Реализация известных алгоритмов на языке программирования Python

- Алгоритм Евклида (нахождение наибольшего общего делителя)
- Анализ выборки
- Вычисление факториала на языке программирования Python
- Двоичный (бинарный) поиск элемента в массиве
- Перебор делителей ("тестирование простоты")
- Перевод чисел из десятичной системы счисления в двоичную
- Пересечение списков (поиск одинаковых элементов в двух списках)
- Решето Эратосфена - алгоритм определения простых чисел
- Сортировка выбором (поиск минимума и перестановка)
- Сортировка методом пузырька
- Числа Фибоначчи (вычисление с помощью цикла while и рекурсии)

Алгоритм Евклида (нахождение наибольшего общего делителя)

Алгоритм Евклида – это алгоритм нахождения наибольшего общего делителя (НОД) пары целых чисел.

Наибольший общий делитель (НОД) – это число, которое делит без остатка два числа и делится само без остатка на любой другой делитель данных двух чисел. Проще говоря, это самое большое число, на которое можно без остатка разделить два числа, для которых ищется НОД.

Описание алгоритма нахождения НОД делением

1. Большее число делим на меньшее.
2. Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).
3. Если есть остаток, то большее число заменяем на остаток от деления.
4. Переходим к пункту 1.

Пример:
Найти НОД для 30 и 18.
30/18 = 1 (остаток 12)
18/12 = 1 (остаток 6)
12/6 = 2 (остаток 0). Конец: НОД – это делитель. НОД (30, 18) = 6

Исходный код на Python

```
a = 50
b = 130

while a!=0 and b!=0:
    if a > b:
        a = a % b
    else:
        b = b % a

print (a+b)
```

Примечание к коду. В цикле в a или b записывается остаток от деления. Когда остатка нет (мы не знаем в a он или b, поэтому проверяем оба условия), то цикл завершается. В конце выводится сумма a и b, т.к. мы не знаем, в какой переменной записан НОД, а в одной из них в любом случае 0, который на результат суммы никак не влияет.

Описание алгоритма нахождения НОД вычитанием

1. Из большего числа вычитаем меньшее.
2. Если получается 0, то значит, что числа равны друг другу и являются НОД (следует выйти из цикла).
3. Если результат вычитания не равен 0, то большее число заменяем на результат вычитания.
4. Переходим к пункту 1.

Пример:
Найти НОД для 30 и 18.
30 - 18 = 12
18 - 12 = 6
12 - 6 = 6
6 – 6 = 0 Конец: НОД – это уменьшаемое или вычитаемое. НОД (30, 18) = 6

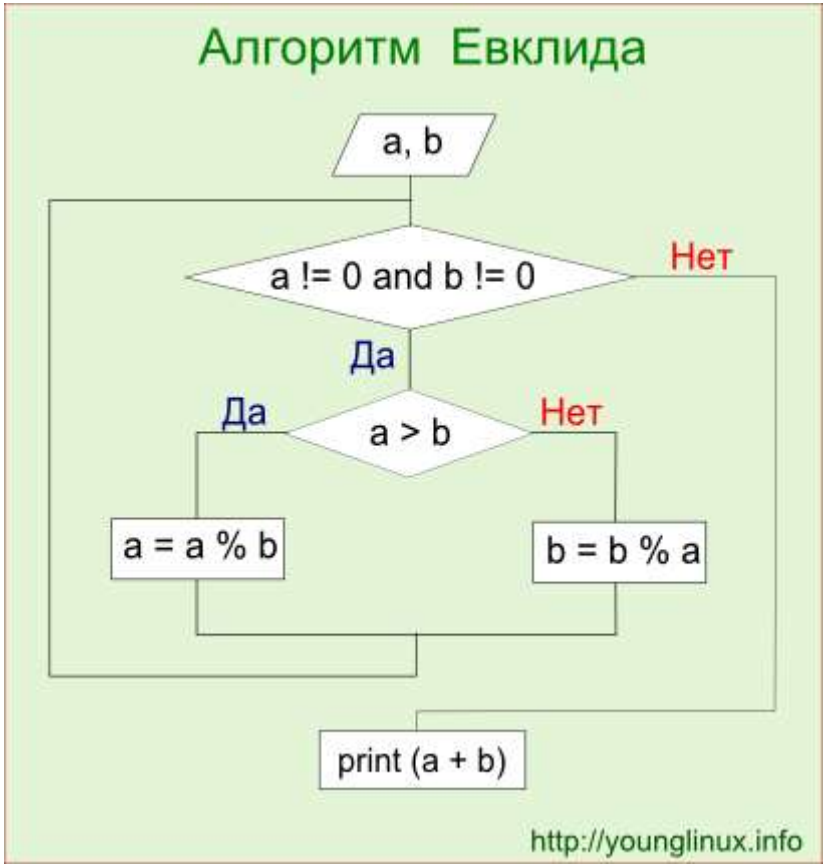
Исходный код на Python

```
a = 50
b = 130

while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a

print (a)
```

Оформление кода в виде функции

```
1.  def gcd(a,b):
2.      while a != b:
3.          if a > b:
4.              a = a - b
5.          else:
6.              b = b - a
7.      print (a)
```

Блок-схема "Алгоритм Евклида"



Анализ выборки

Описание задачи

Часто требуется проанализировать какой-то ряд значений и определить количество значений, попавших в каждый определенный диапазон. Например, дан список, содержащий 1000 значений натуральных чисел в диапазоне от 1 до 100. Требуется подсчитать, сколько значений попало в диапазоны от 1 до 20, от 21 до 30, от 31 до 40 и т.д. Полученный таким образом результат можно использовать для построения графиков и диаграмм частот встречаемости значений.

Пример исходного кода на Python

```
1.  #анализируемый список (можно подставить другой)
2.  a = [3,5,7,3,8,1,8,0,7,3,2,4,6,8,5,4,3,3,6,5,7,8,9,5,3,2,3]
3.
4.  bottom = int(input("нижняя граница: "))
5.  top = int(input("верхняя граница: "))
6.  interval = int(input("интервал: "))
7.
8.  #количество интервалов
9.  num_interval = int((top - bottom) / interval)
10.
11. top = bottom #опускаем верхнюю границу до нижней
12. for i in range(num_interval): #выполняется подсчет значений для каждого интервала
13.     bottom = top #сдвиг нижней границы к верхней
14.     top = top + interval #сдвиг верхней границы на величину интервала
15.     print("От",bottom,"до",top)
16.     calculator = 0 #счетчик для подсчета количества значений в текущем интервале
17.     for j in a: #проверяется каждый элемент в списка ...
18.         if bottom <= j < top: #на вхождение в текущий интервал, в случае успеха ...
19.             calculator += 1 #увеличение значения счетчика
20.     print (calculator,"значений \n")
```

Вычисление факториала на языке программирования Python

Факториалом числа называют произведение всех натуральных чисел до него включительно. Например, факториал числа 5 равен произведению 1*2*3*4*5 = 120. Формулу нахождения факториала можно записать следующим образом: **n! = 1 * 2 * … * n**, где n – это число, а n! – факториал этого числа.

Можно записать немного по-другому: **n! = 1 * … * (n-2) * (n-1) * n**, т.е. каждое предыдущее число меньше на единицу, чем последующее. Нахождение факториала числа по первой формуле можно реализовать с помощью цикла while, а по второй формуле – с помощью рекурсии.

Исходный код на Python с использованием цикла

```
1.   n = input("Факториал числа ")
2.   n = int(n)
3.   fac = 1
4.   i = 0
5.   while i < n:
6.        i += 1
7.        fac = fac * i
8.   print ("равен",fac)
```

Исходный код на Python с использованием рекурсии

```
1.   def fac(n):
2.        if n == 0:
3.             return 1
4.        return fac(n-1) * n
```

Описание пошаговой реализации рекурсии

0 шаг. Вызов функции: fac(5)
1. fac(5) возвращает fac(4) * 5
2. fac(4) => fac(3) * 4
3. fac(3) => fac(2) * 3
4. fac(2) => fac(1) * 2
5. fac(1) => 1
6. 1 * 2 - возврат в вызов fac(2)
7. 2 * 3 - fac(3)
8. 6 * 4 - fac(4)
9. 24 * 5 – fac(5)
10. Возврат в основную ветку программы значения 120.

Двоичный (бинарный) поиск элемента в массиве

Двоичный поиск значения в списке (или массиве) используется для упорядоченных последовательностей (отсортированных по возрастанию или убыванию). Заключается такой поиск в определении, содержит ли массив определенное значение, а также определение места его нахождения.

Описание алгоритма

1. Находится средний элемент последовательности. Для этого первый и последний элементы связываются с переменными, а средний вычисляется.
2. Средний элемент сравнивается с искомым значение. В зависимости от того, больше оно или меньше среднего элемента, дальнейший поиск будет происходить лишь в левой или правой половинах массива. Если значение среднего элемента окажется равным искомому, то поиск завершен.
3. Одна из границ исследуемой последовательности становится равной предыдущему или последующему среднему элементу из п.2.
4. Снова находится средний элемент, теперь уже в «выбранной» половине. Описанный выше алгоритм повторяется уже для данной последовательности.

Исходный код на Python

```
1.   li = [0,3,5,7,10,20,28,30,45,56] # исходный список
2.   x = 28 # искомое значение
3.
4.   i = 1 # первый элемент
5.   j = len(li) # последний элемент
6.   m = int((i + j) / 2) # середина (приблизительно)
7.
8.   # пока искомое значение не равно текущей середине
9.   # или левая граница зашла за правую (элемент не найден)
10.  while li[m]!=x or i>j:
11.       m = int((i + j) / 2) # найти новую середину
12.       if x > li[m]: # если значение больше серединного
13.            i = m + 1 # то сместить левую границу за середину
14.       else: # иначе
15.            j = m - 1 # переместить правую границу до середины
16.
17.  if i > j:
18.       print ("Элемент не найден")
19.  else:
20.       print (m) # вывести индекс искомого элемента
```

Перебор делителей ("тестирование простоты")

Описание алгоритма

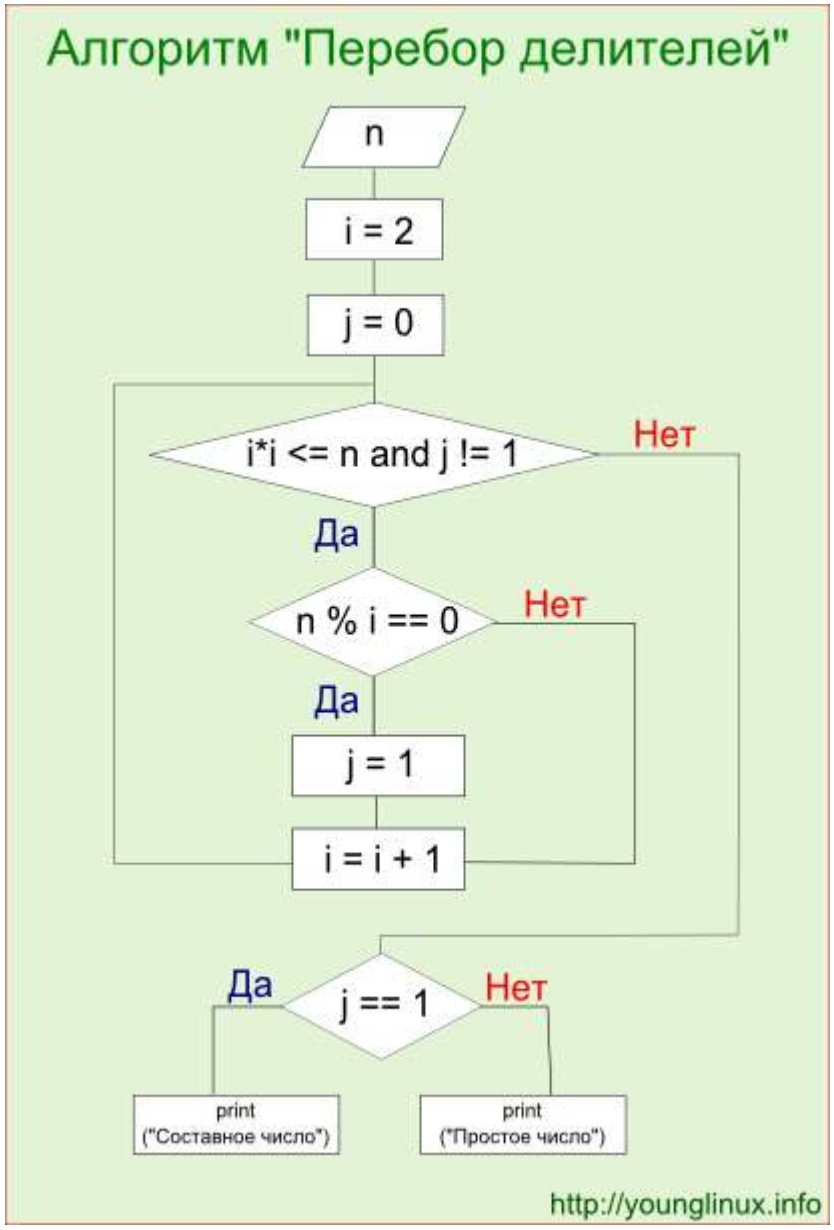
Перебор делителей – это алгоритм, предназначенный для определения, какое число перед нами: простое или составное.

Алгоритм прост и заключается в последовательном делении заданного натурального числа на все целые числа, начиная с двойки и заканчивая значением меньшим или равным квадратному корню тестируемого числа. Если хотя бы один делитель делит тестируемое число без остатка, то оно является составным. Если у тестируемого числа нет ни одного делителя, делящего его без остатка, то такое число является простым.

Исходный код на Python

```
1.  def divider(n):
2.      i = 2
3.      j = 0 # флаг
4.      while i**2 <= n and j != 1:
5.          if n%i == 0:
6.              j = 1
7.          i += 1
8.      if j == 1:
9.          print ("Это составное число")
10.     else:
11.         print ("Это простое число")
```

Блок-схема


Алгоритм "Перебор делителей"
n
i = 2
j = 0
i*i <= n and j != 1
Нет
Да
n % i == 0
Нет
Да
j = 1
i = i + 1
Да
j == 1
Нет
print
("Составное число")
print
("Простое число")
http://younglinux.info

Перевод чисел из десятичной системы счисления в двоичную

Один из алгоритмов получения двоичного числа из десятичного можно описать следующим образом:

1. Исходное десятичное число делится на два (основание двоичной системы счисления).
2. В одну переменную записывается частное в виде целого числа, в другую – остаток в виде строки (если остатка нет, то записывается ноль).
3. Если частное не было равно нулю, то оно снова делится на два. Переменная, связанная со старым частным связывается с новым (прежнее частное теряется). Новый остаток с помощью операции конкатенации добавляется в начало строковой переменной, где хранятся остатки.
4. П. 3 продолжает повторяться до тех пор, пока частное не станет равно нулю.
5. Остатки от деления, записанные в обратном порядке, представляют собой двоичное представление заданного десятичного числа.

```
1.  x = int(input("Введите натуральное число: "))
2.  n = ""
3.
4.  while x > 0:
5.      y = str(x % 2)
6.      n = y + n
7.      x = int(x / 2)
8.
9.  print (n)
```

Пересечение списков (поиск одинаковых элементов в двух списках)

Если даны два списка и необходимо найти их совпадающие элементы («область пересечения списков»), т.е. те которые есть и в одном списке и в другом, то это легко можно сделать с помощью цикла for языка программирования Python.

Код ниже подходит для списков, содержащих неповторяющиеся значения в самих себе. Иначе в результирующем списке могут появится одинаковые элементы.

```
1.  a = [5,[1,2],2,'r',4,'ee']
2.  b = [4,1,'we','ee',2,'r',[1,2]]
3.  c = []
```

```
    for i in a:
        for j in b:
            if i == j:
                c.append(i)
                break

print (c)
```

Алгоритм поиска очень прост. Берется первый элемент первого списка (внешний цикл for) и последовательно сравнивается с каждым элементом второго списка (вложенный цикл for). В случае совпадения (равенства) значений элемент добавляется в третий список, который до этого был создан. Команда break здесь служит для выхода из цикла, т.к. в случае совпадения дальнейший поиск при данном значении i бессмысленный.

Решето Эратосфена - алгоритм определения простых чисел

Описание алгоритма

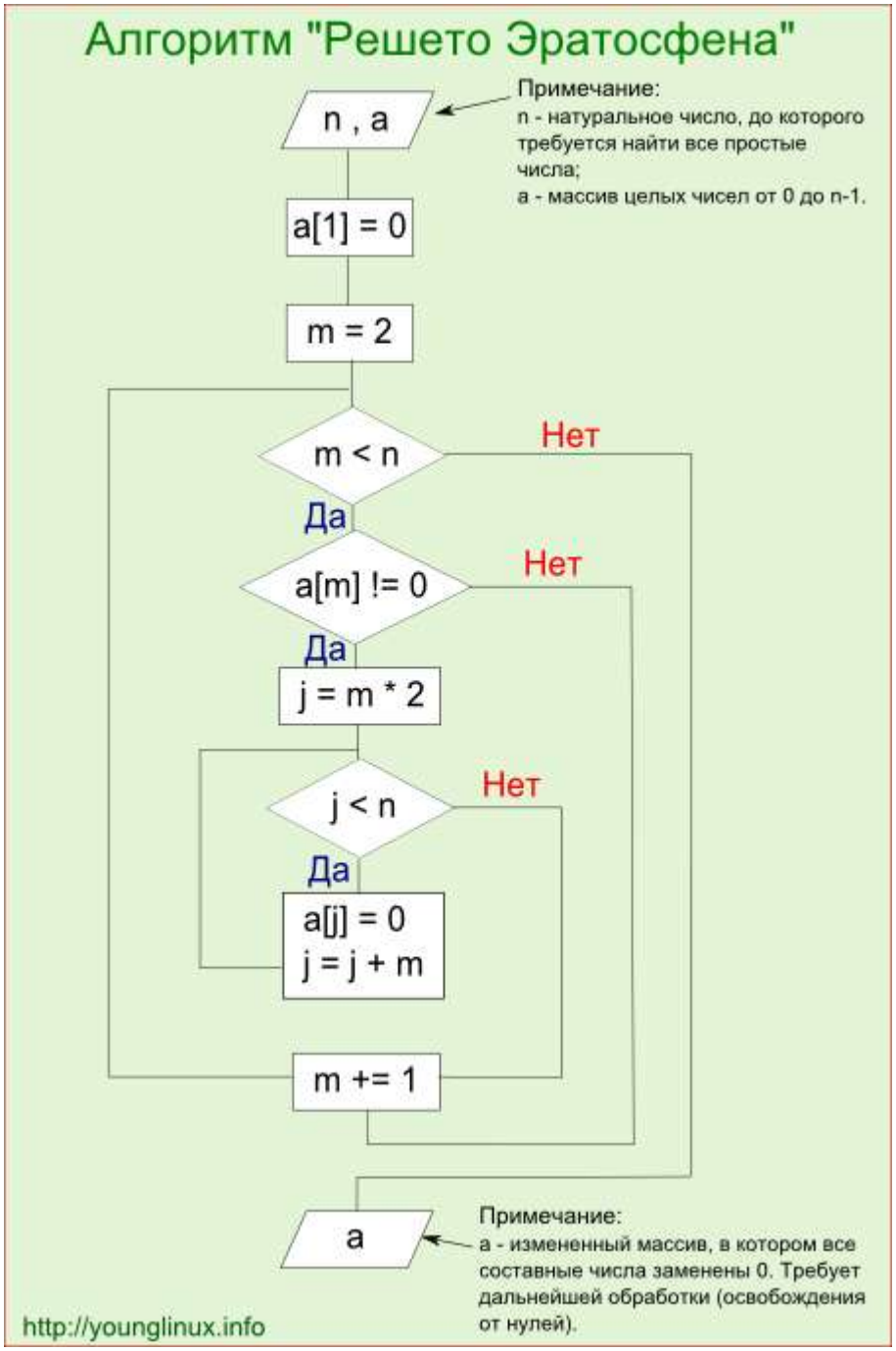
Решето Эратосфена – это алгоритм нахождения простых чисел до заданного числа n. В процессе выполнения данного алгоритма постепенно отсеиваются составные числа, кратные простым, начиная с 2.

Исходный код на Python

```
n = int(input("вывод простых чисел до числа ... "))
a = [0] * n # создание массива с n количеством элементов
for i in range(n): # заполнение массива ...
    a[i] = i # значениями от 0 до n-1

# вторым элементом является единица, которую не считают простым числом
# забиваем ее нулем.
a[1] = 0

m = 2 # замена на 0 начинается с 3-го элемента (первые два уже нули)
while m < n: # перебор всех элементов до заданного числа
    if a[m] != 0: # если он не равен нулю, то
        j = m * 2 # увеличить в два раза (текущий элемент простое число)
        while j < n:
            a[j] = 0 # заменить на 0
            j = j + m # перейти в позицию на m больше
    m += 1
```

```
18.
19.  # вывод простых чисел на экран (может быть реализован как
   угодно)
20.  b = []
21.  for i in a:
22.      if a[i] != 0:
23.          b.append(a[i])
24.
25.  del a
26.  print (b)
```

Блок-схема



Сортировка выбором (поиск минимума и перестановка)

Описание алгоритма

1. Найти наименьшее значение в исходном множестве.
2. Записать его в начало множества, а первый элемент - на место наименьшего.
3. Снова найти наименьший элемент в оставшемся множестве и переместить его на второе место. Второй элемент при этом перемещается на освободившееся место.
4. Продолжать выполнять поиск и обмен, пока не будет достигнут конец множества.

Исходный код на Python

Как найти наименьшее значение в списке?

```
1.   s = [2,4,1,3]  #подопытный список
2.
3.   m = 0   #индекс первого элемента
4.   i = 1     #индекс второго элемента
5.
6.   while i < len(s): #пока индекс меньше длины строки
7.      if s[i] < s[m]: # если значение под индексом i меньше, чем под m,
8.           m = i # то присвоить m индекс i
9.       i += 1 # увеличить i на единицу
10.
11.  print (s[m]) # вывести значение элемента под индексом m
```

Обратите внимание на строку while i < len(s):. Не нужно писать <=, т.к. индексация начинается с нуля. Это значит, что когда i равен 3, то мы обращаемся к 4-му элементу списка (в примере, это как раз конец строки).

Как поменять два значения в списке местами?

```
1.   s = [2,4,9,1,3,7,5]  #подопытный список
2.   # требуется поменять местами первый и четвертый элементы
3.   m = 0   #индекс первого элемента
4.   i = 3   #индекс четвертого элемента
5.
6.   t = s[m] # сохраняется значение под индексом m
7.   s[m] = s[i] # на его место записывается значение под индексом i
8.   s[i] = t # на место значения под индексом i записывается ранее сохраненное значение под индексом m
```

Полный алгоритм сортировки выбором

```
s = [2,4,8,1,0,3,9,5,7,6]
print (s)

#в переменной k хранится индекс элемента, подлежащего обмену (двигаемся слева на право)
k = 0
while k < len(s) - 1: #-1, т.к. последний элемент обменивать уже не надо
    m = k #в m хранится минимальное значение
    i = k + 1 #откуда начинать поиск минимума (элемент следующий за k)
    while i < len(s):
        if s[i] < s[m]:
            m = i
        i += 1
    t = s[k]
    s[k] = s[m]
    s[m] = t
    k += 1 #переходим к следующему значению для обмена

print(s)
```

Оформление алгоритма в виде функции и пример использования цикла for

```
def mymin(mylist):
    for k in range(len(mylist) - 1):
        m = k
        i = k + 1
        while i < len(mylist):
            if mylist[i] < mylist[m]:
                m = i
            i += 1
        t = mylist[k]
        mylist[k] = mylist[m]
        mylist[m] = t
```

Сортировка методом пузырька

Описание алгоритма

1. Проход по неупорядоченному множеству (например, списку). При этом, если последующий элемент меньше предыдущего, то они меняются местами. В итоге, самый «тяжелый» элемент оказывается в конце множества. Пример: дано: 4, 7, 3, 6, 1 меняем: 7 и 3, затем 7 и 6, затем 7 и 1 результат: 4, 3, 6, 1, 7
2. Проход по множеству, исключая последний элемент, т.к. он уже отсортирован. При проходе обмен меньшего последующего на больший предыдущий.
3. Количество проходов по множеству равно количеству элементов минус 1. Последний проход не нужен, т.к. остается один элемент, и его значение меньше остальных. Он «всплыл» в результате предыдущих проходов.

Исходный код на Python

```
1.   li = [5,2,7,4,0,9,8,6]
2.   n = 1
3.   while n < len(li):
4.        for i in range(len(li)-n):
5.             if li[i] > li[i+1]:
6.                  li[i],li[i+1] = li[i+1],li[i]
7.        n += 1
```

Переменная **n** здесь служит для того, чтобы прервать проходы по списку, как только ее значение приблизится к размеру длины строки. Также цикл **for** благодаря **n** сокращается при каждом последующем проходе по **while**. Это оптимизирует алгоритм: последние элементы не просматриваются. Элементы меняются местами лишь в случае, если предыдущий элемент больше последующего.

Числа Фибоначчи (вычисление с помощью цикла while и рекурсии)

Числа Фибоначчи – это ряд чисел, в котором каждое последующее число равно сумме двух предыдущих: 1, 1, 2, 3, 5, 8, 13 и т.д.
Формула:
F1 = 1
F2 = 1
Fn = Fn-1 + Fn-2
Пример вычисления:

F3 = F2 + F1 = 1 + 1 = 2
F4 = F3 + F2 = 2 + 1 = 3
F5 = F4 + F3 = 3 + 2 = 5
F6 = F5 + F4 = 5 + 3 = 8
и т.д.

Вычисление n-го числа ряда Фибоначчи с помощью цикла

Алгоритм

1. Ввести два начальных значения ряда (fib1 и fib2).
2. Ввести номер определяемого элемента.
3. Выполнять нижеследующие действия количество раз, равное по величине номеру определяемого элемента, уменьшенному на две единицы (т.к. первое и второе значение ряда уже известны).
    a. Сложить fib1 и fib2, присвоив результат третьей переменной (fib_sum).
    b. Поменять начальные значения: fib1 = fib2, а fib2 = fib_sum

Код на Python

```
1.  fib1 = 1
2.  fib2 = 1
3.
4.  n = input("Значение какого элемента ряда \
5.  Фибоначчи вы хотите узнать? ")
6.  n = int(n) # преобразование в целое число
7.
8.  i = 2
9.  while i < n:
10.     fib_sum = fib2 + fib1
11.     fib1 = fib2
12.     fib2 = fib_sum
13.     i += 1
14.
15. print (fib_sum)
```

Рекурсивное вычисление n-го числа ряда Фибоначчи

Алгоритм

1. Если n = 1 или n = 2, вернуть в вызывающую ветку единицу (т.к. первый и второй элементы ряда Фибоначчи равны единице).

2. Во всех остальных случаях вызвать эту же функцию с аргументами n-1 и n-2. Результат двух вызовов сложить и вернуть в вызывающую ветку программы.

Код на Python

```
1.   def fib(n):
2.       if n==1 or n==2:
3.           return 1
4.       return fib(n-1) + fib(n-2)
```