

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/UCvmBEbr9NZt7UEh9d0l7_A](https://www.youtube.com/channel/UCvmBEbr9NZt7UEh9d0l7_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

необязательные

1 Конечность.

Алгоритм

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

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

необязательные

- 1 Конечность.
- 2 Дискретность.

Алгоритм

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

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

необязательные

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

Алгоритм

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

Алгоритмы строятся для решения **вычислительных задач**.

Свойства алгоритма:

обязательные

необязательные

1 Конечность.

4 Правильность.

2 Дискретность.

3 Определенность.

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

```
1 #include<iostream> //headers
2 #include<string>
3 //g++ -Wall -Werror -g --std=c++20 complex_1.cpp
4 //g++ - compiler; -Wall - show all the errors;
5 //-Werror - treat warnings as errors
6 int main()//entry point
7 {
8     std::string input;//variable with type string
9     std::cin >> input;//standard input
10    int n = input.size();
11    int sum = 0;
12    for( int i = 0; i < n; i++ )//n
13        for( int j = 0; j < 2 * i; j++ )
14            //01 0123 012345 0123456 012...2i-1
15            sum++;//0 + 2 + 4 + 6 + ... 2(n-1)
16    std::cout << sum << std::endl;
17    return 0;//code of return
18 }
```


Переменные

0100

Переменные

a 4 [-7÷8]

| |
|------|
| 0100 |
|------|

Переменные

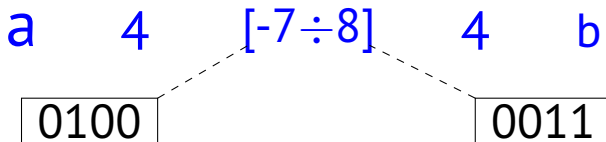
a 4 [-7÷8]

| |
|------|
| 0100 |
|------|

Мы знаем тип a?

Мы знаем диапазон значений!

Переменные

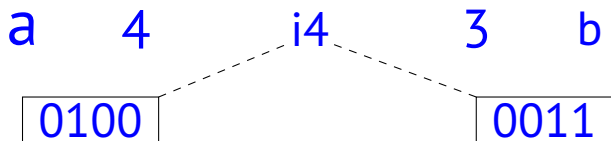


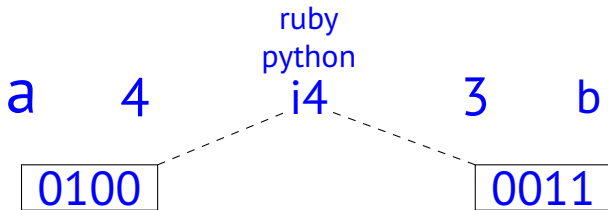
Мы знаем тип a?

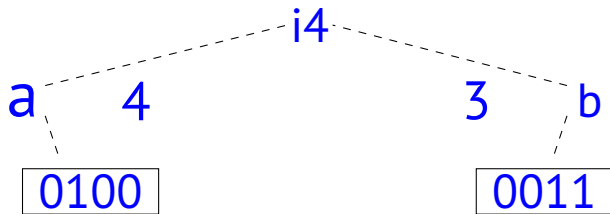
Мы знаем диапазон значений!

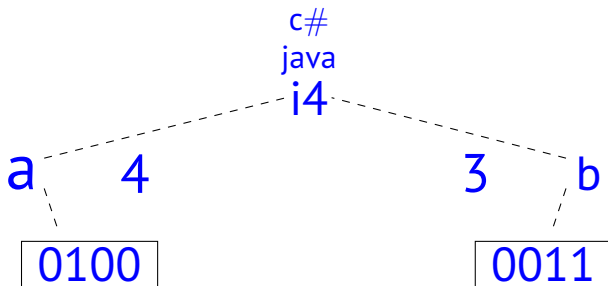
Мы не знаем операции!

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

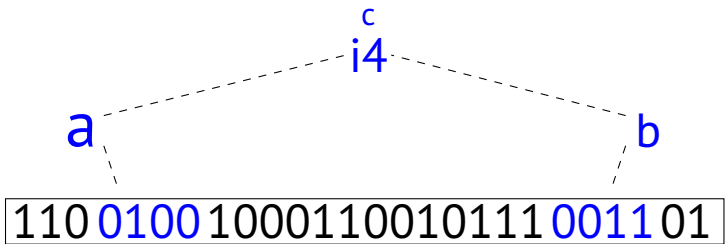


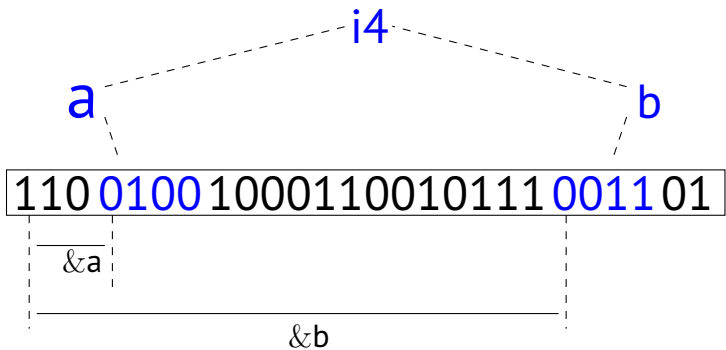


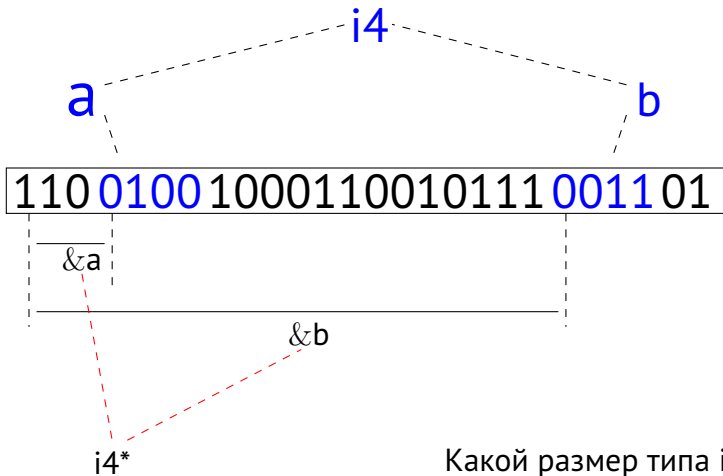




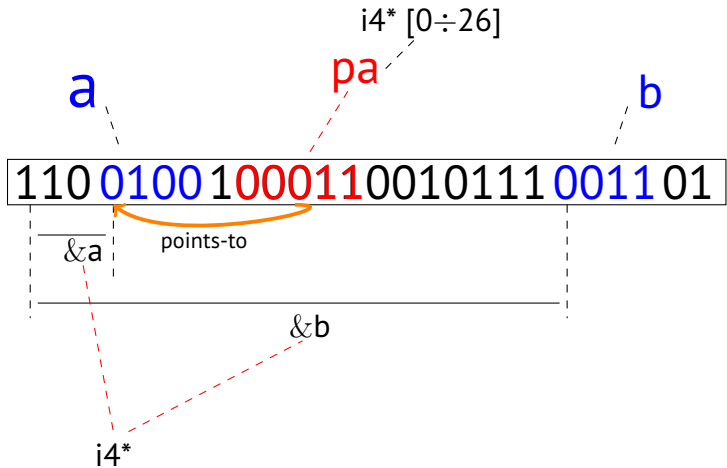
Имя навсегда связано с типом.





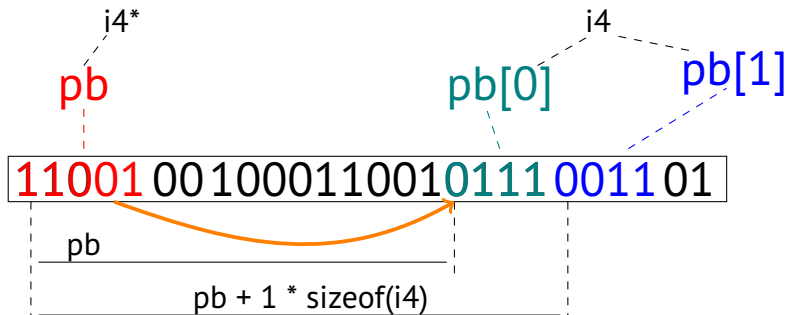


Какой размер типа $i4^*$?
Какое значение CHAR_BIT ?



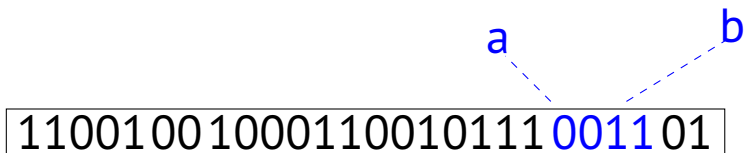
Указатели

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



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!

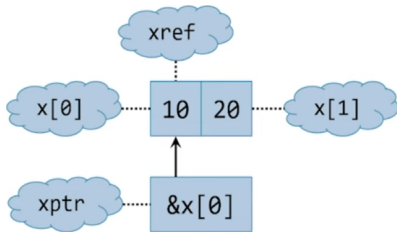
Ссылки

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
```



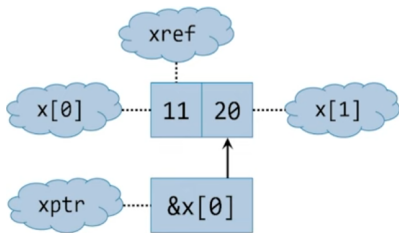
Ссылки

Базовый синтаксис lvalue ссылок это одинарный амперсанд

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

Не путать со взятием адреса!

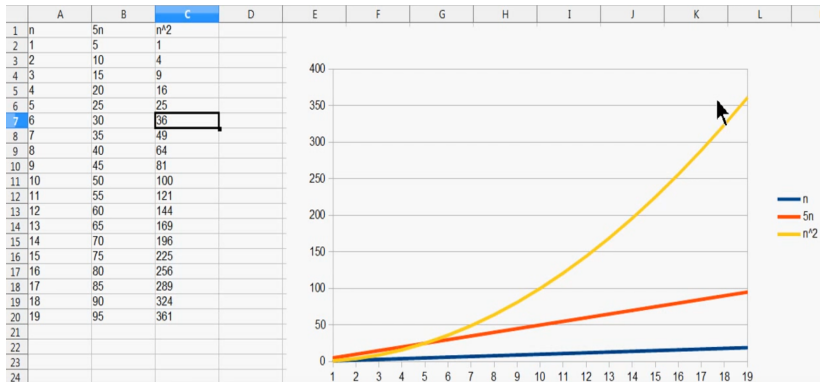
```
1 int x[2] = {10, 20};  
2 int &xref = x[0];  
3 int *xref = &x[0];  
4 xref += 1;  
5 xptr += 1;  
6 assert(xref == 11);  
7 assert(xptr == 20);  
8
```



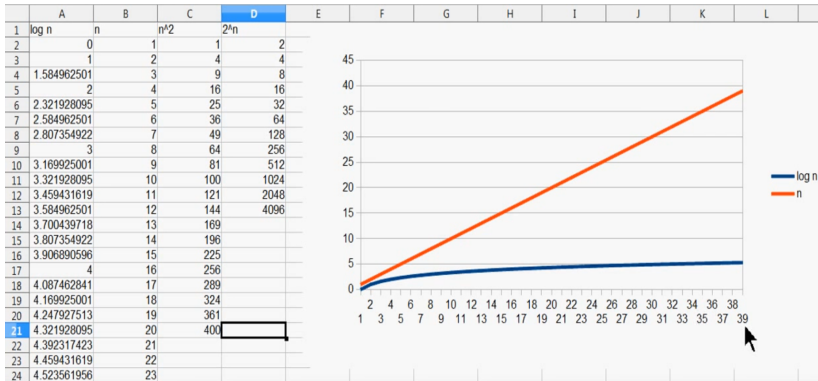
Оценка сложности алгоритма

```
1 #include<iostream> //headers
2 #include<string>
3 //g++ -Wall -Werror -g --std=c++20 complex_1.cpp
4 //g++ - compiler; -Wall - show all the errors;
5 //-Werror - treat warnings as errors
6 int main()//entry point
7 {
8     std::string input;//variable with type string
9     std::cin >> input;//standard input
10    int n = input.size();
11    int sum = 0;
12    for( int i = 0; i < n; i++ )//n
13        for( int j = 0; j < 2 * i; j++ )
14            //01 0123 012345 0123456 012...2i-1
15            sum++;//0 + 2 + 4 + 6 + ... 2(n-1)
16    std::cout << sum << std::endl;
17    return 0;//code of return
18 }
```

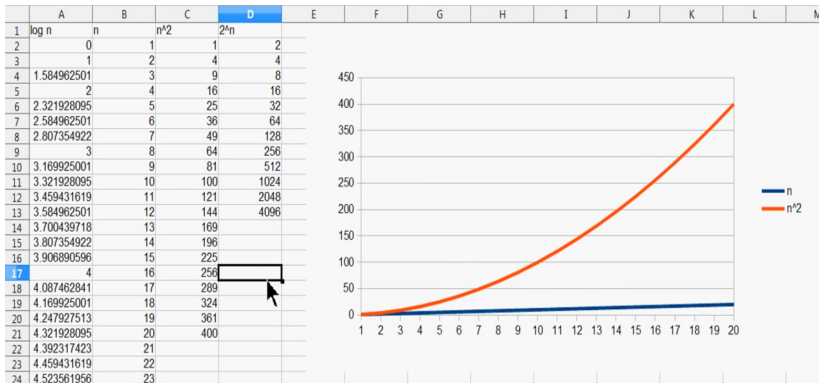
Масштаб роста функций



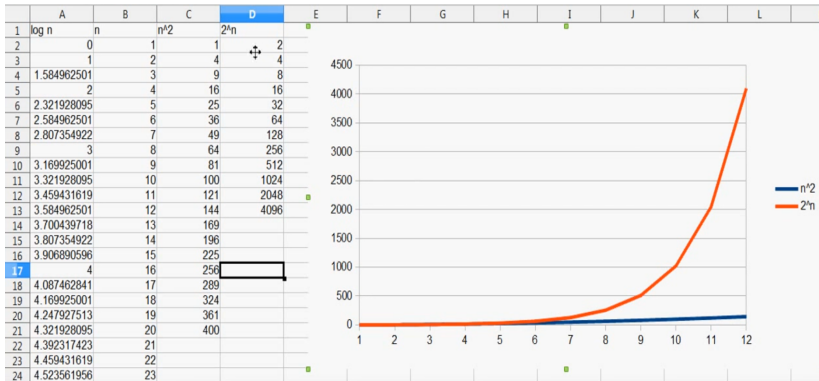
Масштаб роста функций



Масштаб роста функций



Масштаб роста функций



O-символика

$$f(n) = o(g(n)) \quad \forall k > 0 \quad \exists n_0 \quad \forall n > n_0 \quad f(n) < k \cdot g(n) \quad \Leftrightarrow \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \quad f(n) \prec g(n)$$

$$f(n) = O(g(n)) \quad \exists k > 0 \quad \exists n_0 \quad \forall n > n_0 \quad f(n) < k \cdot g(n) \quad \Leftrightarrow \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty \quad f(n) \preceq g(n)$$

$$f(n) = \Theta(g(n)) \quad \begin{array}{l} \exists k_1 k_2 > 0 \quad \exists n_0 \\ \forall n > n_0 \\ k_1 \cdot g(n) < f(n) \\ f(n) < k_2 \cdot g(n) \end{array} \quad \Leftrightarrow \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \quad f(n) \approx g(n)$$

```

1 #include<iostream> //headers
2 #include<string>
3 //g++ -Wall -Werror -g --std=c++20 complex_1.cpp
4 //g++ - compiler; -Wall - show all the errors;
5 //-Werror - treat warnings as errors
6 int main()//entry point
7 {
8     std::string input;//variable with type string
9     std::cin >> input;//standard input
10    int n = input.size();
11    int sum = 0;
12    for( int i = 0; i < n; i++ )//n
13        for( int j = 0; j < 2 * i; j++ )
14            //01 0123 012345 0123456 012...2i-1
15            sum++;//0 + 2 + 4 + 6 + ... 2(n-1)
16    std::cout << sum << std::endl;
17    return 0;//code of return
18 }

```

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.

```
1 #include<iostream>
2 #include<string>
3 #include<cmath>
4 int main()
5 {
6     int n, root;
7     std::cin >> n;
8     root = std::sqrt( n );
9     for( int i=2;i<=root;i++ ){
10         if(n%i == 0){
11             std::cout << i << std::endl;
12             return 0;
13         }
14     }
15     std::cout << n << " is prime" << std::endl;
16     return 0;
17 }
```

```
1 #include<iostream>
2 #include<string>
3 #include<cmath>
4 int main()
5 {
6     int n, root;
7     std::cin >> n;
8     root = std::sqrt( n );
9     for( int i=2;i<=root;i++ ){
10         if(n%i == 0){//2 3 4 5 ... sqrt(n)
11             std::cout << i << std::endl;
12             return 0;
13         }
14     }
15     std::cout << n << " is prime" << std::endl;
16     return 0;
17 }
```

Домашнее задание

- Написать программу реализующую алгоритм Эратосфена.
- Оценить сложность - написать в программе комментариями мысли и результат.
- Сделать pull request в папку lecture_01/homework/ с файлом решения.