Введение. Основные понятия.

Алгоритмы и структуры данных

Мулюгин Н. В. Кузнецов М. А.

03.09.2022

Информационные источники

1. А. В. Столяров. Введение в профессию. http://stolyarov.info

2. К. Владимиров. youtube.com/channel/UCvmBEbr9NZt7UEh9doI7_A

3. Ю. Окуловский. https://ulearn.me/

Информационные источники

4 Томас Х. Кормен. Алгоритмы: построение и анализ.

5 Томас Х. Кормен. Алгоритмы. Вводный курс.

6 Род Стивенс. Алгоритмы. Теория и практическое применение.

7 Никлаус Вирт. Алгоритмы и структуры данных.

8 Д. Кнут. Искусство программирования.

Алгоритм

Алгоритм представляет собой набор правил преобразования исходных данных(input) в выходные(output).

Алгоритм

Алгоритм представляет собой набор правил преобразования исходных данных(input) в выходные(output).

Алгоритмы строятся для решения **вычислительных задач**.

Алгоритм

Алгоритм представляет собой набор правил преобразования исходных данных(input) в выходные(output).

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

необязательные

Алгоритм

Алгоритм представляет собой набор правил преобразования исходных данных(input) в выходные(output).

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

1. Конечность.

необязательные

Алгоритм

Алгоритм представляет собой набор правил преобразования исходных данных(input) в выходные(output).

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

1. Конечность.
2. Дискретность.

необязательные

Алгоритм

Алгоритм представляет собой набор правил преобразования исходных данных(input) в выходные(output).

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

1. Конечность.
2. Дискретность.
3. Определенность.

необязательные

Алгоритм

Алгоритм представляет собой набор правил преобразования исходных данных(input) в выходные(output).

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

1. Конечность.
2. Дискретность.
3. Определенность.

необязательные

4. Правильность.

Алгоритм

Алгоритм представляет собой набор правил преобразования исходных данных(input) в выходные(output).

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

1. Конечность.
2. Дискретность.
3. Определенность.

необязательные

4. Правильность.
5. Завершаемость.

Алгоритм

Алгоритм представляет собой набор правил преобразования исходных данных(input) в выходные(output).

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

1. Конечность.
2. Дискретность.
3. Определенность.

необязательные

4. Правильность.
5. Завершаемость.
6. Массовость.

Сложность алгоритма

Время работы алгоритма - число элементарных операций, которые он выполняет.

Сложность алгоритма

Время работы алгоритма - число элементарных операций, которые он выполняет.

Время работы алгоритма в худшем случае (временная сложность) - максимальное время работы для входов данного размера.

Сложность алгоритма

Время работы алгоритма - число элементарных операций, которые он выполняет.

Время работы алгоритма в худшем случае (временная сложность) - максимальное время работы для входов данного размера.

Асимптотическая сложность алгоритма - оценка, характеризующая вид зависимости времени работы алгоритма от длины входа.

Сложность алгоритма

```cpp
#include<iostream> //headers
#include<string>
//g++ -Wall -Werror -g --std=c++20 complex_1.cpp
//g++ - compiler; -Wall - show all the errors;
//-Werror - treat warnings as errors
int main()//entry point
{
    std::string input;//variable with type string
    std::cin >> input;//standard input
    int n = input.size();
    int sum = 0;
    for( int i = 0; i < n; i++ )//n
        for( int j = 0; j < 2 * i; j++ )
        //01 0123 012345 0123456 012..2n-2
            sum++;//0 + 2 + 4 + 6 + ... 2(n-1)
    std::cout << sum << std::endl;
    return 0;//code of return
}
```

Переменные

0100

Переменные

a 4 [-7÷8]

0100

Переменные

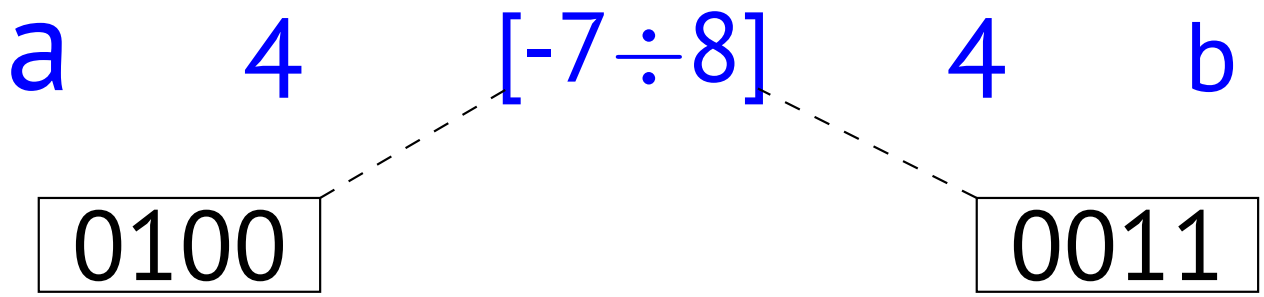
a 4 [-7÷8]

0100

Мы знаем тип a?

Мы знаем диапозон значений!

Переменные



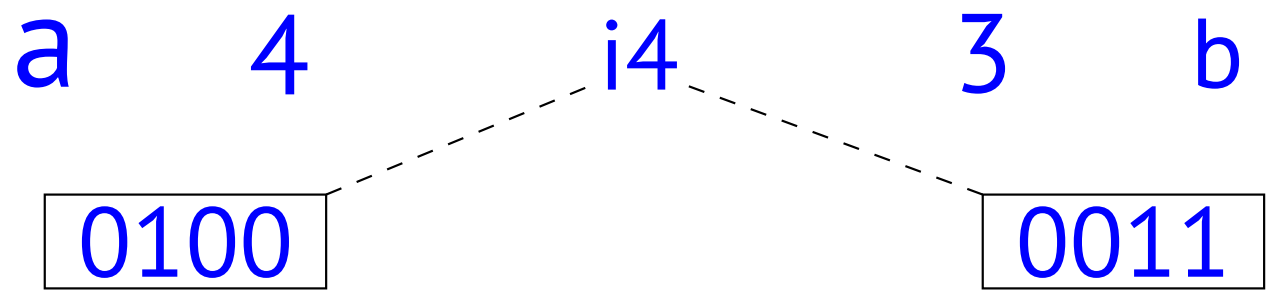
Мы знаем тип a?

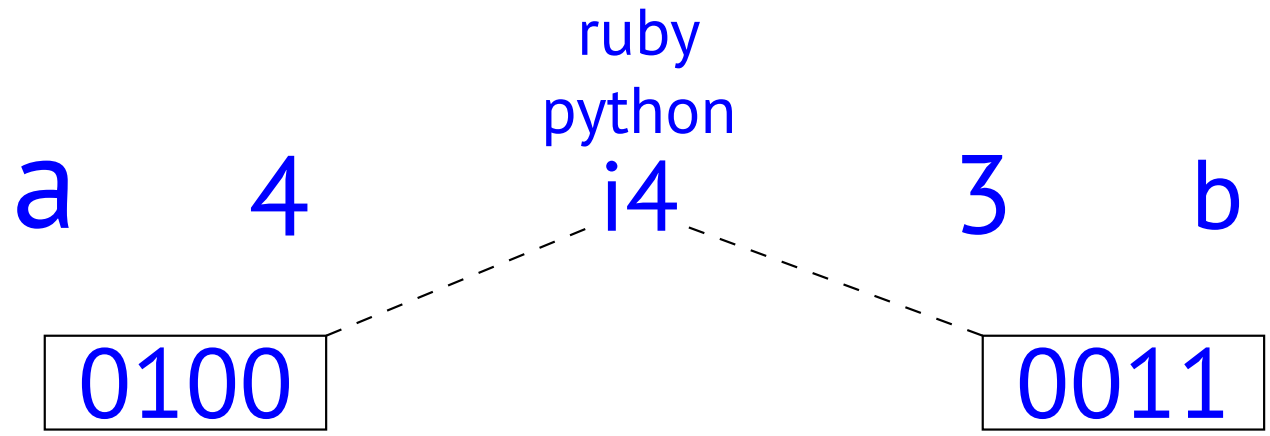
Мы знаем диапозон значений!

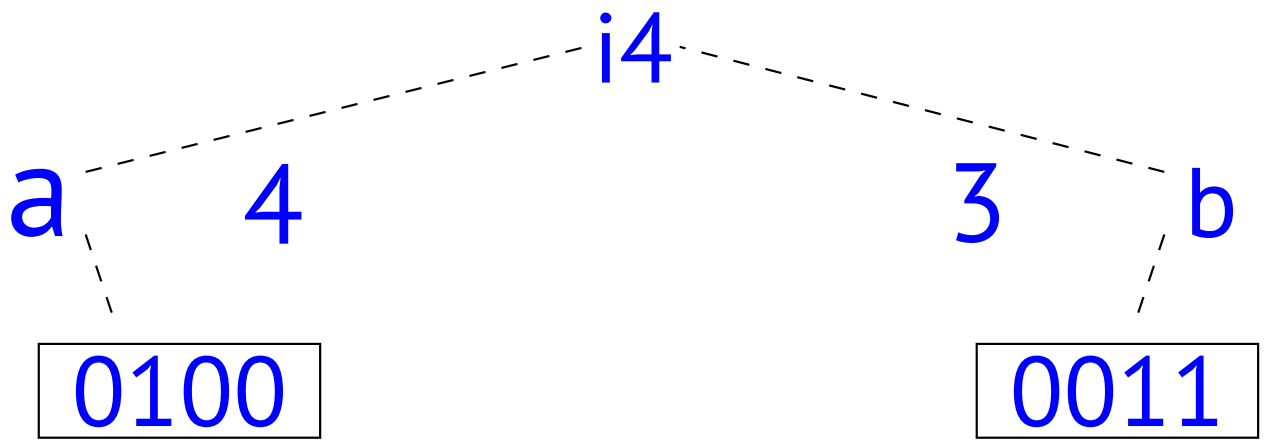
Мы не знаем операции!

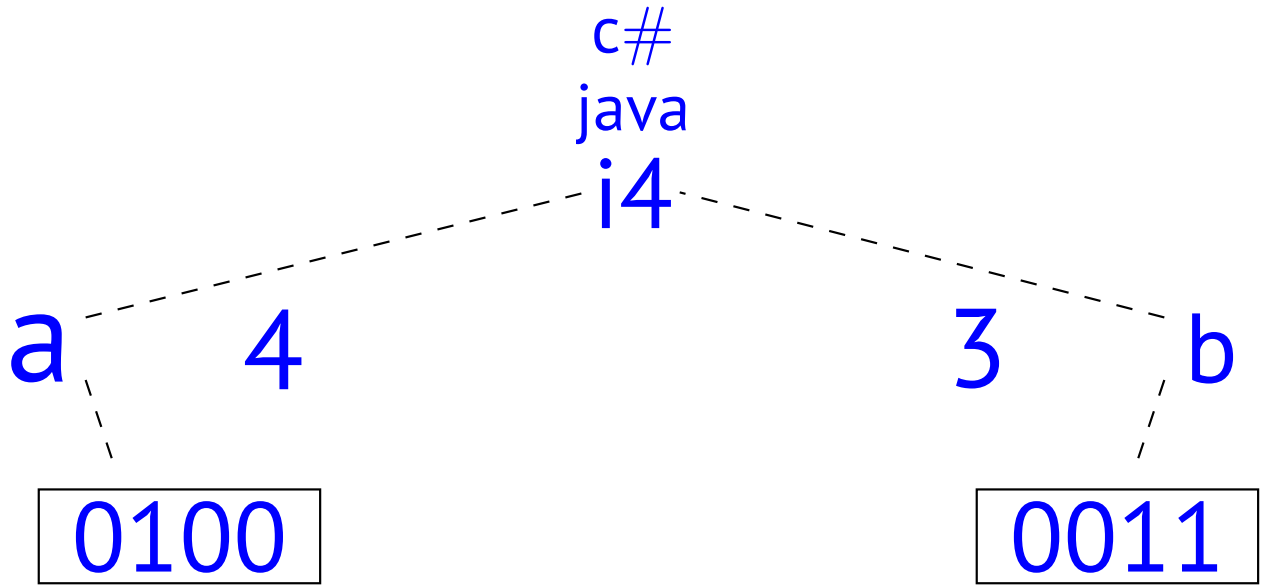
Типы

- Что такое тип?
- value type: диапозон значений объекта
- object type: совокупность операций над объектом
  - 5/2 даст 2 для int но 2.5 для double
- Назовем целочисленный арифметический тип i4

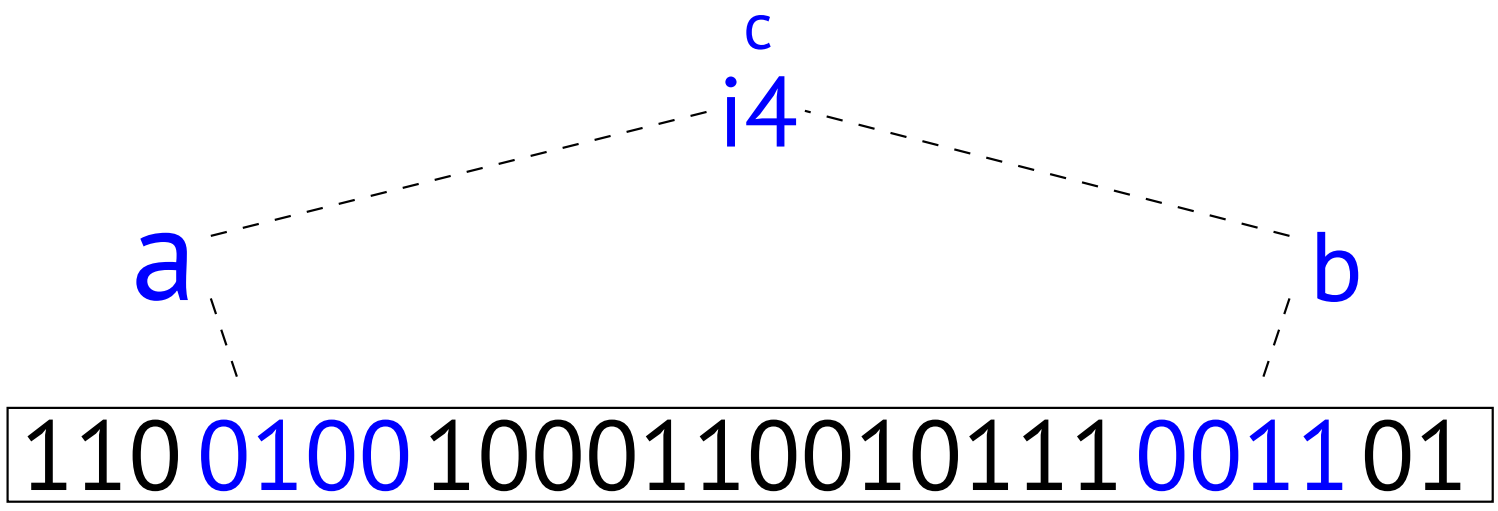

a
4
i4
3
b
0100
0011

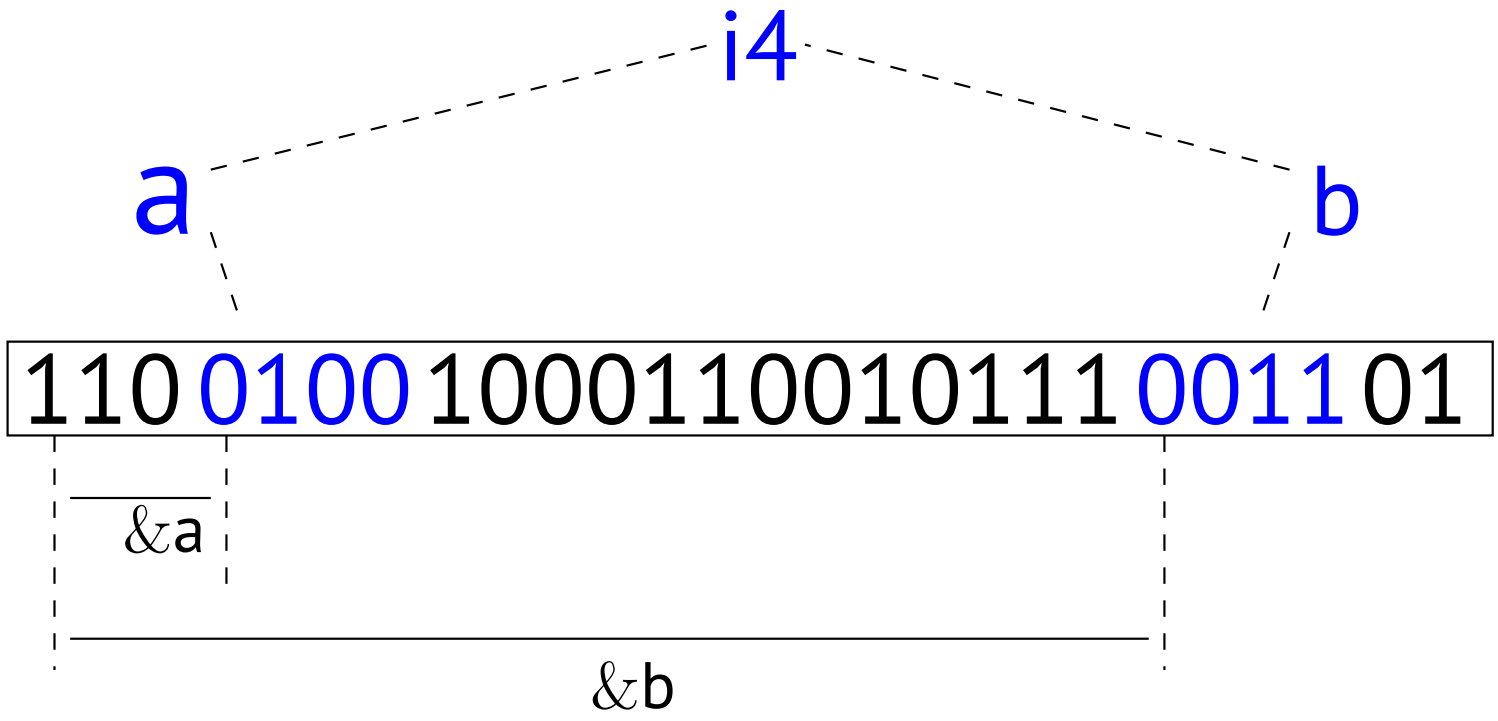

ruby
python
a
4
i4
3
b
0100
0011

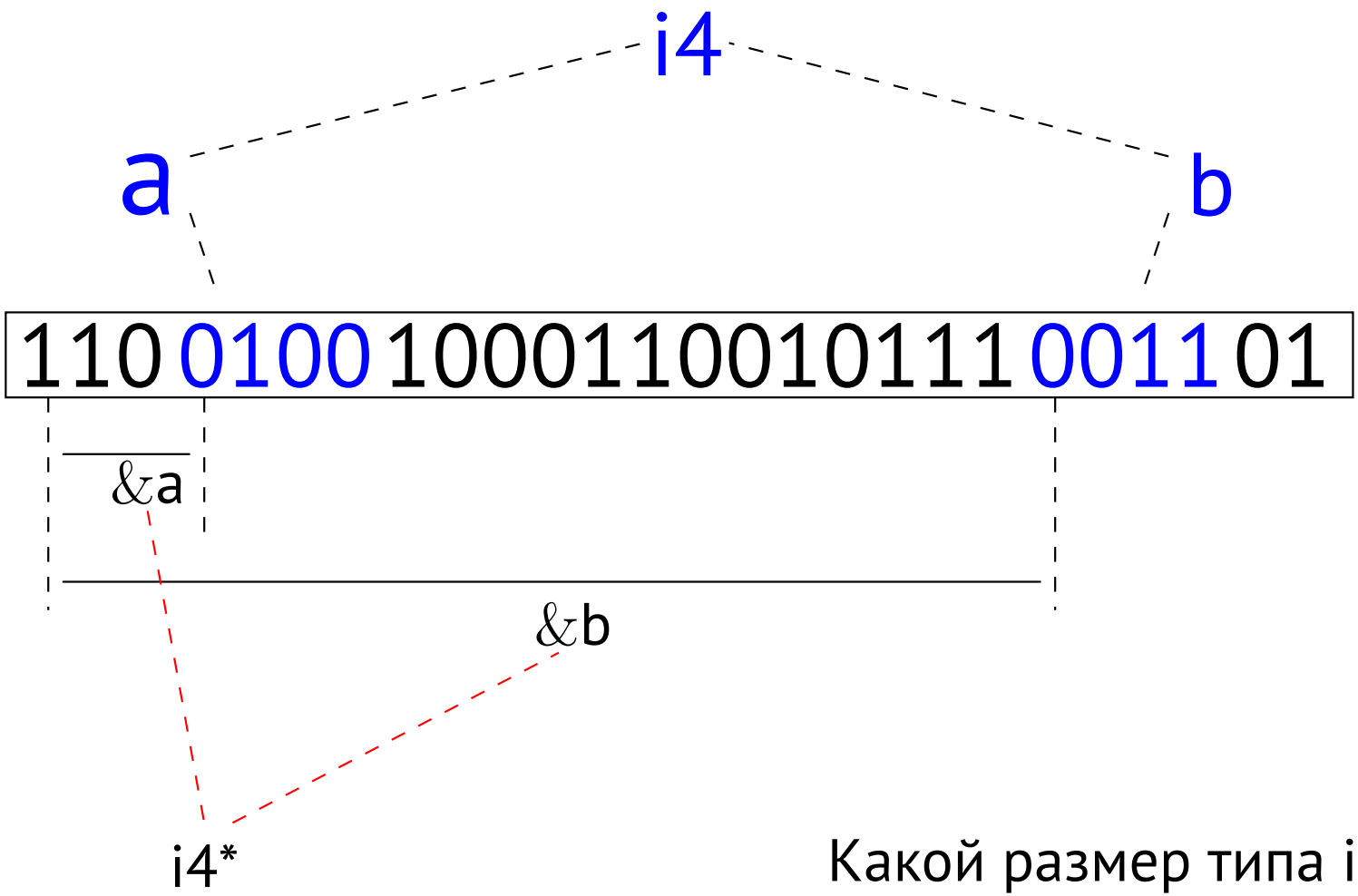

i4
a
4
3
b
0100
0011



Имя навсегда связано с типом.

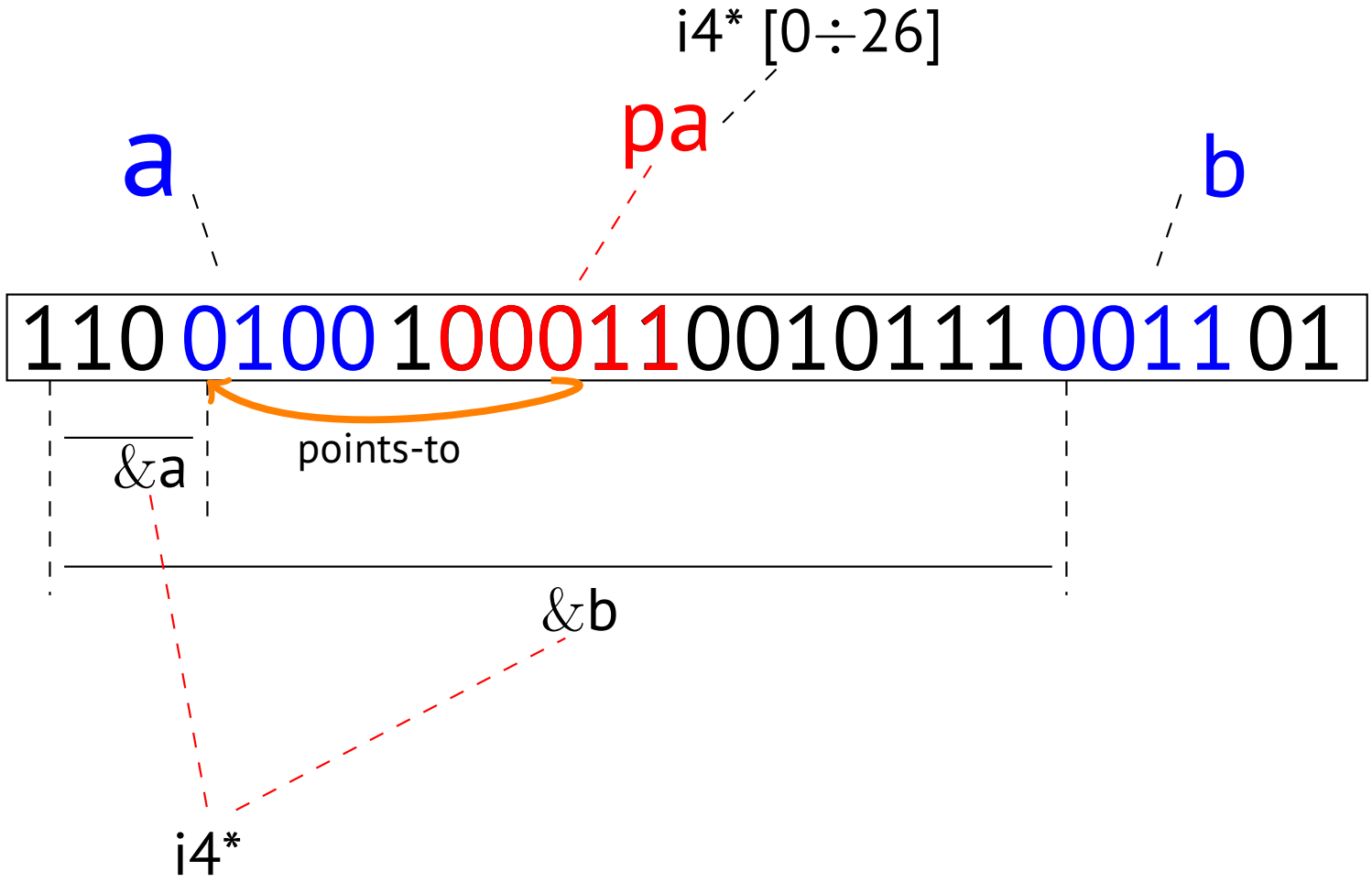

c
i4
a
b
110 0100 10001100 10111 0011 01


i4
a
b
110 0100 10001100 10111 0011 01
&a
&b



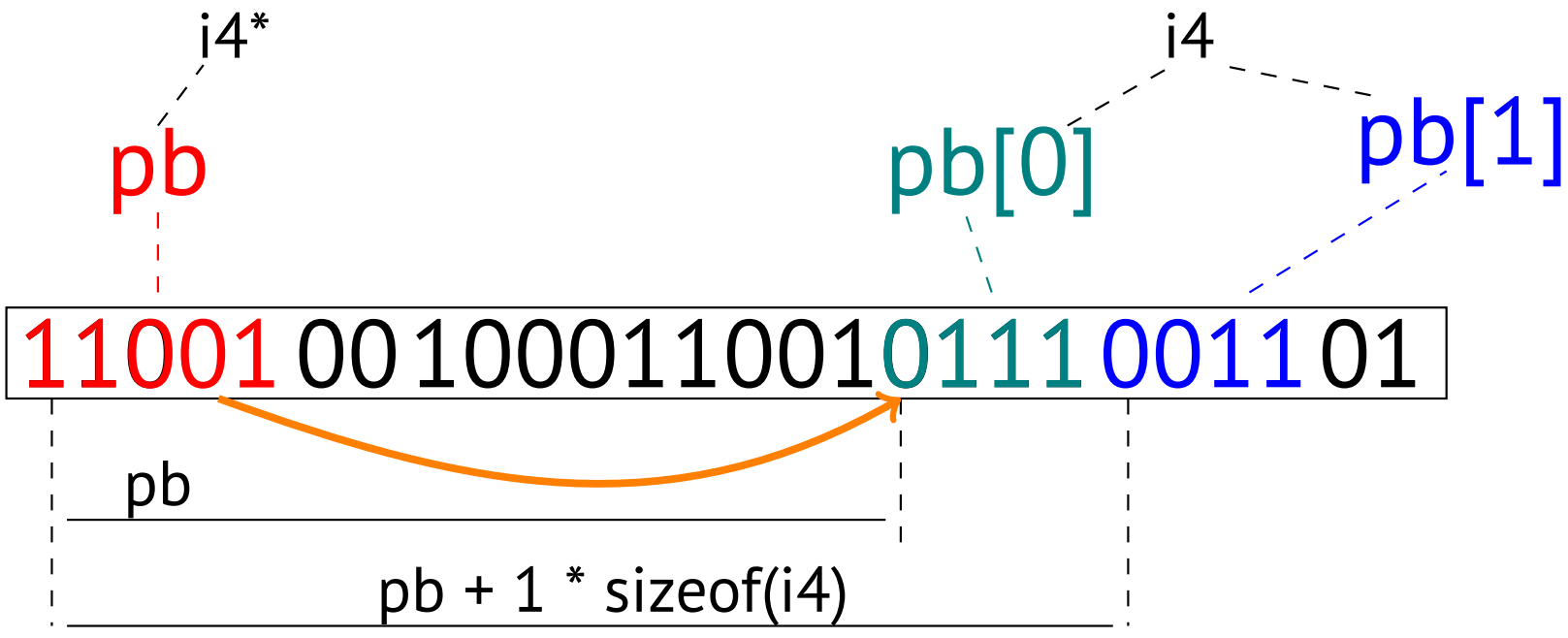
Какой размер типа i4* ?

Какое значение CHAR_BIT ?


i4* [0÷26]
pa
a
b
110 0100 1000 1100 1010 1110 0011 01
points-to
&a
&b
i4*

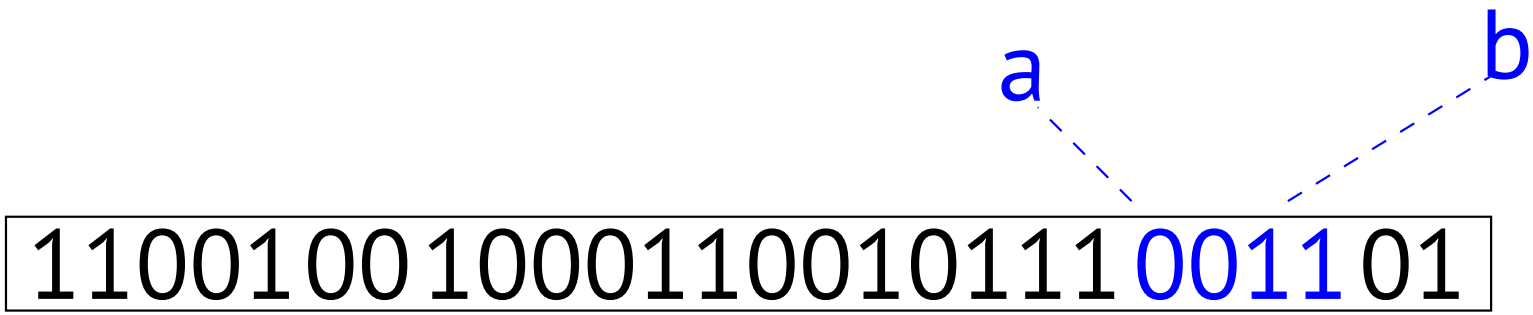
Указатели

- Если указатель это просто расстояние, то может быть нулевое расстояние?
  - nullptr
- p[2] == *(p+2)


i4*
i4
pb
pb[0]
pb[1]
11001 00 10001 10010 111 00 11 01
pb
pb + 1 * sizeof(i4)

Ссылки

- Два имени у одного объекта в C++.



Ссылки

Базовый синтаксис lvalue ссылок это одинарный амперсанд

```
1 int x;
2 int &y = x; // y - another name for x
```

Ссылки

Базовый синтаксис lvalue ссылок это одинарный амперсанд

```
1 int x;
2 int &y = x; // y - another name for x
```

Не путать со взятием адреса!

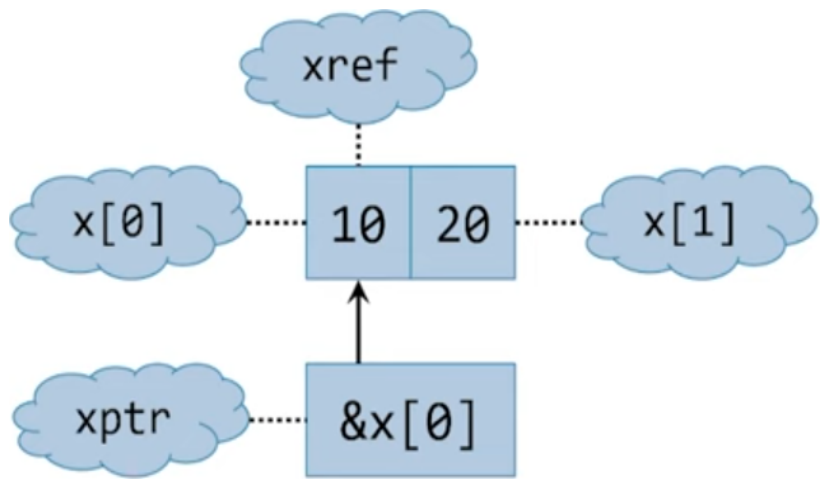
Ссылки

Базовый синтаксис lvalue ссылок это одинарный амперсанд

```
1 int x;
2 int &y = x; // y - another name for x
```

Не путать со взятием адреса!

```
1 int x[2] = {10, 20};
2 int &xref = x[0];
3 int *xref = &x[0];
4 xref += 1;
5 xptr += 1;
6 assert(xref == 11);
7 assert(xptr == 20);
8
```



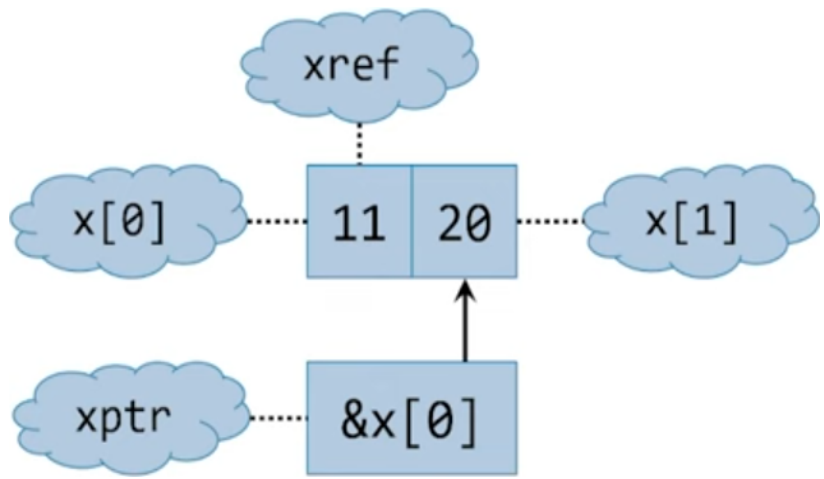
Ссылки

Базовый синтаксис lvalue ссылок это одинарный амперсанд

```
1 int x;
2 int &y = x; // y - another name for x
```

Не путать со взятием адреса!

```
1  int x[2] = {10, 20};
2  int &xref = x[0];
3  int *xref = &x[0];
4  xref += 1;
5  xptr += 1;
6  assert(xref == 11);
7  assert(xptr == 20);
8
```

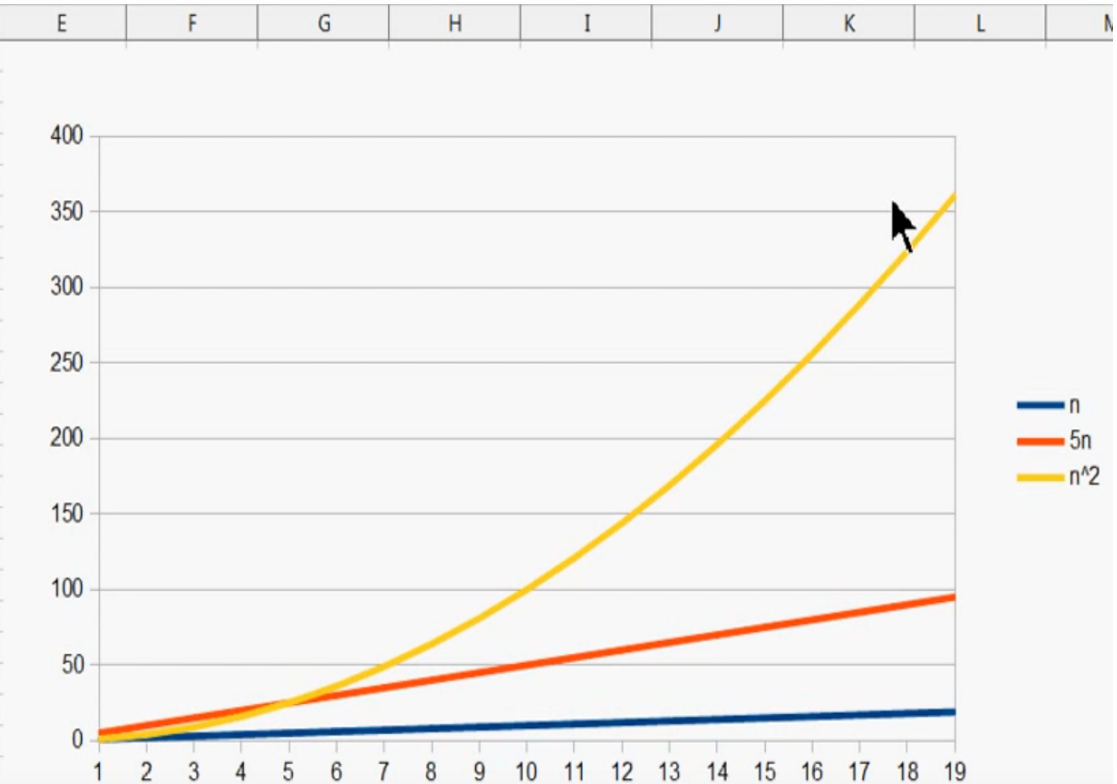


Оценка сложности алгоритма

```cpp
#include<iostream> //headers
#include<string>
//g++ -Wall -Werror -g --std=c++20 complex_1.cpp
//g++ - compiler; -Wall - show all the errors;
//-Werror - treat warnings as errors
int main()//entry point
{
    std::string input;//variable with type string
    std::cin >> input;//standard input
    int n = input.size();
    int sum = 0;
    for( int i = 0; i < n; i++ )//n
        for( int j = 0; j < 2 * i; j++ )
        //01 0123 012345 0123456 012..2n-2
            sum++;//0 + 2 + 4 + 6 + ... 2(n-1)
    std::cout << sum << std::endl;
    return 0;//code of return
}
```

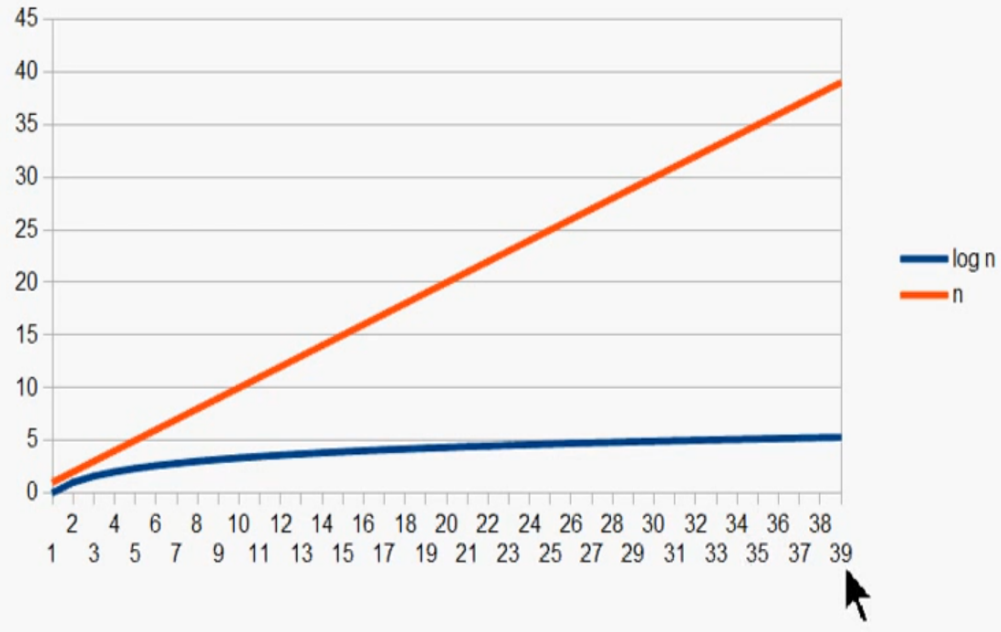
Оценка сложности алгоритма

	A	B	C
1	n	5n	n^2
2	1	5	1
3	2	10	4
4	3	15	9
5	4	20	16
6	5	25	25
7	6	30	36
8	7	35	49
9	8	40	64
10	9	45	81
11	10	50	100
12	11	55	121
13	12	60	144
14	13	65	169
15	14	70	196
16	15	75	225
17	16	80	256
18	17	85	289
19	18	90	324
20	19	95	361
21			
22			
23			
24			



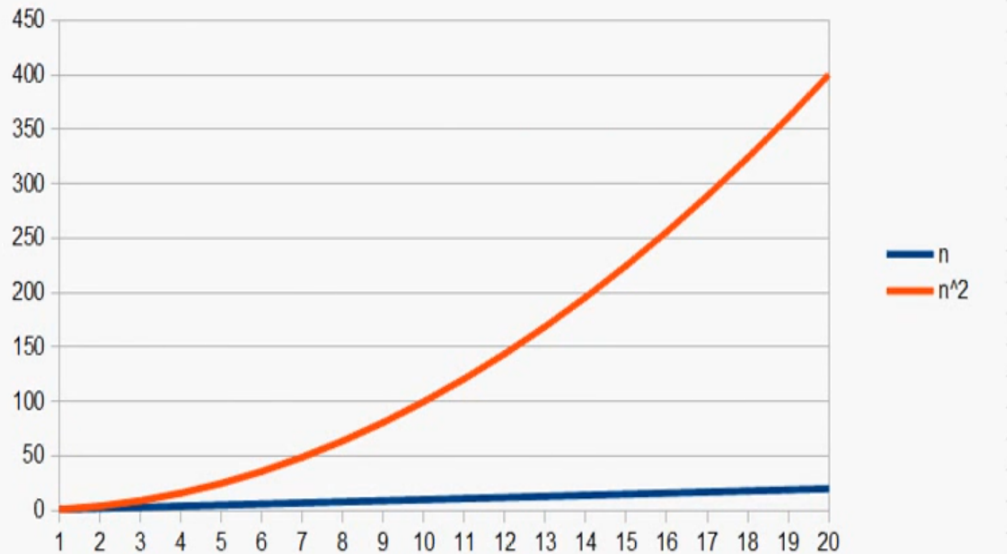
Оценка сложности алгоритма

	A	B	C	D
1	log n	n	n^2	2^n
2	0	1	1	2
3	1	2	4	4
4	1.584962501	3	9	8
5	2	4	16	16
6	2.321928095	5	25	32
7	2.584962501	6	36	64
8	2.807354922	7	49	128
9	3	8	64	256
10	3.169925001	9	81	512
11	3.321928095	10	100	1024
12	3.459431619	11	121	2048
13	3.584962501	12	144	4096
14	3.700439718	13	169	
15	3.807354922	14	196	
16	3.906890596	15	225	
17	4	16	256	
18	4.087462841	17	289	
19	4.169925001	18	324	
20	4.247927513	19	361	
21	4.321928095	20	400	
22	4.392317423	21		
23	4.459431619	22		
24	4.523561956	23		



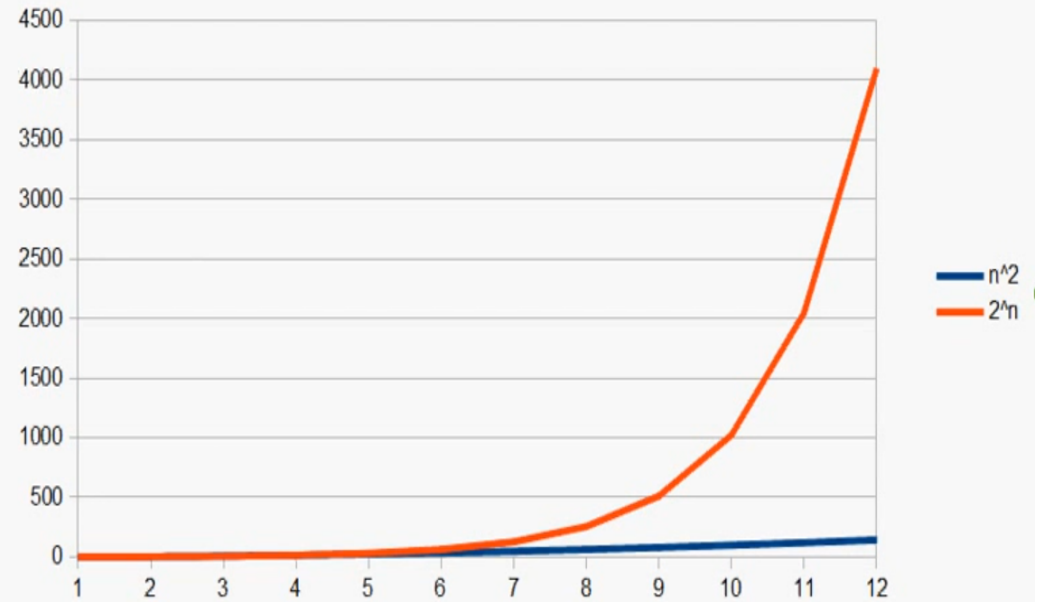
Оценка сложности алгоритма

	A	B	C	D
1	log n	n	n^2	2^n
2	0	1	1	2
3	1	2	4	4
4	1.584962501	3	9	8
5	2	4	16	16
6	2.321928095	5	25	32
7	2.584962501	6	36	64
8	2.807354922	7	49	128
9	3	8	64	256
10	3.169925001	9	81	512
11	3.321928095	10	100	1024
12	3.459431619	11	121	2048
13	3.584962501	12	144	4096
14	3.700439718	13	169	
15	3.807354922	14	196	
16	3.906890596	15	225	
17	4	16	256	
18	4.087462841	17	289	
19	4.169925001	18	324	
20	4.247927513	19	361	
21	4.321928095	20	400	
22	4.392317423	21		
23	4.459431619	22		
24	4.523561956	23		



Оценка сложности алгоритма

	A	B	C	D
1	log n	n	n^2	2^n
2	0	1	1	2
3	1	2	4	4
4	1.584962501	3	9	8
5	2	4	16	16
6	2.321928095	5	25	32
7	2.584962501	6	36	64
8	2.807354922	7	49	128
9	3	8	64	256
10	3.169925001	9	81	512
11	3.321928095	10	100	1024
12	3.459431619	11	121	2048
13	3.584962501	12	144	4096
14	3.700439718	13	169	
15	3.807354922	14	196	
16	3.906890596	15	225	
17	4	16	256	
18	4.087462841	17	289	
19	4.169925001	18	324	
20	4.247927513	19	361	
21	4.321928095	20	400	
22	4.392317423	21		
23	4.459431619	22		
24	4.523561956	23		



O-символика

$f(n) = o(g(n))$ $\forall k > 0\ \exists n_0\ \forall n > n_0$ $f(n) < k \cdot g(n)$ $\Leftrightarrow \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$ $f(n) \prec g(n)$

$f(n) = O(g(n))$ $\exists k > 0\ \exists n_0\ \forall n > n_0$ $f(n) < k \cdot g(n)$ $\Leftrightarrow \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ $f(n) \preceq g(n)$

$f(n) = \Theta(g(n))$ $\exists k_1 k_2 > 0\ \exists n_0$ $\forall n > n_0$ $k_1 \cdot g(n) < f(n)$ $f(n) < k_2 \cdot g(n)$ $\Leftarrow \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = c$ $f(n) \approx g(n)$

```cpp
#include<iostream> //headers
#include<string>
//g++ -Wall -Werror -g --std=c++20 complex_1.cpp
//g++ - compiler; -Wall - show all the errors;
//-Werror - treat warnings as errors
int main()//entry point
{
    std::string input;//variable with type string
    std::cin >> input;//standard input
    int n = input.size();
    int sum = 0;
    for( int i = 0; i < n; i++ )//n
        for( int j = 0; j < 2 * i; j++ )
        //01 0123 012345 0123456 012..2n-2
            sum++;//0 + 2 + 4 + 6 + ... 2(n-1)
    std::cout << sum << std::endl;
    return 0;//code of return
}
```

```cpp
#include<iostream>
#include<string>
#include<cmath>
int main()
{
    int n, root;
    std::cin >> n;
    root = std::sqrt( n );
    for( int i=2;i<=root;i++ ){
        if(n%i == 0){
            std::cout << i << std::endl;
            return 0;
        }
    }
    std::cout << n << " is prime" << std::endl;
    return 0;
}
```

Классы сложности

Алгоритм со сложностью f(n) называется:

- Если f $= \Theta(\log^k n)$: логарифмическим при k=1, полилогарифмическим при k>1.
- Если f $= \Theta(n)$: линейным.
- Если f $= \Theta(n \log^k n)$: linearithmic при k=1, квазилинейным при k>1.
- Если f $= \Theta(n^k)$: полиномиальным, при k=2 квадратическим.
- Если f $= \Theta(2^{n^k})$: экспоненциальным.