Продолжаем (и заканчиваем) разбирать фишки C++, а также первые алгоритмы

Мячин Данил
ВШЭ ФКН ПМИ

Что у вас было в дз

5 дней Превышено ограничение времени на тесте 37 → 94657217

4 дня Превышено ограничение времени на тесте 8 → 94586469

32:54:12 Превышено ограничение времени на тесте 8 → 94341442
32:56:45 Неправильный ответ на тесте 1 → 94341907
5 дней Неправильный ответ на тесте 1 → 94656690
5 дней Превышено ограничение времени на тесте 8 → 94656792
5 дней Превышено ограничение времени на тесте 9 → 94658249
5 дней **Полное решение** → 94658613

А как понять, пройдет ли моя программа по времени?

А как понять, пройдет ли моя программа по времени?

C. Усложнённое распознавание степени двойки

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

А как понять, пройдет ли моя программа по времени?

C. Усложнённое распознавание степени двойки

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

C++ выполняют порядка 10^9 операций в секунду

Поговорим о сложности программ

B. Это правда степень двойки?

ограничение по времени на тест: 0.4 секунд
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Задача звучит очень просто. На вход программе подаётся число. Надо определить является ли это число степенью двойки. Если да - вывести "Yes", иначе "No" (всё без кавычек).

Входные данные
На единственной строке находится число n $(1 \leq n \leq 10^9)$.

Выходные данные
Нужно вывести на одной строке слово "Yes", если число является степенью двойки и "No" иначе

Примеры

входные данные	Скопировать
419	
выходные данные	Скопировать
No	

входные данные	Скопировать
16	
выходные данные	Скопировать
Yes	

```cpp
#include <iostream>

using namespace std;

int main() {

    int n; cin >> n;

    while (n > 1) {
        if ((n & 1) == 1) {
            cout << "No";
            return 0;
        }
        n = (n >> 1); // n /= 2
    }

    cout << "Yes";

    return 0;
}
```

- n каждый раз делится на 2

- n каждый раз делится на 2
- Тогда максимально нам понадобится log(2, 10^9) операций для цикл

- n каждый раз делится на 2
- Тогда максимально нам понадобится log(2, n) операций для цикла
- Как это можно интерпретировать? Можно посчитать, сколько раз нам нужно 1 умножить на 2, чтобы получить число не меньшее n

- n каждый раз делится на 2
- Тогда максимально нам понадобится log(2, n) операций для цикла
- Как это можно интерпретировать? Можно посчитать, сколько раз нам нужно 1 умножить на 2, чтобы получить число не меньшее n
- Чтобы посчитать кол-во операций – нужно брать наихудшие случаи (в нашем случае n = 10^9)

- n каждый раз делится на 2
- Тогда максимально нам понадобится log(2, n) операций для цикла
- Как это можно интерпретировать? Можно посчитать, сколько раз нам нужно 1 умножить на 2, чтобы получить число не меньшее n
- Чтобы посчитать кол-во операций – нужно брать наихудшие случаи (в нашем случае n = 10^9)
- Тогда программисты говорят, что асимптотика программы
- O(log(2, n))

Что такое асимптотика?

- Асимптотика – кол-во действий без констант: мы не считаем действием ровно один вывод числа – это крайне малая единица действия, которая будет ничтожно мала по сравнению со всей программой

Что такое асимптотика?

- Асимптотика – кол-во действий без констант: мы не считаем действием ровно один вывод числа – это крайне малая единица действия, которая будет ничтожно мала по сравнению со всей программой

```cpp
#include <iostream>

using namespace std;

int main() {

    int n; cin >> n;

    while (n > 1) {
        if ((n & 1) == 1) {
            cout << "No";
            return 0;
        }
        n = (n >> 1); // n /= 2
    }

    cout << "Yes";

    return 0;
}
```

Более простой пример

E. Кадиллак

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Однажды музыкальный исполнитель Моргенштерн записывал клип "Кадиллак". Этот клип очень понравился Пете. Он настолько ему понравился, что слушал его в колонках во дворе. Но все бы ничего, если бы Петя не заметил одну особенность клипа: откуда у машин такие номера?. При чем машины были из разных регионов России. У Петя появился вопрос: а в каком регионе России больше всего любят Моргенштерна? Для этого он решил узнать регион, из которого приехало больше всего машин. Может это не будет соответствовать действительности, но для Пети это правда важно!

Входные данные

На первой строке ввода находится два числа n и k $(1 \leq n,\ k \leq 10^6)$ - кол-во машин в клипе и кол-во регионов в России. На второй строке ввода находится n чисел - регионы каждой из машин. Если reg_i регион i-й машины, то $1 \leq reg_i \leq k$.

Выходные данные

Выведите ответ на вопрос Пети: из какого региона машин больше всего? Вы должны вывести одно число. Если ответов несколько - вывести наименьший

Примеры

входные данные	Скопировать
10 5 1 4 2 3 5 1 1 5 1 3	
выходные данные	Скопировать
1	

входные данные	Скопировать
10 7 3 6 5 4 3 2 4 3 1 3	
выходные данные	Скопировать
3	

Примечание

Такая задача (только несколько в иной формулировке) может встретиться вам на ЕГЭ по информатике

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main() {
        int n, m;

        cin >> n >> m;

        vector<int> reg(m, 0);

        int i;
        while (n--) {
                cin >> i;
                reg[i-1]++;
        }

        int largestElemIdx = 0;
        for (i = 0; i < reg.size(); i++) {
                if (reg[i] > reg[largestElemIdx]) largestElemIdx = i;
        }

        cout << largestElemIdx+1 << endl;

        return 0;
}
```

А что делать, если у меня в цикле используются функции и прочее?

- Мы можем цикл рассматривать как отдельную функцию, которая нам прибавляет время. То есть O(кол-во итераций вектора * (действия в векторе)).

А что делать, если у меня в цикле используются функции и прочее?

- Мы можем цикл рассматривать как отдельную функцию, которая нам прибавляет время. То есть O(кол-во итераций вектора * (действия в векторе)).
- В предыдущем слайде у нас было O(n*(1)) = O(n)

А что делать, если у меня в цикле используются функции и прочее?

- Мы можем цикл рассматривать как отдельную функцию, которая нам прибавляет время. То есть O(кол-во итераций вектора * (действия в векторе)).
- В предыдущем слайде у нас было O(n*(1)) = O(n)
- А давайте теперь вспомним про задачу на нахождение суммы в подматрице

ЗАДАЧИ ОТОСЛАТЬ МОИ ПОСЫЛКИ СТАТУС ПОЛОЖЕНИЕ АДМ. РЕД. ЗАПУСК

D. Сумма в подматрице

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Вам просто подаётся на вход матрица, а потом поступает некоторое кол-во запросов такого вида: $i_1\ j_1\ i_2\ j_2$. На этот запрос нужно отвечать суммой чисел, которые стоят в подматрице на пересечении строк с i_1 по i_2 и столбцов с j_1 по j_2. Если вам нужна формальная формула, то это так:

$$\sum_{i=i_1}^{i_2}\sum_{j=j_1}^{j_2} a[i][j]$$

где $a[i][j]$ - элемент матрицы.

Входные данные

В первой строке находятся два числа: n и m $(1 \le n, m \le 10^3)$ - кол-во строк и столбцов в матрице. Далее на n строках находятся по m чисел - значения чисел в матрице, не превышающие 10^9 по абсолютной величине. На n+2-й строке находится число q - кол-во запросов. Далее на q строках находятся по 4 числа - $i1\ j1\ i2\ j2$ $(0 \le i_1 \le i_2 \le n-1,\ 0 \le j_1 \le j_2 \le m-1)$.

Выходные данные

В качестве ответа вам нужно вывести q строк - ответы на запросы в порядке их поступления.

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
	int n, m, q;

	cin >> n >> m;

	vector<vector<unsigned int>> mx(n, vector<unsigned int> (m, 0));

	for (int i = 0; i < n; i++) {
		for (int j = 0; j < m; j++) {
			cin >> mx[i][j];
		}
	}

	cin >> q;

	int i1, j1, i2, j2;
	unsigned long long int sum;
	while (q--) {
		cin >> i1 >> j1 >> i2 >> j2;
		sum = 0;

		for (int i = i1; i <= i2; i++) {
			for (int j = j1; j <= j2; j++) {
				sum += mx[i][j];
			}
		}

		cout << sum << endl;
	}

	return 0;
}
```

Ещё одна классная вещь, которая нам в будущем пригодится - struct

- Структуры нужны для того, чтобы хранить какие-то значения (возможно даже разных типов) в «одной» переменной. Это бывает очень удобно

Как объявить структуру?

Как объявить структуру?

```cpp
#include <iostream>
using namespace std;

struct hero {
    int strength;
    int intell;
    int agility;
    double health;
    double mana;
    double regen_healt;
    double regen_mana;
    double attack;
};

int main() {
    int n; cin >> n;
    vector<hero> heroes(n);
    for (int i = 0; i < n; ++i) {
        cin >> heroes[i]. strength >> heroes[i].intell >> ....
    }
    return 0;
}
```

Первый алгоритм

Бинарный(двоичный) поиск

Бинарный(двоичный) поиск

- В классическом понимании – поиск элементов в отсортированном массиве быстрее, чем за O(n)

Бинарный(двоичный) поиск

- В классическом понимании – поиск элементов в отсортированном массиве быстрее, чем за O(n)
- Работает за O(log(2, n)), где n – количество чисел в массиве

Бинарный(двоичный) поиск

- В классическом понимании – поиск элементов в отсортированном массиве быстрее, чем за O(n)
- Работает за O(log(2, n)), где n – количество чисел в массиве
- Если массив не отсортирован – вы проиграли

Как выглядит сам алгоритм

Как выглядит сам алгоритм

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

Источник: https://clck.ru/FUfAp

Давайте посмотрим визуализацию поиска

Давайте посмотрим визуализацию поиска

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

Давайте посмотрим визуализацию поиска

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

Давайте посмотрим визуализацию поиска

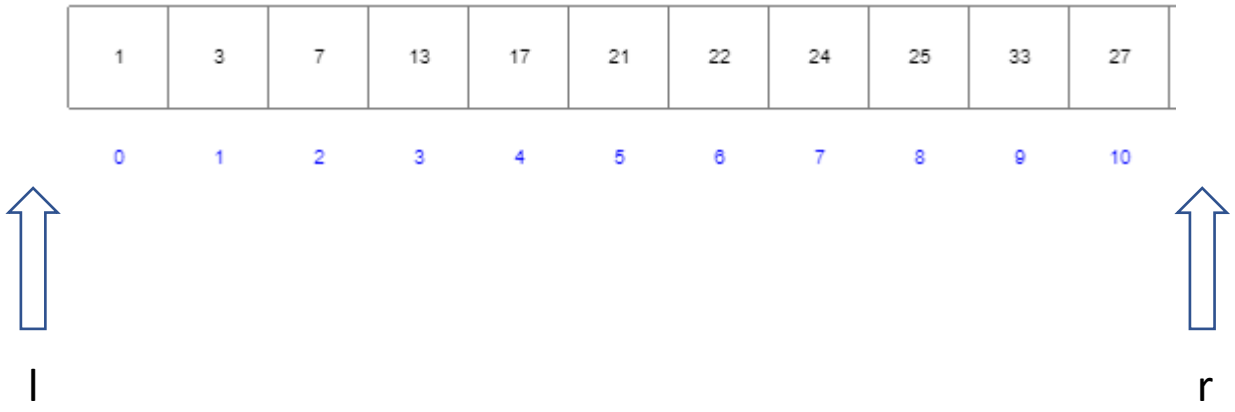
```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

Ищем число 13

Давайте посмотрим визуализацию поиска

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

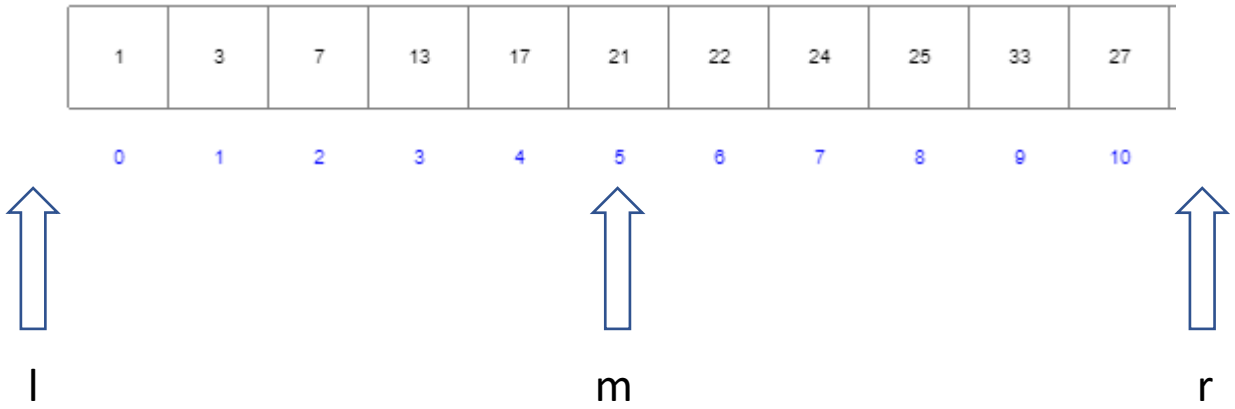
Ищем число 13



Давайте посмотрим визуализацию поиска

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

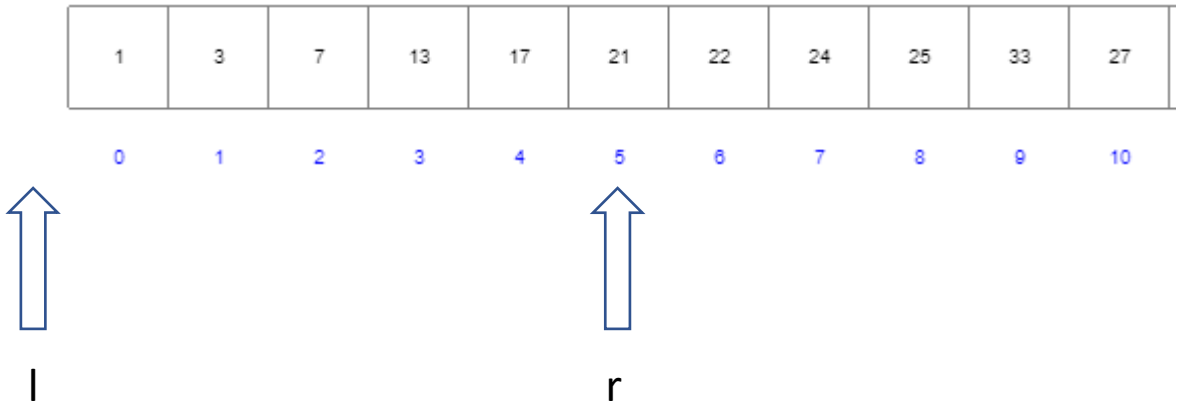
Ищем число 13



Давайте посмотрим визуализацию поиска

```
int binSearch(int[] a, int key):    // Запускаем бинарный поиск
    int l = -1                       // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                 // Запускаем цикл
        m = (l + r) / 2             // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                   // Сужение границ
    return r
```

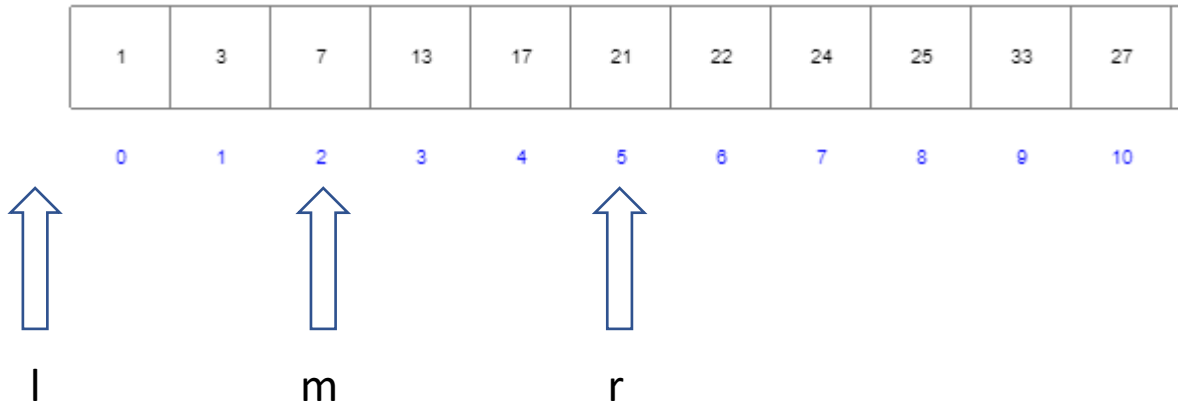
Ищем число 13



Давайте посмотрим визуализацию поиска

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

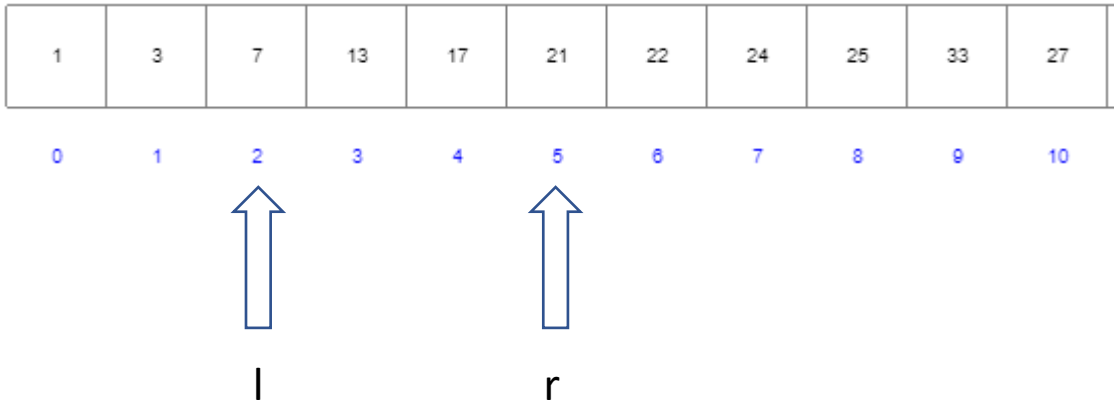
Ищем число 13



Давайте посмотрим визуализацию поиска

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

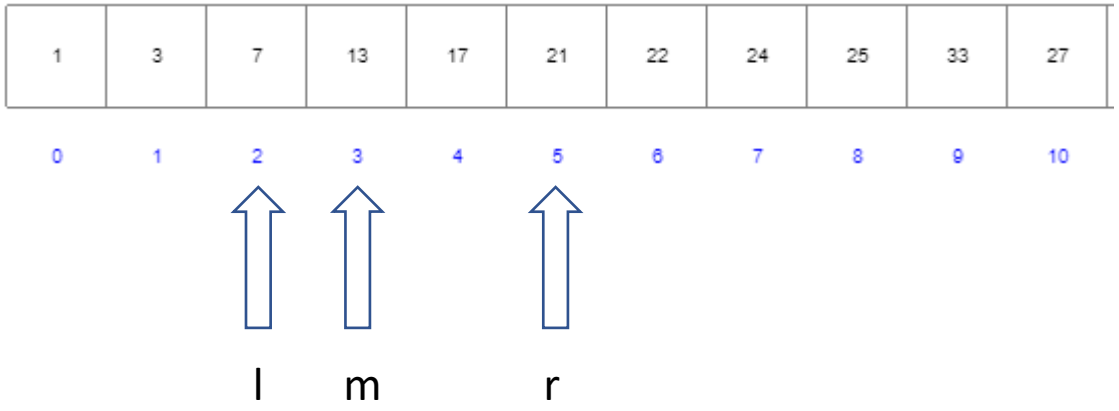
Ищем число 13



Давайте посмотрим визуализацию поиска

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

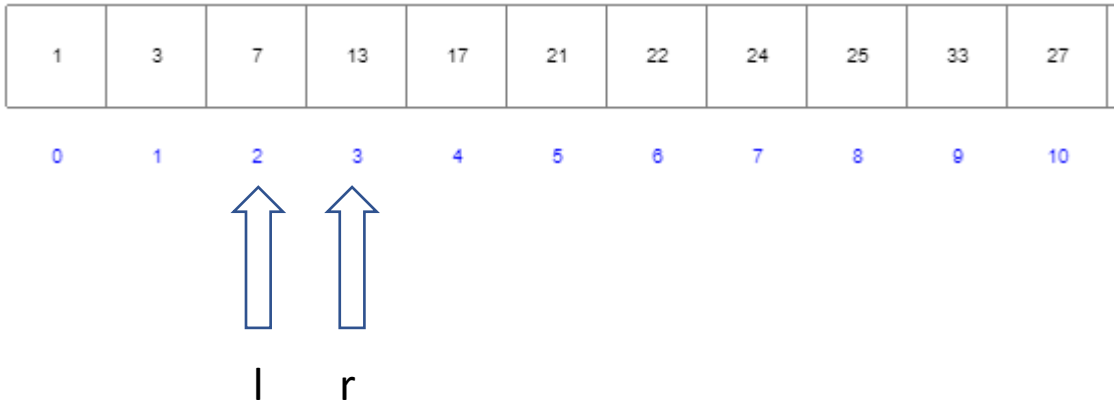
Ищем число 13



Давайте посмотрим визуализацию поиска

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

Ищем число 13



Давайте теперь поищем число, которого нет

Давайте теперь поищем число, которого нет

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

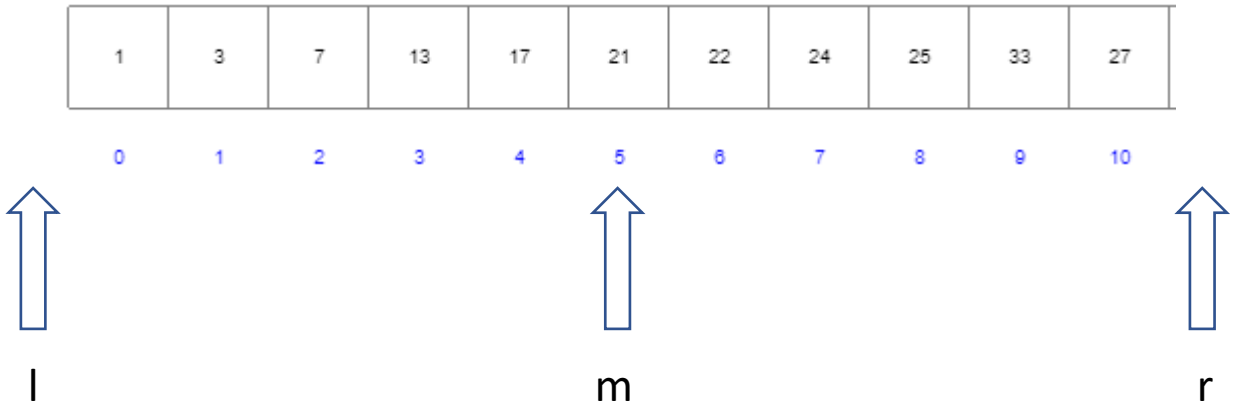
Ищем число 23

1	3	7	13	17	21	22	24	25	33	27
0	1	2	3	4	5	6	7	8	9	10

Давайте теперь поищем число, которого нет

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

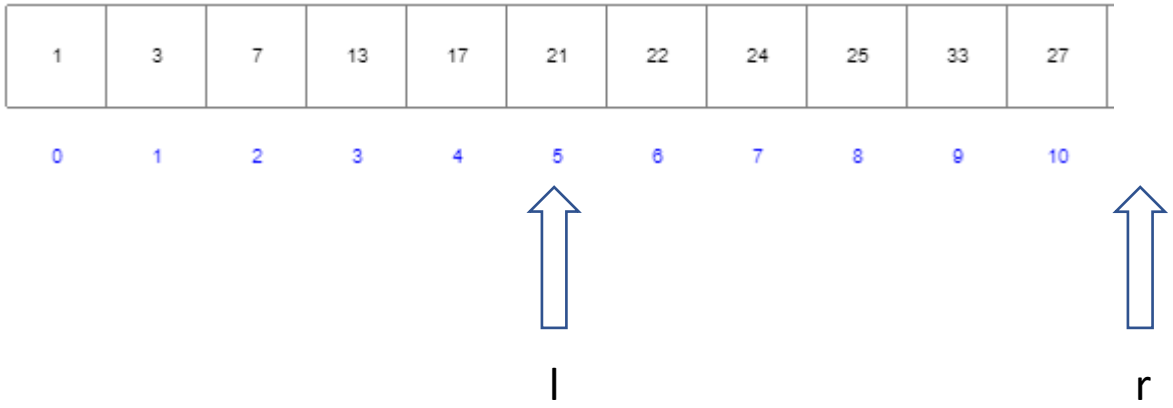
Ищем число 23



Давайте теперь поищем число, которого нет

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

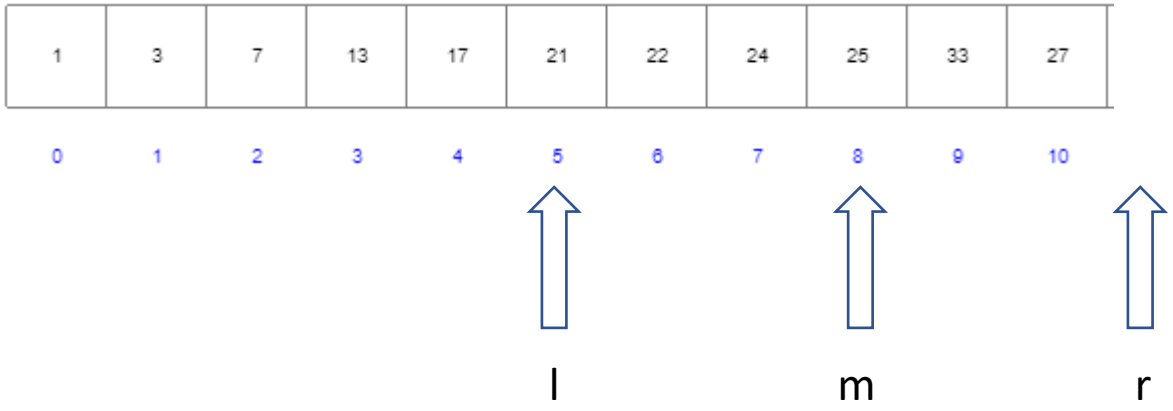
Ищем число 23



Давайте теперь поищем число, которого нет

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

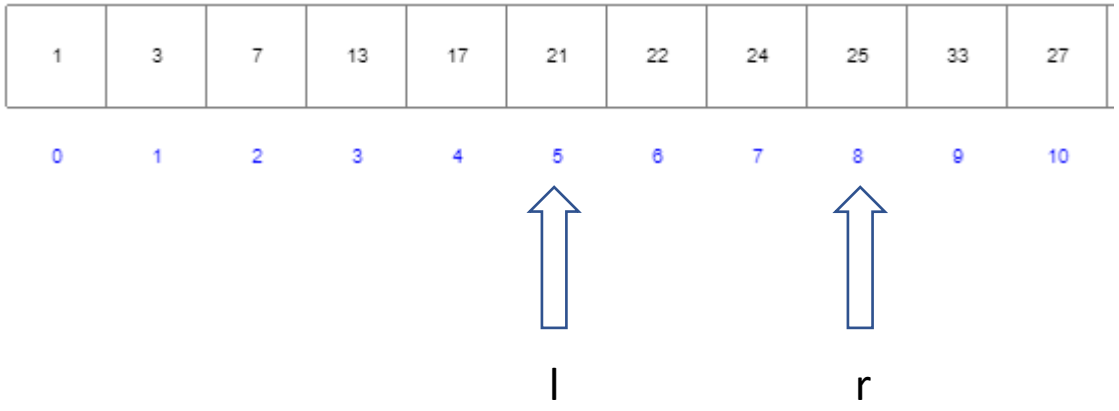
Ищем число 23



Давайте теперь поищем число, которого нет

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

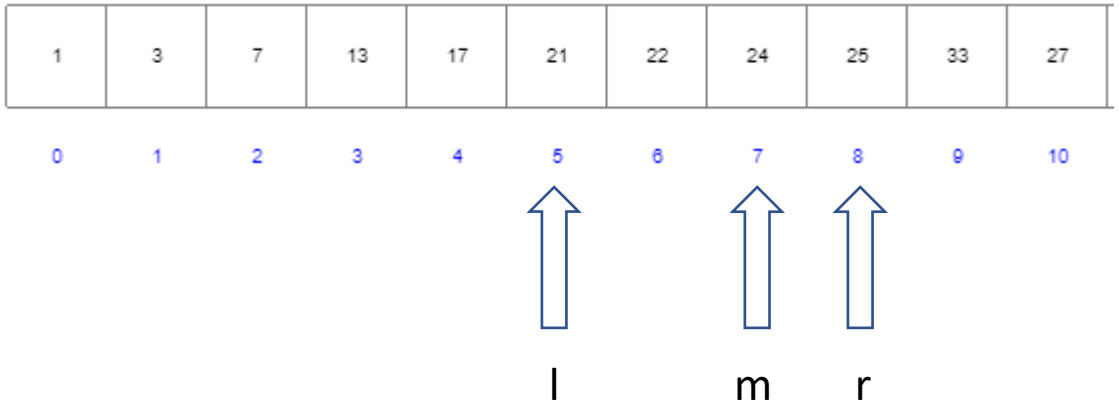
Ищем число 23



Давайте теперь поищем число, которого нет

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

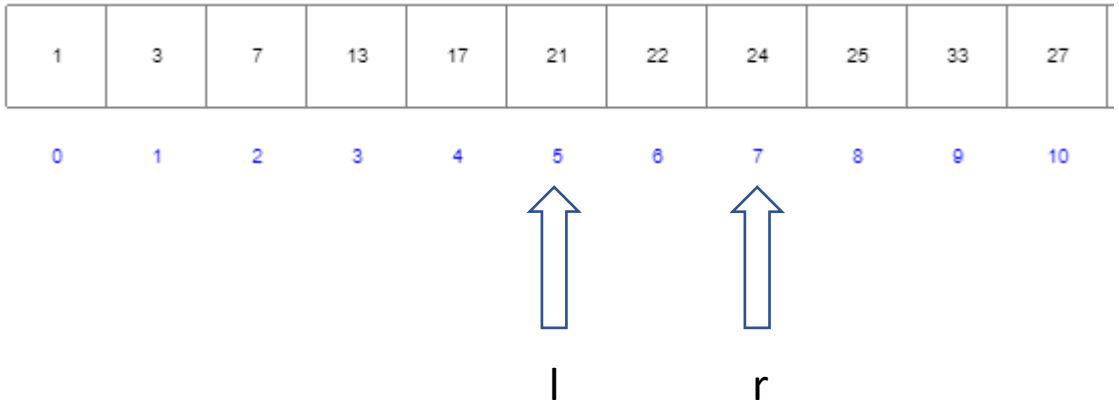
Ищем число 23



Давайте теперь поищем число, которого нет

```
int binSearch(int[] a, int key):    // Запускаем бинарный поиск
    int l = -1                       // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                 // Запускаем цикл
        m = (l + r) / 2             // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                   // Сужение границ
    return r
```

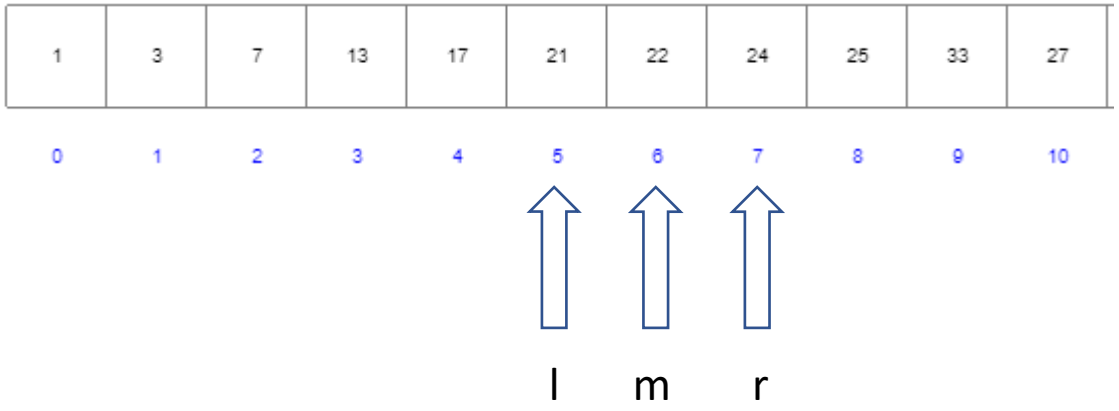
Ищем число 23



Давайте теперь поищем число, которого нет

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

Ищем число 23



Давайте теперь поищем число, которого нет

```
int binSearch(int[] a, int key):   // Запускаем бинарный поиск
    int l = -1                      // l, r – левая и правая границы
    int r = len(a)
    while l < r - 1                // Запускаем цикл
        m = (l + r) / 2            // m – середина области поиска
        if a[m] < key
            l = m
        else
            r = m                  // Сужение границ
    return r
```

Ищем число 23