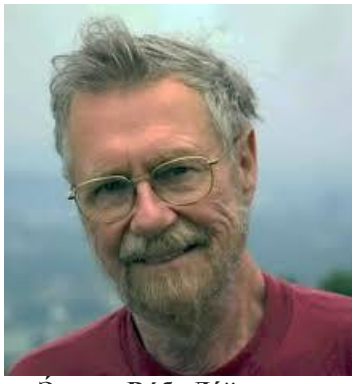
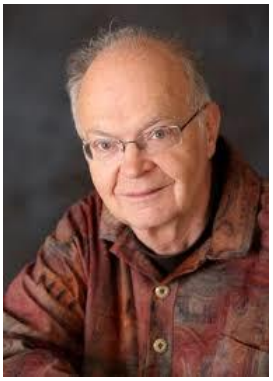
Университетские субботы-2021. Информатика.

Юлия Борисовна Буркатовская, ОИТ

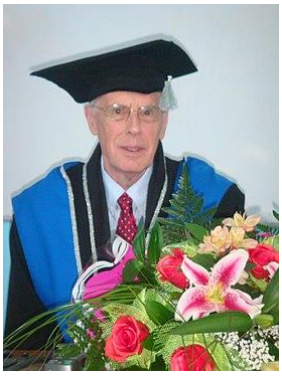
Эффективные алгоритмы



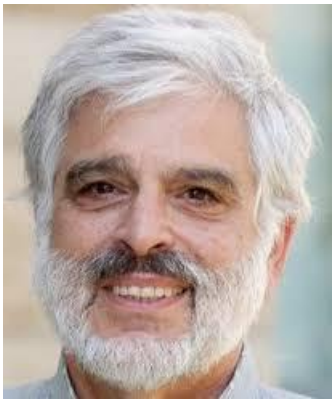
Э́дсгер Ви́бе Де́йкстра
(11 мая 1930— 6 августа 2002)



Дональд Эрвин Кнут (10 января 1938)

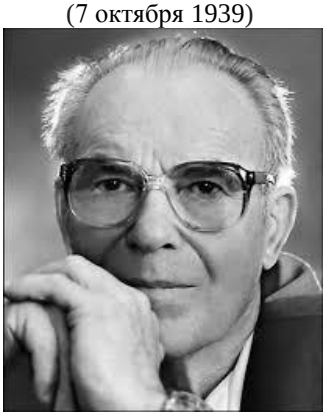


Джон Эдвард Хопкрофт
(7 октября 1939)



Джеффри Дэвид Ульман
(22 ноября 1942)

Альфред Ахо
(9 августа 1941)



Андрей Петрович Ершов
(19 апреля 1931 — 8 декабря 1988)

- *Дейкстра Э.* Дисциплина программирования.
- *Кнут Д.* Искусство программирования.
- *Кнут Д.* Конкретная математика.
- *А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман.* Структуры данных и алгоритмы.
- *А. Ахо, Дж. Ульман*. Теория синтаксического анализа, перевода и компиляции.
- *Хопкрофт, Джон, Мотвани, Раджив, Ульман, Джеффри, Д*. Введение в теорию автоматов, языков и вычислений.
- *Ершов А. П.* Введение в теоретическое программирование: Беседы о методе

Задания с сайтов

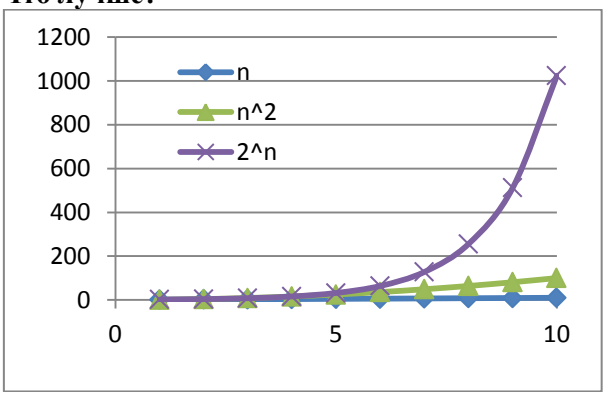
- https://kpolyakov.spb.ru/school/ege.htm
- https://informatics.msk.ru/
- https://inf-ege.sdamgia.ru/

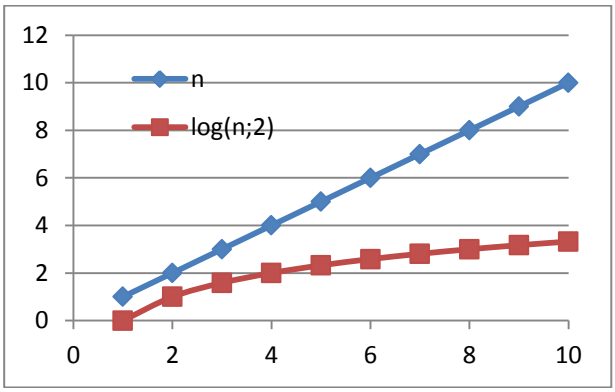
Характеристики алгоритмов:

- **Время** – если размер исходных данных n, то сколько элементарных операций понадобится для решения задачи? *Элементарная операция* – сложение, умножение, сравнение…
- **Память –** если размер исходных данных n, сколько данных мы храним? Одномерный массив – n, двумерный – n^2.

Время. *О-нотация*. Говорят, что вычислительная сложность алгоритма есть $O(F(n))$, если максимальное число элементарных операций не превосходит $CF(n)$, $F(n)$ – неубывающая функция от n (обычно n, n^2, n^α, $\log n$, 2^n ...), C – константа.

Что лучше?





n	$\lceil \log_2 n \rceil$	n^2	2^n
10	4	100	1024
100	7	10000	$\approx 10^{30}$
1000	10	1000000	$\approx 10^{301}$
10000	14	100000000	$\approx 10^{3030}$

Выводы:

- Программисты любят логарифм, но не всегда нам так везет.
- Линейный алгоритм лучше квадратичного, хотя на количестве данных до 1000 мы этого и не заметим, а вот больше…
- 2^n – это очень много. Избегаем по возможности, подходит только для небольших объемов данных.

Как понять, какая скорость?

- Линейный алгоритм – один цикл; либо есть цикл в цикле, но один из циклов не зависит от значения n.
- Квадратичный – двойной цикл, оба зависят от n.
- Логарифм – размерность данных каждый раз уменьшается наполовину.
- Экспоненциальная скорость – обычно это полный перебор вариантов.
- Если вы используете встроенные функции, понимайте, какая у них скорость. Если вы вызываете сортировку одной командой, это не значит, что выполняется одна операция.
- Это все кратко, общая теория сложнее.

Примеры задач с такими скоростями:

Линейная скорость – поиск элемента в неупорядоченном массиве. Везде сам поиск – перебор всех элементов массива с индексами от 0 до $n-1$. Один цикл.

Pascal

```
var n,i,x,f:integer;
   a:array[0..100] of integer;
begin
  readln(n);
  for i:=0 to n-1 do
      read(a[i]);
  readln(x);
  f:=-1;
  for i:=0 to n-1 do
      if (a[i]=x) then
        begin
          f:=i;
          break;
        end;
  if (f>-1) then
    writeln('a[',f,']=',x)
  else
    writeln(x,' is not in a');
end.
```

C++

```
#include <iostream>

using namespace std;

int main() {
    int n,x,i,f=-1;
    cin >> n;
    int a[n];
    for(i=0;i<n;i++)
        cin >> a[i];
    cin >> x;
    for(i=0;i<n;i++)
        if(a[i]==x)
        {
            f=i;
            break;
        }
    if (f>-1)
      cout<<"a["<<f<<"]="<<x;
    else
      cout<<x<<" is not in a";
return 0;
}
```

Python

```
n=int(input())
f=-1
a=list(map(int,input().split()))
x=int(input())
for i in range(n)
    if a[i]==x:
        f=i
        break
if f>-1:
    print('a[',f,']=',x)
else:
    print(x,' is not in a')
```

Квадратичная скорость – сортировка массива простыми алгоритмами (прямой выбор, прямая вставка, пузырек). В школе любят пузырек. Цикл в цикле (двойной цикл), оба цикла зависят от n.

Pascal

```
var n,i,x,j:integer;
   a:array[0..100] of integer;
begin
  readln(n);
  for i:=0 to n-1 do
      read(a[i]);
  for i:=2 to n do
    for j:=0 to n-i do
        if (a[j]>a[j+1]) then
        begin
            x:=a[j];
            a[j]:=a[j+1];
            a[j+1]:=x;
        end;
    for i:=0 to n-1 do
        write(a[i],' ');
end.
```

C++

```
#include <iostream>

using namespace std;

int main() {
    int n,x,i,j;
    cin >> n;
    int a[n];
    for(i=0;i<n;i++)
        cin >> a[i];
    for(i=1;i<n;i++)
        for(j=0;j<n-i;j++)
            if(a[j]>a[j+1])
            {
                x=a[j];
                a[j]=a[j+1];
                a[j+1]=x;
            }
    for(i=0;i<n;i++)
        cout << a[i] << ' ';
return 0;
}
```

Python

```
n=int(input())
a=list(map(int,input().split()))
for i in range(1,n):
    for j in range(0,n-i):
        if a[j]>a[j+1]:
            x=a[j]
            a[j]=a[j+1]
            a[j+1]=x
print(a)
```

Логарифмическая скорость – поиск элемента в упорядоченном массиве. Массив каждый раз делится на две части, искомое число находится в одной из них. Цикл продолжается, пока часть, в которой находится искомое число, не сузится до одного числа.

Pascal	C++	Python

var n,i,x,l,r,m:integer;
a:array[0..100] of integer;
begin
readln(n);
for i:=0 to n-1 do
read(a[i]);
readln(x);
l:=0;
r:=n-1;
while (l<r) do begin
m:=(l+r) div 2;
if (a[m]<x) then
l:=m+1
else
r:=m;
end;
if (a[l]=x) then
writeln('a[',l,']=',x)
else
writeln(x,' is not in a');
end.
	#include <iostream>
	
	using namespace std;
	
	int main() {
	int n,x,i,l,r,m;
	cin >> n;
	int a[n];
	for(i=0;i<n;i++)
	cin >> a[i];
	cin >> x;
	l=0;
	r=n-1;
	while(l<r)
	{
	m=(l+r)/2;
	if (a[m]<x)
	l=m+1;
	else
	r=m;
	}
	if (a[l]==x)
	cout<<"a["<<l<<"]="<<x;
	else
	cout<<x<<" is not in a";
	return 0;
	}
		n=int(input())
		f=-1
		a=list(map(int,input().split()))
		x=int(input())
		l=0
		r=n-1
		while l<r:
		m=(r+l)//2
		if a[m]<x:
		l=m+1
		else:
		r=m
		if a[l]==x:
		print('a[',l,']=',x)
		else:
		print(x,' is not in a')

- Полиномиальная сложность алгоритма: $O(n^{\alpha})$, неполиномиальная $O(2^n), O(n!)$ (NP-полные задачи.
- Сложность зависит от задачи.
- Сортировку не выполнишь за линейное время (если это не массив, в котором могут быть только целые числа из ограниченного диапазона, тогда просто просматриваем массив и считаем, сколько каких чисел).
- Медленные сортировки $O(n^2)$, быстрые $O(n \log n)$.
- Поиск в неупорядоченном массиве имеет линейную сложность, в упорядоченном – логарифмическую.
- Для других структур (списки, кучи, деревья, файлы…) существуют другие алгоритмы, соответственно сложность другая, для каждой структуры своя.
- Неполиномиальная сложность обычно связана с перебором всех вариантов (подмножеств, перестановок…).
- Если сложность принципиально снизить нельзя, можно побороться за константу.

Идея **линейного поиска** в неупорядоченном массиве: начинаем с нулевого элемента, на каждом шаге цикла увеличиваем индекс, проверяем, не вышли ли мы за границу массива и сравниваем элемент массива с образцом. Итого: на каждом шаге два сравнения, одно сложение.

Идея **барьерного поиска** в неупорядоченном массиве: искомый элемент искусственно добавляем в конец массива. Теперь мы уверены, что он в массиве есть, и мы на него наткнемся, поэтому выход за границу массива проверять не нужно, только увеличиваем индекс и сравниваем элемент массива с образцом. Итого: на каждом шаге одно сравнение, одно сложение.

Линейный поиск

Pascal	C++	Python

var n,i,x,f:integer;
  a:array[0..100] of integer;
begin
  readln(n);
  for i:=0 to n-1 do
  read(a[i]);
  readln(x);
  f:=-1;
  for i:=0 to n-1 do
  if (a[i]=x) then
  begin
  f:=i;
  break;
  end;
  if (f>-1) then
  writeln('a[',f,']=',x)
  else
  writeln(x,' is not in a');
end.
	#include <iostream>
	
	using namespace std;
	
	int main() {
	  int n,x,i,f=-1;
	  cin >> n;
	  int a[n];
	  for(i=0;i<n;i++)
	  cin >> a[i];
	  cin >> x;
	  for(i=0;i<n;i++)
	  if(a[i]==x)
	  {
	  f=i;
	  break;
	  }
	  if (f>-1)
	  cout<<"a["<<f<<"]="<<x;
	  else
	  cout<<x<<" is not in a";
	return 0;
	}
		n=int(input())
		f=-1
		a=list(map(int,input().split()))
		x=int(input())
		for i in range(n)
		  if a[i]==x:
		  f=i
		  break
		if f>-1:
		  print('a[',f,']=',x)
		else:
		  print(x,' is not in a')

Обратите внимание, в C++ мы явно прописываем изменение переменной и проверку на принадлежность элементу массива в цикле

```
for(i=0;i<n;i++)
```

В Pascal и Python мы это не пишем, но и проверка, и изменение переменной все равно выполняются. Делается одно и то же, синтаксис разный.

Барьерный поиск

Pascal	C++	Python

var n,i,x:integer;
  a:array[0..100] of integer;
begin
  readln(n);
  for i:=0 to n-1 do
  read(a[i]);
  readln(x);
  a[n]:=x;
  i:=0;
  while (a[i]<>x) do
  i:=i+1;
  if (i<n) then
  writeln('a[',i,']=',x)
  else
  writeln(x,' is not in a');
end.
	#include <iostream>
	
	using namespace std;
	
	int main() {
	  int n,x,i;
	  cin >> n;
	  int a[n+1];
	  for(i=0;i<n;i++)
	  cin >> a[i];
	  cin >> x;
	  a[n]=x;
	  i=0;
	  while (a[i]!=x)
	  i++;
	  if (i<n)
	  cout<<"a["<<i<<"]="<<x;
	  else
	  cout<<x<<" is not in a";
	return 0;
	}
		n=int(input())
		a=list(map(int,input().split()))
		x=int(input())
		a.append(x);
		i=0;
		while a[i]!=x:
		  i+=1
		if i<n:
		  print('a[',i,']=',x)
		else:
		  print(x,' is not in a')

Обратите внимание, здесь в Pascal и Python не используем цикл `for`, иначе в нем будет проверяться выход за границу массива и идея повышения скорости пропадет. В C++ можно, вместо

```
while (a[i]!=x)
        i++;
```

можно написать

```
for(i=0;a[i]!=x;i++);
```

Как сделать алгоритм эффективным по скорости? Разбираемся с математикой.

https://informatics.msk.ru/

Пример 1. Дан массив, состоящий из целых чисел. Нумерация элементов начинается с 0. Напишите программу, которая выведет элементы массива, номера которых четны (0, 2, 4...). (Для Python дан список).

Как решает 70% учеников?

Pascal	C++	Python

`for i:=0 to n-1 do`
`  if (i%2=0) then`
`  write(a[i],' ');`
	`for(i=0;i<n;i++)`
	`  if(i%2==0)`
	`  cout<<a[i]<<' ';`
		`for i in range(n):`
		`  if i%2==0:`
		`  print(a[i])`

Как решает 10% учеников?

Pascal	C++	Python

`m=(n-1) div 2;`
`for i:=0 to m do`
`  write(a[2*i],' ');`
	`for(i=0;i<n;i+=2)`
	`  cout<<a[i]<<' ';`
		`for i in range(0,n,2):`
		`  print(a[i])`

В хорошем решении те элементы, которые не нужны, даже не рассматриваются.

Остальные 20% решают что-то свое.

Пример 2. Выведите все точные квадраты натуральных чисел, не превосходящие данного числа N.

Как решает 60% учеников? Запускает цикл от 1 до N, проверяет, является ли число полным квадратом. Сложность $O(N)$.

Хорошее решение. Генерировать квадраты. Сложность $O(\sqrt{N})$.

Pascal	C++	Python

`M:=1;`
`while (m*m<=N) do begin`
`  write(m*m,' ');`
`  m:=m+1;`
`end;`
	`for(m=1;m*m<=N;m++)`
	`  cout<<m*m<<' ';`
		`m=1`
		`while m*m<=N:`
		`  print(m*m)`
		`  m+=1`

Пример 3. Улитка ползёт по вертикальному шесту высотой h метров, поднимаясь за день на a метров, а за ночь спускаясь на b метров. На какой день улитка доползёт до вершины шеста?

Решение. Есть формула.

$$d = \left\lceil \frac{h-a}{a-b} \right\rceil + 1$$

Пример 4. На столе лежит кучка из N спичек. Двое играют в такую игру. За один ход разрешается взять из кучки одну, две или три спички, так чтобы оставшееся количество спичек не было простым числом (Например, можно оставить в кучке 1 или 4 спички, но нельзя оставить 2 или 3). Выигрывает тот, кто забирает последнюю спичку. Требуется определить, кто из игроков имеет выигрышную стратегию.

Решение. Задача находится в разделе «динамическое программирование». На самом деле выигрывает тот, кто может подсунуть сопернику число спичек, кратное 4. То есть, если N кратно 4, выигрывает второй, иначе выигрывает первый.

Поиск делителей числа (нужно в задании 25)

$$N = m \times k$$
$$m \le \sqrt{N}$$
$$k \ge \sqrt{N}$$

Например, делители нескольких чисел

$N = 48$		$N = 25$		$N = 100$	
m	k	m	k	m	k
1	48	1	25	1	100
2	24	5	5	2	50
3	16			4	25
4	12			5	20
6	8			10	10

Таким образом, находя один делитель m (не превышающий корня), мы находим и второй $k = N/m$ (больший или равный корню). Поэтому при поиске делителей числа рассматриваем только числа, не превышающие $\sqrt{N}$.

Проверка, является ли число простым (там же)

У простого числа ровно два делителя, единица и само число.

40% школьников проверяет, делится ли число на 1 и на само себя, если да, пишут, что простое.

Еще 20% проверяют все числа, если находят делитель, пишут, что не простое. И так много раз. Потом выходят из цикла и пишут, что простое.

Как эффективно?

- Все простые числа, кроме 2, нечетны. Поэтому проверяем отдельно деление на 2. Если число делится на 2, то оно не простое.
- Если число не делится на 2, то оно не делится и на 4, 6…. Поэтому далее проверяем делимость только на нечетные числа.
- Если не нашли делителей, меньших или равных $\sqrt{N}$, то делителей нет вообще.

Задача 25
https://inf-ege.sdamgia.ru/

Пример 1 (27422). Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [174457; 174505], числа, имеющие ровно два различных натуральных делителя, не считая единицы и самого числа. Для каждого найденного числа запишите эти два делителя в две соседних столбца на экране с новой строки в порядке возрастания произведения этих двух делителей. Делители в строке также должны следовать в порядке возрастания.

Решение. Если делителей всего два, то один меньше корня, другой больше. При этом число не будет полным квадратом, поскольку у полного квадрата число различных делителей нечетно. Произведение этих делителей равно самому числу, поэтому просто перебираем числа в порядке возрастания. Начиная с `j=2`, проверяем, является ли `j` делителем `n`. Если да, увеличиваем счетчик делителей `c` и запоминаем делитель `j` в переменной `m`. Продолжаем проверки до тех пор, пока `c<2` и `j*j<=n`. У числа ровно два делителя, если мы нашли один (не превышающий корня из числа), при этом он не равен корню.

Pascal
```
var c,j,n,m:longint;
begin
  for n:=174457 to 174505 do
  begin
    c:=0;
    j:=2;
    while((c<2)and(j*j<=n)) do
    begin
      if (n mod j = 0) then
      begin
        c:=c+1;
        m:=j;
      end;
      j:=j+1;
    end;
    if((c=1)and(m*m<n)) then
      writeln(m,' ',n div m);
  end
end.
```

C++
```
#include <iostream>
using namespace std;
int main() {
  long n,j,c,m;
  for (n=174457;n<=174505;n++)
  {
     c=0;
     j=2;
     while((c<2)&&(j*j <=n))
     {
        if (n%j==0)
        {
            c+=1;
            m=j;
        }
        j+=1;
     }
     if ((c==1)&&(m*m<n))
     cout<<m<<' '<<n/m<< endl;
   }
}
```

Python
```
for n in range(174457,174506):
    c=0
    j=2
    while c<2 and j*j <=n:
        if n%j==0:
            c+=1
            m=j
    j+=1
    if c==1 and m*m<n:
        print(m,n//m)
```

Пример 2 (27850). Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [245 690; 245 756] простые числа. Выведите на экран все найденные простые числа в порядке возрастания, слева от каждого числа выведите его порядковый номер в последовательности. Каждая пара чисел должна быть выведена в отдельной строке.

Решение. Перебираем числа в порядке возрастания, причем только нечетные числа. Все четные не простые. Делим очередное число на 3,5,7…, т.е. на все нечетные число, не превосходящие корня из числа. Если найден хотя бы один делитель, число не простое.

Pascal
```
var c,j,n:longint;
begin
  n:=245691;
  while(n<245756) do
  begin
    c:=0;
    j:=3;
    while((c=0)and(j*j<=n)) do
    begin
      if (n mod j = 0) then
        c:=1;
      j:=j+2;
   end;
   if (c=0) then
     writeln(n-245689,' ',n);
   n+=2;
 end;
end.
```

C++
```
#include <iostream>
using namespace std;
int main() {
  long n,j,c;
  for(n=245691;n<=245755;n+=2)
  {
     c=0;
     j=3;
     while((c==0)&&(j*j <=n))
     {
        if (n%j==0)
          c=1;
        j+=2;
      }
      if (c==0)
cout<<n-245689<<' '<< n<<endl;
    }
}
```

Python
```
for n in range(245691,245756,2):
    c=0
    j=3
    while c==0 and j*j<=n:
        if n%j==0:
            c=1
        j+=2
    if c==0:
        print(n-245689,n)
```

Пример 3 (27854). Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [110203; 110245], числа, имеющие ровно четыре различных чётных натуральных делителя. Для каждого найденного числа запишите эти четыре делителя в четыре соседних столбца на экране с новой строки. Делители в строке должны следовать в порядке возрастания.

Решение. Есть условие четности делителей, поэтому напрямую использовать идею поиска делителей до корня, не получится. Простой вариант – перебирать только четные числа, и для них – только четные делители. Само число – один из этих делителей, число 2 – второй, так что надо найти еще два делителя. Остальные делители числа n не превосходят $n/2$. Вариант посложнее – искать, как и ранее, делители до корня из числа n. Пусть m – очередной делитель, сразу проверять парный ему делитель n/m, четный он или нечетный, четные делители заносить в список (четный делитель может быть парным как у четного, так и у нечетного). Список надо выводить в порядке возрастания, можно просто отсортировать в конце.

Пример 4 (27857). Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [84052; 84130], число, имеющее максимальное количество различных натуральных делителей, если таких чисел несколько — найдите минимальное из них. Выведите на экран количество делителей такого числа и само число.

Решение. А вот здесь отлично подходит идея искать делители, только не превосходящие корень из числа. Надо только учесть, что если число – полный квадрат, то у его корня нет парного делителя.

Пример 5 (29673). Назовём нетривиальным делителем натурального числа его делитель, не равный единице и самому числу. Например, у числа 6 есть два нетривиальных делителя: 2 и 3. Найдите все натуральные числа, принадлежащие отрезку [123456789; 223456789] и имеющие ровно три нетривиальных делителя. Для каждого найденного числа запишите в ответе его наибольший нетривиальный делитель. Ответы расположите в порядке возрастания.

Решение. Тут нам поможет математика – вместо того, чтобы искать такие числа (перебирая 10000000 чисел), лучше их генерировать. Если делителей нечетное число, то само число n – полный квадрат, $n = m \times m$. При этом m – не простое число, иначе n имело бы только один нетривиальный делитель. Пусть $m = a \times b$, тогда $n = a^2b^2$, и делителями n также будут являться $a, b, a^2, b^2, ab^2, a^2b$, но нам нужно еще всего лишь два делителя. Значит, $a = b, m = a^2, n = a^4$, a – простое число. Итак, нам нужны нечетные числа, которые являются четвертыми степенями простых чисел. Мы просто сгенерируем такие числа. Сначала найдем первое нечетное число a, которое при возведении в четвертую степень не меньше 123456789 (понятно, что $a > 100$). Затем будем рассматривать все нечетные числа, начиная с a, и если это число простое, и $a^4 \leq 223456789$, будем выводить a^4, а затем a^3 (наибольший делитель $n = a^4$).

C++	Python

`#include <iostream>`
 	
`using namespace std;`
 	
`int main() {`
`  long j,c,a;`
` for (a=101;a*a*a*a<123456789;a+=2);`
`  for(;a*a*a*a<=223456789;a+=2)`
`  {`
`  c=0;`
`  for(j=3;(c==0)&&(j*j <=a);j+=2)`
`  if (a%j==0)`
`  c=1;`
`  if (c==0)`
`  cout<<a*a*a*a<<' '<<a*a*a<<endl;`
`  }`
`}`
	`a=101`
	`while a*a*a*a < 123456789:`
	`  a+=2;`
	`while a*a*a*a <= 223456789:`
	`  c=0;`
	`  j=3;`
	`  while c==0 and j*j<=a:`
	`  if a%j==0:`
	`  c=1`
	`  j+=2`
	`  if c==0:`
	`  print(a*a*a*a,a*a*a)`
	`  a+=2`

Задача 17
https://inf-ege.sdamgia.ru/

Пример 1 (27414). Рассматривается множество целых чисел, принадлежащих числовому отрезку [1016; 7937], которые делятся на 3 и не делятся на 7, 17, 19, 27. Найдите количество таких чисел и максимальное из них. В ответе запишите два целых числа без пробелов и других дополнительных символов: сначала количество, затем максимальное число.

Решение. Задача решается перебором, но можно его сократить. Первое число в промежутке, кратное 3 – это 1017. Начиная с него, рассматриваем все числа с шагом 3, проверяем для них остальные условия. Делимость на 3 уже не проверяем.

Пример 2. (27610) Рассматривается множество целых чисел, принадлежащих числовому отрезку [4197; 9182], которые делятся на 5 и не делятся на 6, 10, 13, 16. Найдите количество таких чисел и максимальное из них. В ответе запишите два целых числа без пробелов и других дополнительных символов: сначала количество, затем максимальное число.

Решение. Число, которое делится на 5 и не делится на 10, оканчивается на 5. Первое такое число из промежутка – это 4205. Начинаем с него и идем с шагом 10, проверяем, что число не делится на 13, потому что на 6 и 16 наши числа делиться не могут. Так как числа перебираем в порядке возрастания, каждое следующее число больше предыдущего, и число, удовлетворяющее условиям, мы записываем в переменную m. В итоге там окажется максимальное число.

Pascal	C++	Python

var k,m,n: integer;
begin
  k:=0;
  n:=4205;
  while(n<9183) do
  begin
  if(n mod 13>0) then
  begin
  k:=k+1;
  m:=n;
  end;
  n:=n+10;
  end;
  writeln(k,m);
end.
	#include <iostream>
	using namespace std;
	int main() {
	  int n,k=0,m;
	  for(n=4205;n<9183;n+=10)
	  if(n%13>0)
	  {
	  k++;
	  m=n;
	  }
	  cout <<k<<m;
	}
		k=0
		for n in range(4205,9183,10):
		  if n%13>0:
		  k+=1
		  m=n
		print(k,m)

Пример 3 (27611). Рассматривается множество целых чисел, принадлежащих числовому отрезку [1813; 6861], которые делятся на 5 и не делятся на 6, 10, 15, 23. Найдите количество таких чисел и минимальное из них. В ответе запишите два целых числа без пробелов и других дополнительных символов: сначала количество, затем минимальное число.

Решение. Нужно минимальное число, так что числа логично перебирать в порядке убывания. Последнее, которое мы найдем, и будет минимальным. Как и в прошлой задаче, число должно оканчиваться на 5, значит, оно не делится на 6. Проверяем только неделимость на 15 и 23.

Задача 26. Системный администратор раз в неделю создаёт архив пользовательских файлов. Однако объём диска, куда он помещает архив, может быть меньше, чем суммарный объём архивируемых файлов. Известно, какой объём занимает файл каждого пользователя.

По заданной информации об объёме файлов пользователей и свободном объёме на архивном диске определите максимальное число пользователей, чьи файлы можно сохранить в архиве, а также максимальный размер имеющегося файла, который может быть сохранён в архиве, при условии, что сохранены файлы максимально возможного числа пользователей.

В первой строке входного файла находятся два числа: *S* — размер свободного места на диске (натуральное число, не превышающее 10 000) и *N* — количество пользователей (натуральное число, не превышающее 1000). В следующих *N* строках находятся значения объёмов файлов каждого пользователя (все числа натуральные, не превышающие 100), каждое в отдельной строке.

Запишите в ответе два числа: сначала наибольшее число пользователей, чьи файлы могут быть помещены в архив, затем максимальный размер имеющегося файла, который может быть сохранён в архиве, при условии, что сохранены файлы максимально возможного числа пользователей.

100 4
80
30
50
40

При таких исходных данных можно сохранить файлы максимум двух пользователей. Возможные объёмы этих двух файлов 30 и 40, 30 и 50 или 40 и 50. Наибольший объём файла из перечисленных пар — 50, поэтому ответ для приведённого примера:

2 50

Решение. Перебор не нужен. Задача решается **жадным алгоритмом** – если надо сохранить как можно больше файлов, сами файлы должны быть как можно меньше. Поэтому сохраняем в массив все объемы файлов. Затем можно упорядочить массив любым алгоритмом. Суммируем числа в массиве по возрастанию, пока сумма не превысил заданного числа *S*. Количество чисел в последней сумме, меньшей или равной *S*, и будет искомым количеством файлов. Затем надо попробовать заменять последний добавленный файл на больший, пока это возможно, чтобы найти максимальный файл, который можно сохранить.

C++

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
ifstream F;
F.open("26_demo.txt");
int s,n,i,j,x;
F>>s>>n;
int a[n];
for(i=0;i<n;i++)
{
    F >> a[i];
    j=i-1;
    x=a[i];
    for(j=i-1;(j>=0)&&(a[j]>x);j--)
        a[j+1]=a[j];
    a[j+1]=x;
}
int sum = 0,  k = 0,  Max = 0;
  for(i=0;(i<n) && (sum+a[i]<=s);i++)
      sum=sum+a[i];
  Max=a[i-1];
  k=i;
  summa=summa-Max;
  for(;(i<n) && (summa+a[i]<=s);i++)
      Max=a[i];
  cout << k << " " << Max;
  return 0;
}
```

Python

```
with open("26_demo.txt") as F:
s,n = map(int, F.readline().split())
a = []
for i in range(n):
v = F.readline()
    a.append(int(v))
a.sort()
Sum = 0
k = 0
Max = 0
i=0
while (i<n) and (Sum+a[i]<=s):
    Sum=Sum+a[i]
    i+=1
Max=a[i-1]
k=i
Sum=Sum-Max
while(i<n) and (Sum+a[i]<=s):
    Max=a[i]
    i+=1
print(k,Max)
```