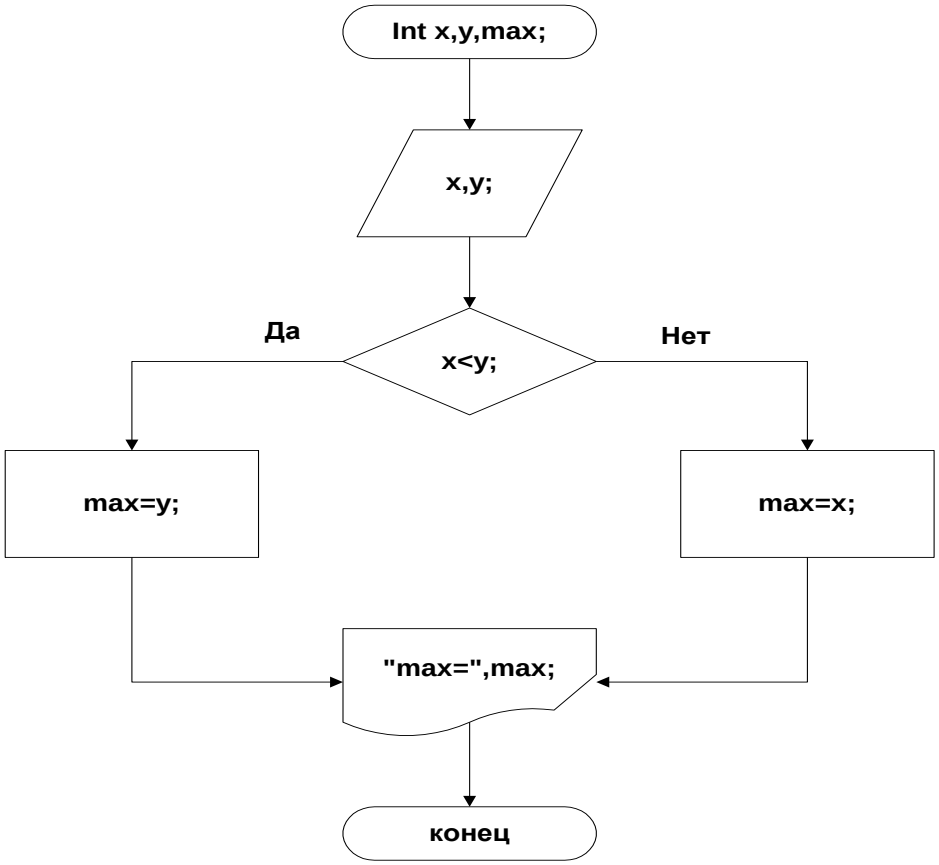
Линейным называется алгоритм, в котором результат получается путем однократного выполнения заданной последовательности действий при любых исходных данных. Операторы задействованы последовательно, один за другим, в соответствии с их расположений в тексте программы



Разветвляющийся алгоритм

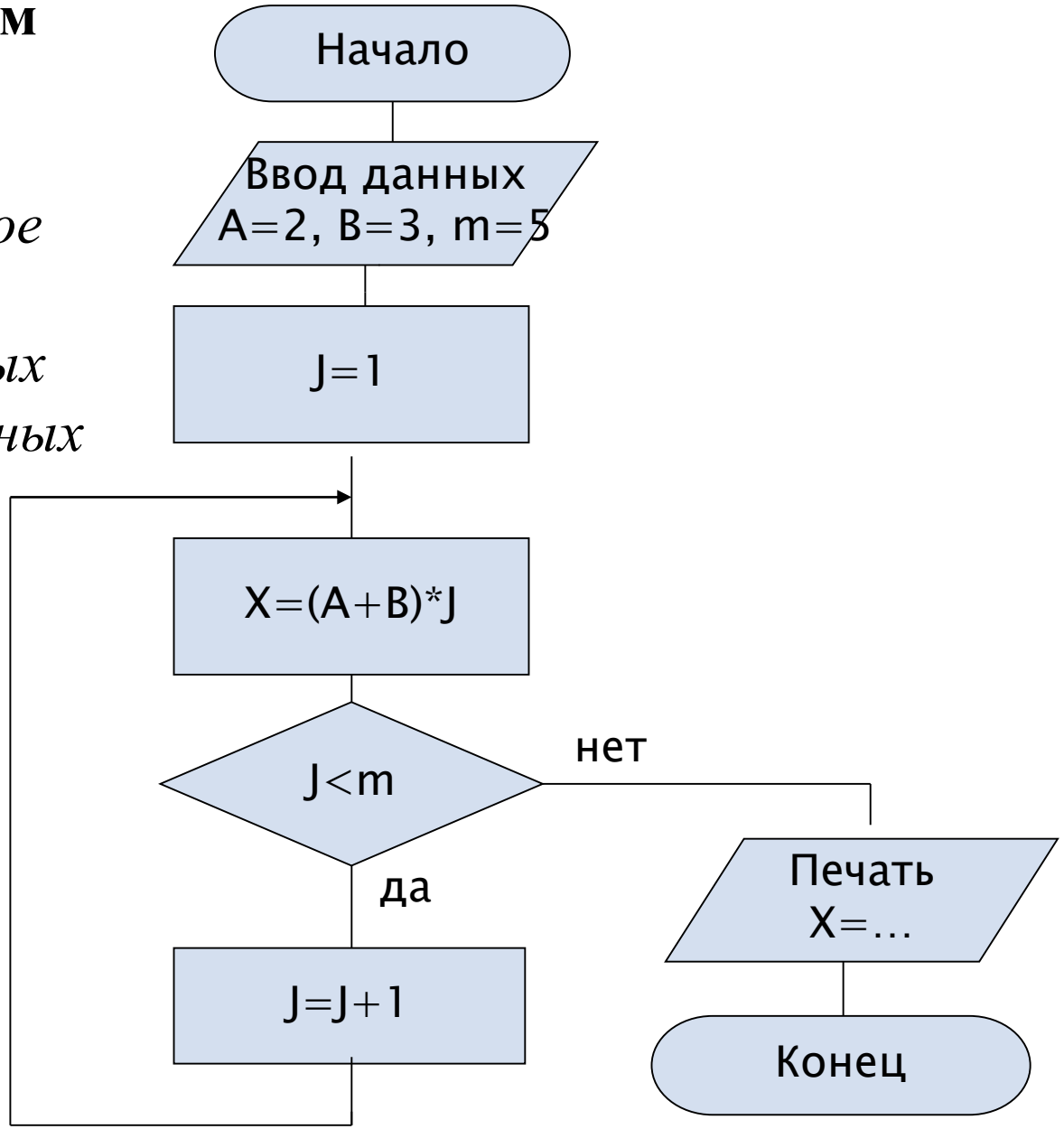
Алгоритм называется разветвляющимся, если он содержит несколько ветвей выполнения программы отличающихся друг от друга содержанием вычислений

Нахождение максимального числа



Циклический алгоритм

Алгоритм называется циклическим, если он содержит многократное выполнение одних и тех же ветвей при различных значениях промежуточных данных



Псевдокод

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов.

Особенности:

- *Не приняты строгие синтаксические правила для записи команд, что облегчает запись алгоритма на стадии его проектирования.*
- *Имеются некоторые конструкции, присущие формальным языкам.*
- *Есть служебные слова, смысл которых определен раз и навсегда. Они выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются.*
- *Единого или формального определения псевдокода не существует, возможны различные псевдокоды, отличающиеся набором служебных слов и основных конструкций. К таким конструкциям обычно относят* ***ветвления*** *(если ...то ... иначе ...) и* ***циклы*** *(цикл от ... до ..., цикл пока, цикл до...).*

```
алг HELLOWORLD
нач
вывод ('Hello,World')
кон алг HELLOWORLD

алг Сумма квадратов (арг цел n, рез цел S)
дано | n > 0
надо | S = 1*1 + 2*2 + 3*3 + ... + n*n
нач цел i
  ввод n; S:=0
  нц для i от 1 до n
    S:=S+i*i
  кц
  вывод "S = ", S
кон
```

Программная форма представления алгоритмов.

Для реализации на ПК алгоритм необходимо описать на одном из ***языков программирования****.*
Алгоритм, записанный на языке программирования, называется ***программой.***
Текст программы называют ***листинг****.*
При вводе в ПК программа - транслятор "переводит" алгоритм на ***машинный алгоритмический язык****, в котором все данные и все действия представляются* в виде двоичных чисел.
Ниже приведено описание алгоритма “Одеться по погоде” средствами языка C#:
Если на улице температура ниже 0, то необходимо надеть шубу, иначе – куртку.

```
void DressedDependOnWeather ( )
        {
            double Temperature; //Описываем новую переменную типа число с плавающей точкой
            Console.WriteLine("Введите температуру воздуха:"); //Вывод в консоль запроса
            Temperature = Convert.ToDouble(Console.ReadLine()); //Считывание с консоли введенного числа
            if(Temperature < 0.0) //Проверяем меньше ли 0 температура на улице
            {
              Console.WriteLine("Оденьте шубу"); //Вывод ответа в консоль
            }
            else
            {
              Console.WriteLine("Оденьте куртку"); //Вывод ответа в консоль
            }
        }
```

Сложность алгоритмов

Анализ трудоёмкости алгоритмов

Целью анализа трудоёмкости алгоритмов является нахождение оптимального алгоритма для решения данной задачи.

Определение 4. ***Теория сложности****, являясь частью* ***теории вычислений****, изучает ресурсы или стоимость вычислений, необходимые для выполнения поставленной проблемы.*

Определение 5. ***Вычислительная сложность алгоритма*** *— это функция, определяющая зависимость объёма работы, выполняемой некоторым алгоритмом, от свойств входных данных. Объём работы обычно измеряется абстрактными понятиями времени и пространства, называемыми вычислительными ресурсами.*

Время *определяется количеством элементарных шагов, необходимых для решения проблемы, тогда как* ***пространство*** *определяется объёмом памяти или места на носителе данных.*

Центральный вопрос разработки алгоритмов: «как изменится время исполнения и объём занятой памяти в зависимости от размера входа и выхода?».

Градации сложности

$f(n) = O(1)$ – *константная;*

$f(n) = O(\log n)$ – *логарифмическая; (см. пример ^ алгоритм 2)*

$f(n) = O(n)$ – *линейная; (см. пример * алгоритм 2)*

$f(n) = O(n^c)$ - *полиномиальная (квадратичная, кубическая …); (см. пример* алгоритм 1)*

$f(n) = O(c^n)$ – *экспоненциальная; (см. пример ~ алгоритм 1)*

$f(n) = O(n!)$ – *факториальная*

Запись вида f(n) = O(g(n)) означает, что ф-ия $f(n)$ *возрастает медленнее чем ф-ия* $g(n)$*, т.е если при* $g(n) > 0$ *и* $n > 0$ *существуют положительные* c_1 *и* n_0*, такие что при любом* $n > n_0$ *выполняется* $c_1 \cdot g(n) \geq f(n)$.
c_1 *и* n_0 *могут быть сколь угодно большими числами.*

Такая ***О оценка*** *дает нам верхнюю оценку временной трудоемкости алгоритма – его асимптотическую сложность.*

Практические соображения сложности алгоритмов

- *Сложность по времени и по используемой памяти;*
- *Требования к решению;*
- *Сложность реализации и отладки алгоритма;*
- *Константа скрытая в* обозначении сложности $\Theta(k), k = ?$
- *Оценки выполнения алгоритма в среднем и худшем случае.*

Примеры алгоритмов и их сложности

Задача 1. *Вычислить n-ое число Фибоначчи (**полиномиальная сложность**):*

$$f_1 = 1, f_2 = 1, f_k = f_{k-2} + f_{k-1}$$

Алгоритм 1.1

- *Считаем рекурсивно «с конца»* $(f_n, f_{n-1}, \ldots)$;

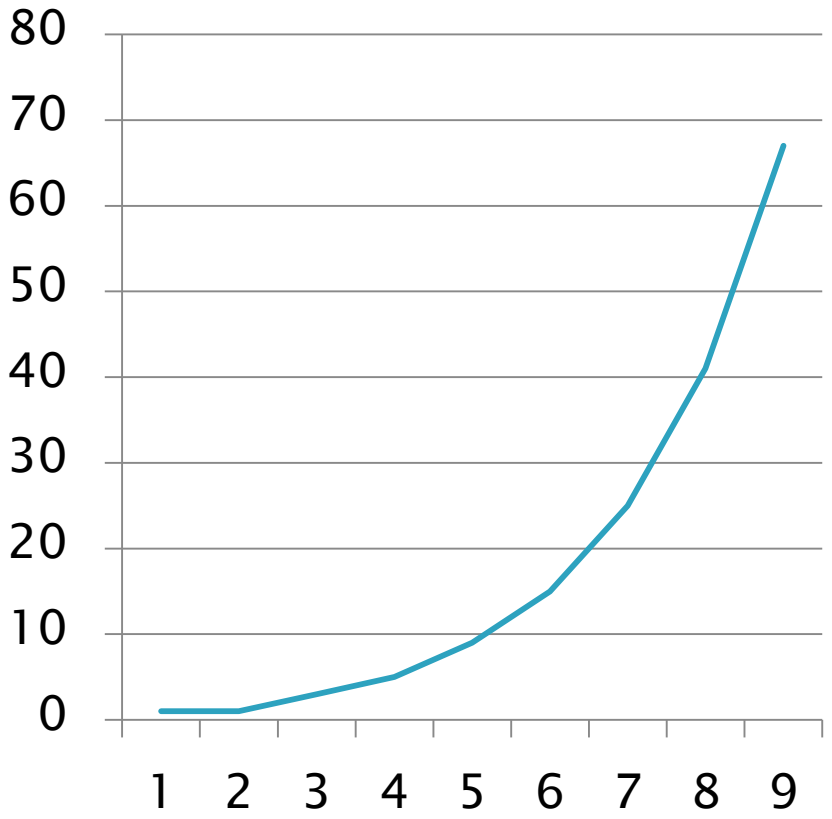
$$f_n = f_{n-2} + f_{n-1} \rightarrow f_{n-1} = f_{n-2} - f_{n-3} \rightarrow \cdots f_3 = f_1 + f_2$$

- *Сложность выполнения алгоритма* $O(2^n)$

```
double FibFunc(int IndexNumber)//Передаем в функцию номер числа
{
   if (IndexNumber == 1 || IndexNumber == 2) //Проверяем условие дошли ли до
            1 или 2 числа
             return 1.0;//Если дошли, то возвращаем известные значения
   else
             return (FibFunc(IndexNumber - 1) + FibFunc(IndexNumber - 2));//Если
           нет, то рекурсивно вызываем сами себя используя формулу
}
```

Количество вызовов функции

Номер числа	Число Фибоначчи	Число вызовов функций
1	1	1
2	1	1
3	2	3
4	3	5
5	5	9
6	8	15
7	13	25
8	21	41
9	34	67



Алгоритм 1.2

- *Считаем в цикле «с начала»* $(f_1, f_2, \ldots)$;

$$f_3 = f_1 + f_2 \rightarrow f_4 = f_2 + f_3 \rightarrow \cdots f_n = f_{n-2} + f_{n-1}$$

- *Сложность выполнения алгоритма* $O(n)$ ***(линейная)***

```
double FibFunc(int IndexNumber)
    {
        double f1, f2, f3; //Объявляем переменные
        f1 = f2 = 1.0; //Задаем извесные значения для 1го и 2го числа
        f3 = 0.0;
        int i = 3; //Объявляем и задаем значение переменной шага
        while (i <= IndexNumber)//Пока не достигнем необходимого номера числа
                                Фибоначчи
        {
            f3 = f2 + f1; // Определяем iое число по формуле
            //Меняем для следующего шага переменные
            f1 = f2; //f2 становится f1
            f2 = f3; //f3 становится f2
            i = i  + 1; //Увеличиваем шаг (номер числа) на единицу
        }
        return  f3; //Возвращаем последнее посчитанное значение
    }
```

Задача 2. *Вычислить* a^n:

Алгоритм 2.1 (линейная сложность)

- *Считаем в цикле* $a, a^2, a^3, \ldots$;
- *Сложность выполнения алгоритма* $O(n)$

Алгоритм 1.2 (логарифмическая сложность)

- *Считаем используя свойство* $a^{2k} = (a^2)^k$
- *Сложность выполнения алгоритма* $O(log_2 n)$

```
long mypow(long a, long k)
       { long b = 1;
         while (k !=0 )
         {
           if (k % 2 == 0) {k /= 2;a *= a;}
               else {k--; b *= a;}
         }
         return b;
       }
```

Количество итераций

Степень	Число итераций
1	1
2	2
3	3
4	3
5	4
6	4
7	5
8	4
9	5
10	5
11	6
12	5
13	6
14	6