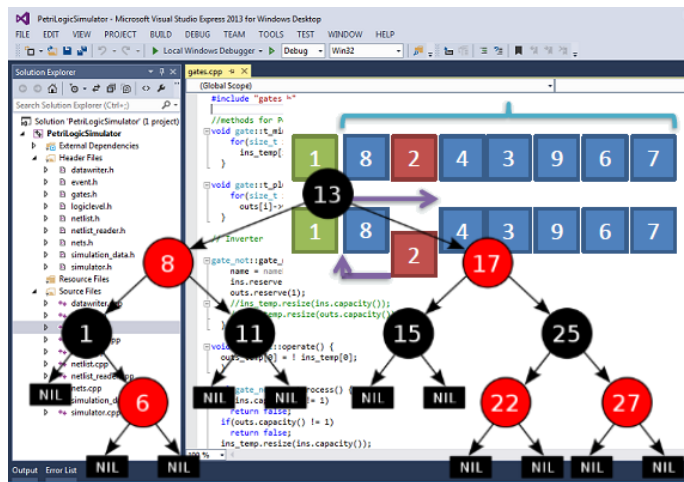




Теория алгоритмов

Лекция 1

Общие сведения об алгоритмах

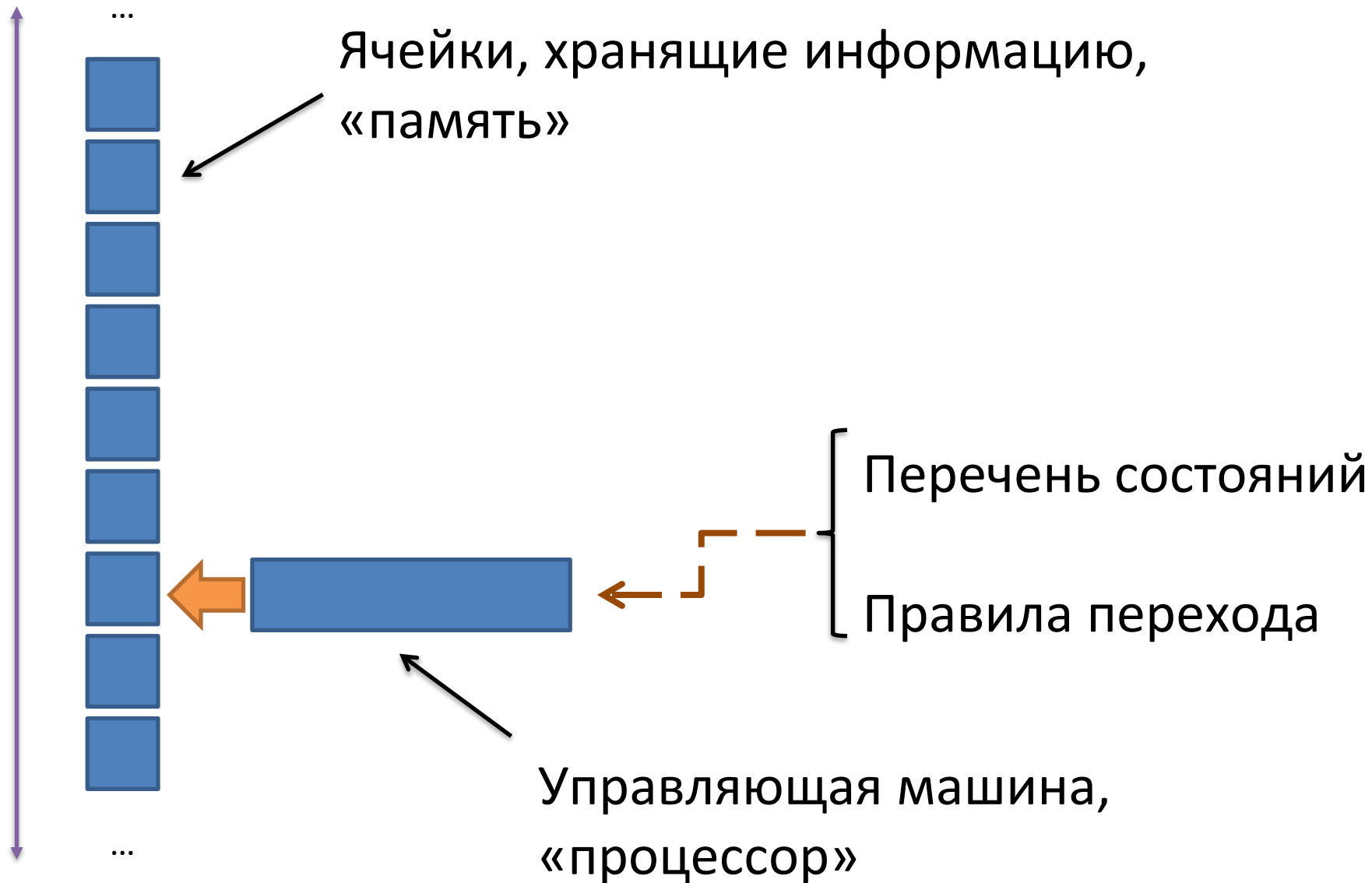


Алгоритм: определение



Алгоритм - это организованная последовательность действий, понятных для некоторого исполнителя, ведущая за конечное число шагов к решению поставленной задачи.

Машина Тьюринга



Алан Мэтисон Тьюринг
(1912 - 1954)

Тезисы Чёрча-Тьюринга

1. Любая функция, которая может быть вычислена физическим устройством, может быть вычислена машиной Тьюринга



Алан Мэтисон Тьюринг
(1912 - 1954)



Алонзо Чёрч
(1903 - 1995)

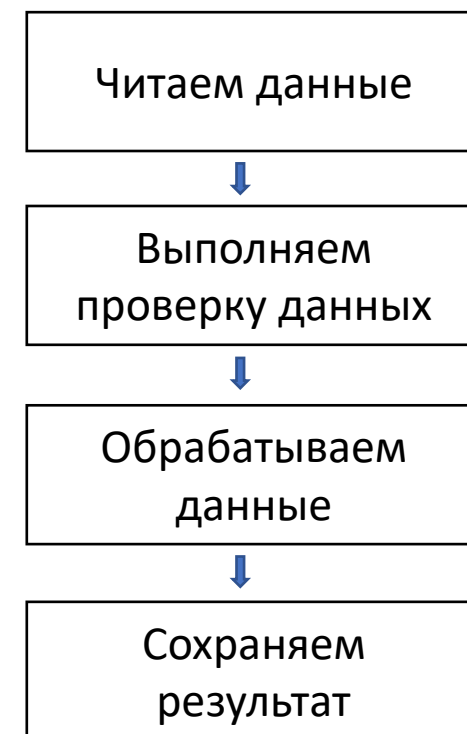
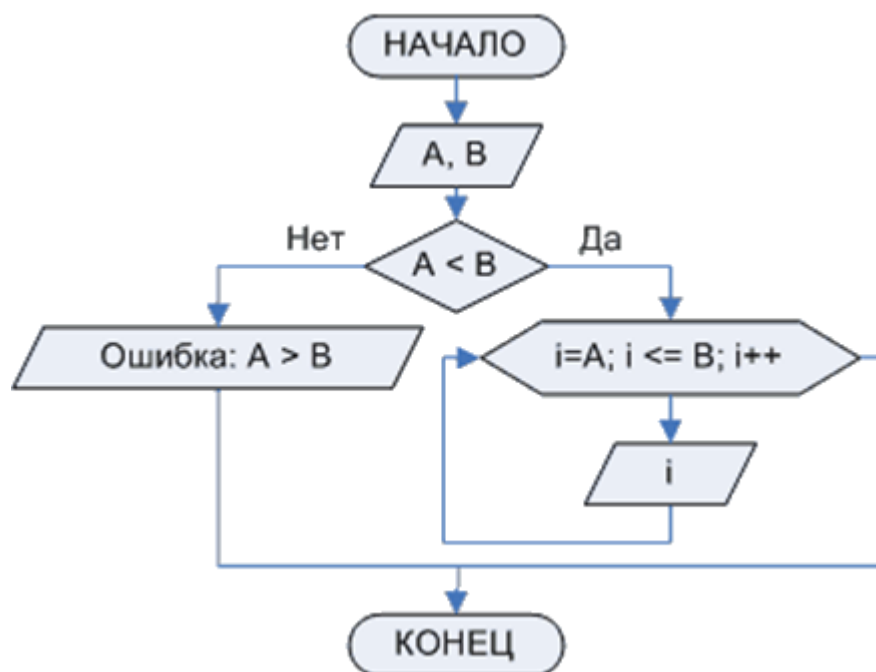
2. Любой конечный физический процесс, не использующий аппарат, связанный с непрерывностью и бесконечностью, может быть вычислен физическим устройством

Свойства алгоритмов

- **Дискретность**
- **Корректность**
- **Детерминированность**
- **Массовость** - с помощью алгоритма можно решать не одну конкретную задачу, а любую задачу из некоторого класса однотипных задач при всех допустимых значениях исходных данных
- **Конечность** - последовательность элементарных действий алгоритма не может быть бесконечной, неограниченной, хотя может быть очень большой
- **Результативность** - выполнение алгоритма обязательно должно привести к решению поставленной задачи, либо к сообщению о том, что при заданных исходных величинах задачу решить невозможно

Свойства алгоритмов: дискретность

Дискретность - означает, что алгоритм состоит из последовательности отдельных шагов - элементарных действий, выполнение которых на заданном уровне абстракции не представляет сложности и вполне ПОНЯТНО.



Свойства алгоритмов: корректность

Корректность - означает, что, если алгоритм создан для решения определенной задачи, то для всех исходных данных он должен всегда давать правильный результат и ни для каких исходных данных не будет получен неправильный результат.

Сортировка:

- для целых чисел (int)
- для чисел с плавающей точкой (float)
- для символьных (char)
- как для положительных, так и для отрицательных чисел
- для массива как в 10, так и в 1000 элементов

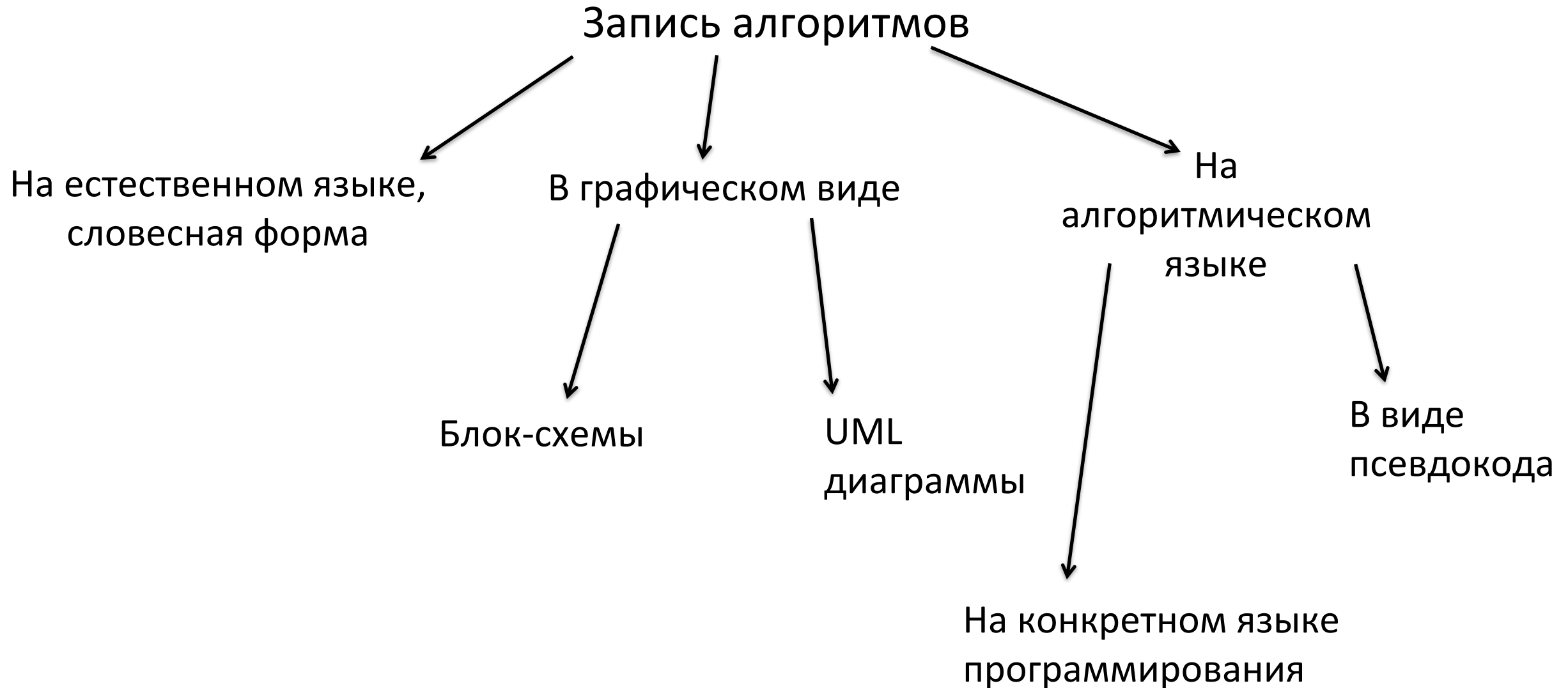
Свойства алгоритмов: детерминированность

Детерминированность (определенность, точность, однозначность).

Это свойство заключается в том, что при задании одних и тех же исходных данных несколько раз алгоритм будет выполняться абсолютно одинаково и всегда будет получен один и тот же результат.

На каждом шаге выполнения алгоритма всегда точно известно, что делать дальше, а каждое действие однозначно понятно исполнителю и не может быть истолковано неопределенно.

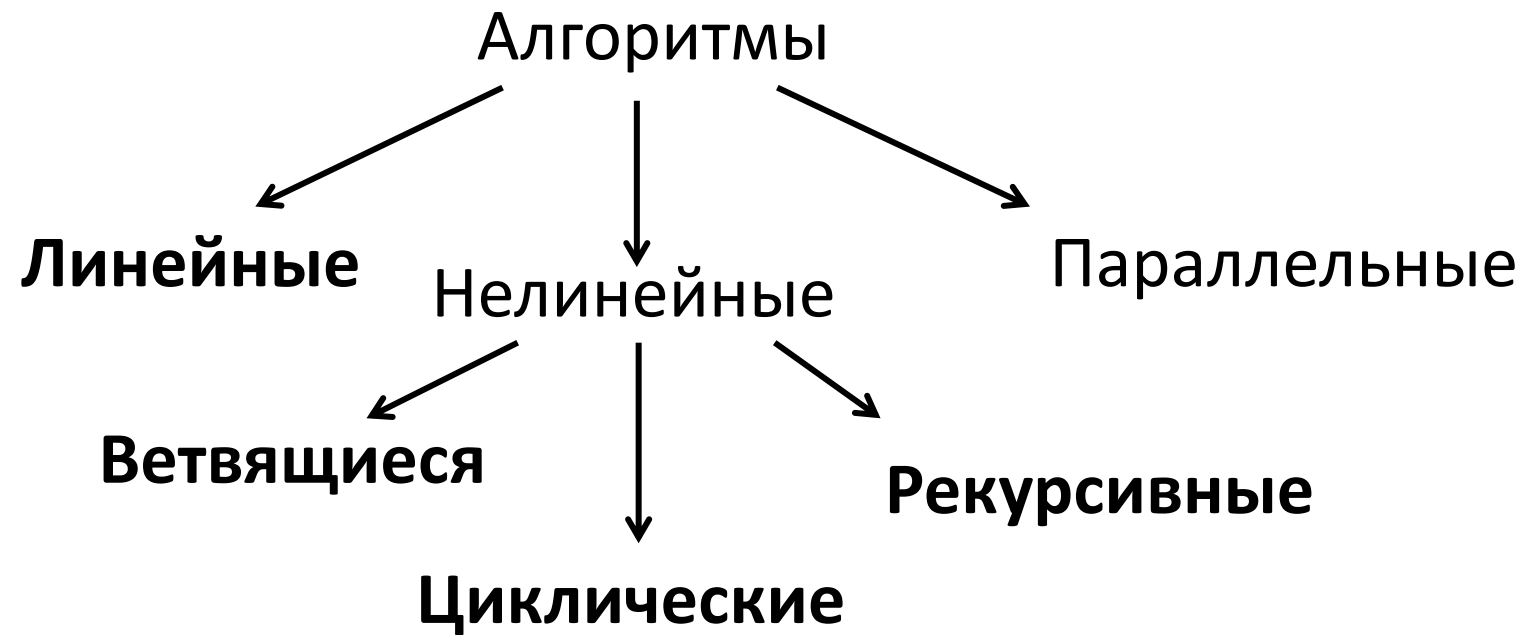
Формы записи алгоритмов



Виды алгоритмов с точки зрения их исполнения

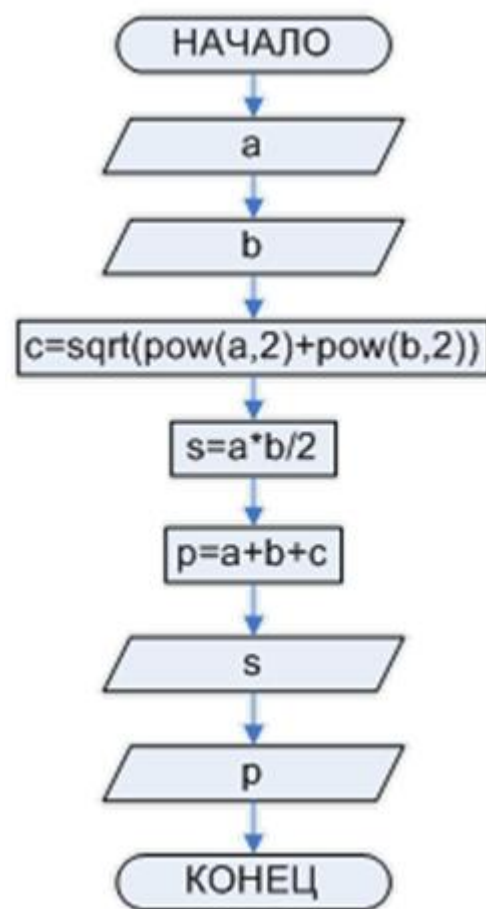
Не рассматриваем вопросы области применения алгоритмов: сортировки, поиска, обхода графа и т.д.

Только виды самих алгоритмов по принципу хода исполнения.



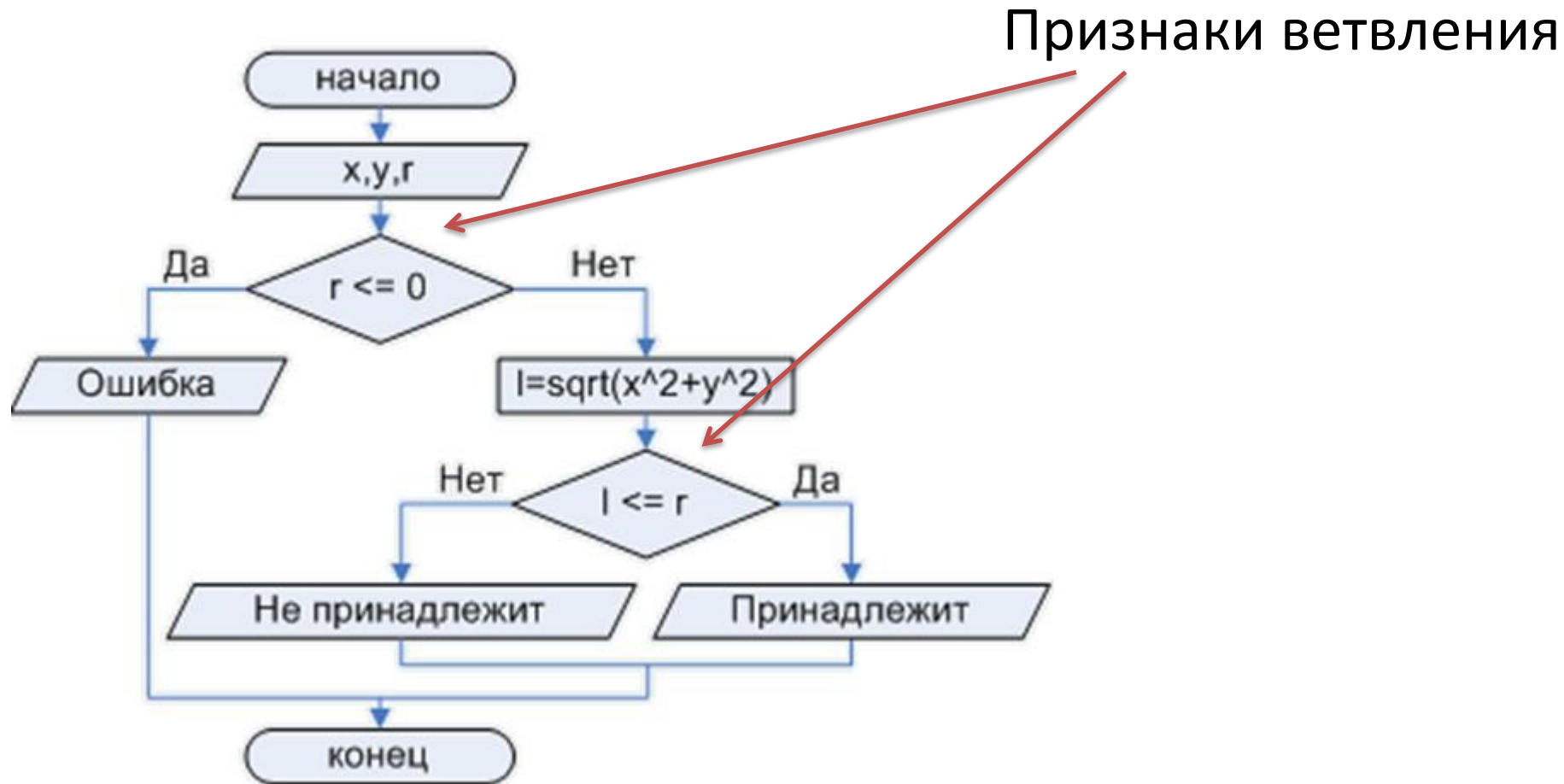
Виды алгоритмов: линейные

Линейные – на практике реально почти не применяются, но играют важную роль при разработке работы программы на высоком уровне абстракции.



Виды алгоритмов: ветвящиеся

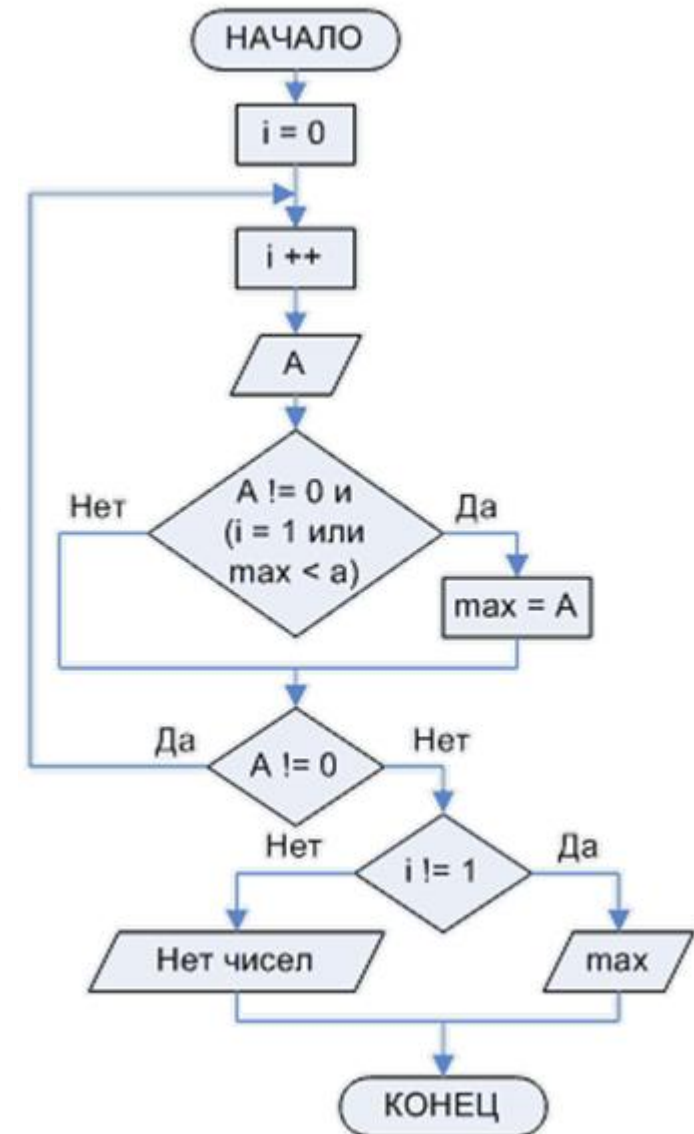
Разветвляющиеся алгоритмы – те, в которых есть проверка логических выражений.



Виды алгоритмов: циклические

Циклические алгоритмы – те, в которых используется многократное повторное выполнение одного и того же программного кода, реализованного в теле цикла.

Признак
циклическости



Виды алгоритмов: рекурсивные

Рекурсивные алгоритмы – те, в которых подпрограмма вызывается сама из себя.

```
long Fib_NoRec(long A) {  
    long a = 0, b = 1, c = 0;  
    while (A - 1) {  
        c = a + b;  
        a = b;  
        b = c;  
        --A;  
    }  
    return c;  
}
```

← Перебор

Рекурсия →

```
long Fib_Rec(long A) {  
    if (A == 0 || A == 1)  
        return A;  
  
    return (Fib_Rec(A - 1) + Fib_Rec(A - 2));  
}
```

Виды алгоритмов по принципу работы

Детерминированные (жёсткие) алгоритмы – предполагают одну и ту же последовательность действий вне зависимости от входных данных.

Вероятностные (гибкие, адаптивные) алгоритмы – предполагают различное решение в зависимости от некоторых случайным образом генерируемых данных.

Эвристические (статистически верные) алгоритмы – не имеют чёткого обоснования, но дают верный ответ в большинстве случаев.

Парадигмы разработки алгоритмов: Линейная

Линейная

Разделяй и властвуй

Динамическое программирование

Реализация «в ширину» и «в глубину»

К линейной парадигме разработки относят:

1. большинство вычислительных алгоритмов: методы решения СЛАУ;
2. «медленные» алгоритмы поиска;
3. «медленные» алгоритмы сортировки данных.

Парадигмы разработки алгоритмов: «Разделяй и властвуй»

Линейная

Разделяй и властвуй

Динамическое программирование

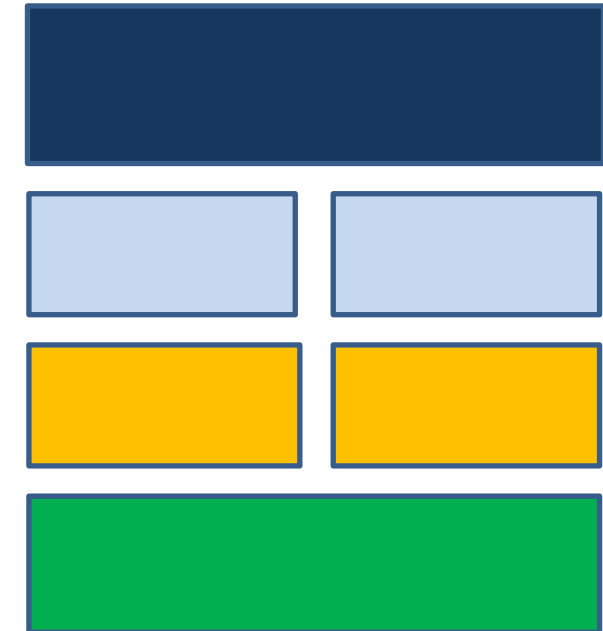
Реализация «в ширину» и «в глубину»

Жадные алгоритмы

Вариант 1



Вариант 2



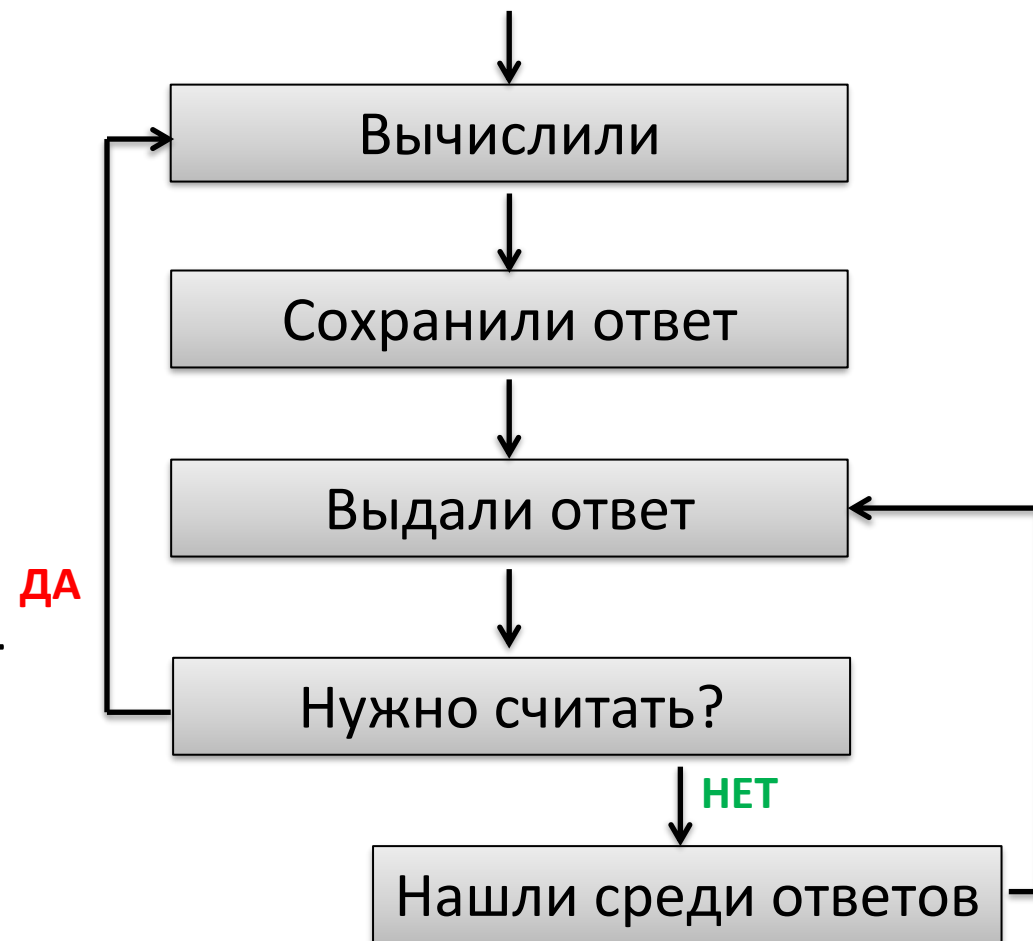
Основные этапы:

1. разделить целостные данные на части;
2. обработать выбранный/каждый блок данных;
3. при необходимости – объединить результаты обработки разных блоков;

Парадигмы разработки алгоритмов: динамическое программирование

Линейная
Разделяй и властвуй
Динамическое программирование
Реализация «в ширину» и «в глубину»
Жадные алгоритмы

Динамическая парадигма предполагает сохранение результатов предыдущих вычислений.



Парадигмы разработки алгоритмов: реализация «в ширину» и «в глубину»

Линейная
Разделяй и властвуй
Динамическое программирование
Реализация «в ширину» и «в глубину»
Жадные алгоритмы

```
long Fib_Rec(long A) {  
    if (A == 0 || A == 1)  
        return A;  
  
    return (Fib_Rec(A - 1) + Fib_Rec(A - 2));  
}
```

«в глубину»



```
long Fib_NoRec(long A) {  
    long a = 0, b = 1, c = 0;  
    while (A - 1) {  
        c = a + b;  
        a = b;  
        b = c;  
        --A;  
    }  
    return c;  
}
```

«в ширину»



Парадигмы разработки алгоритмов: жадные алгоритмы

Линейная

Разделяй и властвуй

Динамическое программирование

Реализация «в ширину» и «в глубину»

Жадные алгоритмы - дам пример в самом конце лекции

Жадные алгоритмы, получая первое же подходящее (и, вполне вероятно, не оптимальное) решение, отказываются от рассмотрения всех остальных, принимая его за конечное решение

Чем определяется сложность алгоритма?

Сложность алгоритма определяется **количеством** некоторых операций в зависимости от числа обрабатываемых элементов.

«О большое» – математическое обозначение для сравнения асимптотического поведения функций

$$f(n) = O(g(n))$$

Такая запись читается следующим образом: функция $f(n)$ асимптотически ограничена сверху функцией $g(n)$

$O(N)$
 $O(N \cdot \log N)$
 $O(N+M)$
 $O(N^2)$

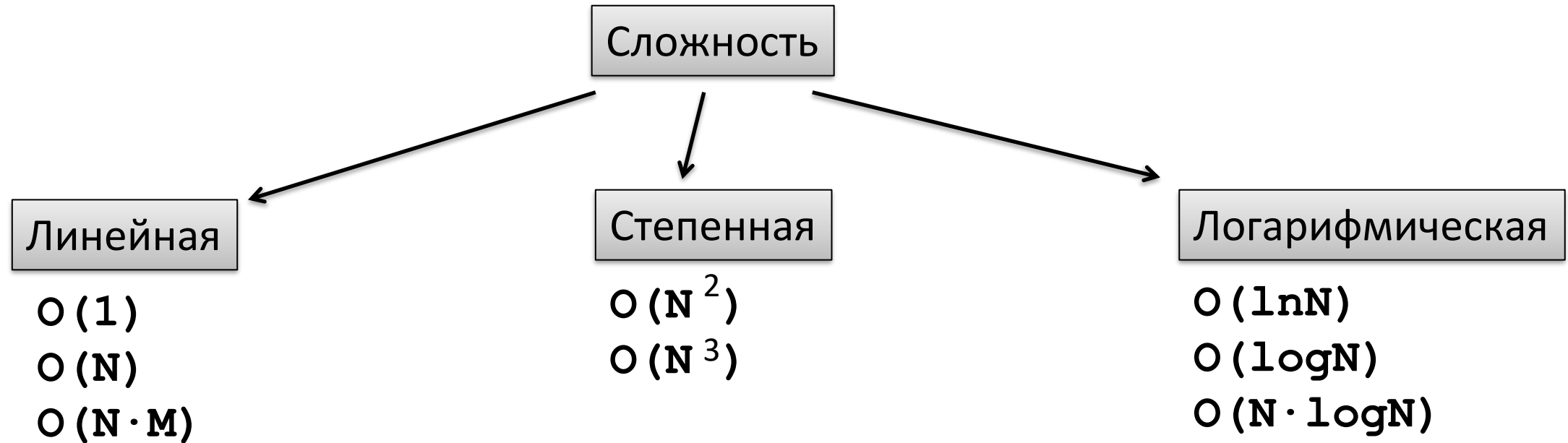
Алгоритмическая сложность может измеряться для:

- худшего случая;
- среднего случая;
- лучшего случая.

Подобный принцип может применяться для:

- количества операций;
- требуемого объёма памяти.

Примеры обозначения сложностей алгоритмов



сложность доступа к элементу массива
 $O(1)$

сложность доступа к элементу списка
 $O(N)$

сложность обхода элементов матрицы
 $O(N^2)$

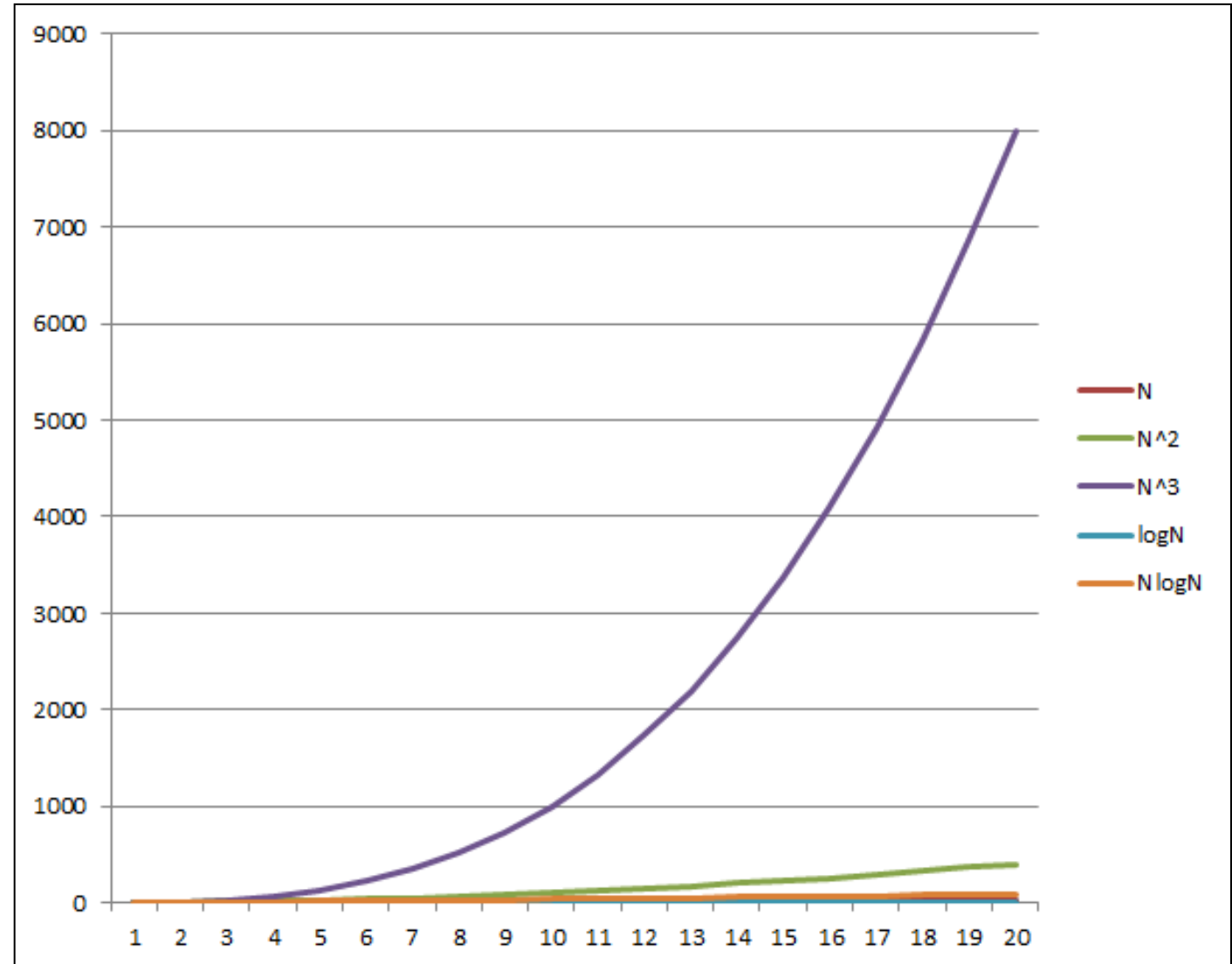
сложность плохих алгоритмов сортировки
 $O(N^2)$

сложность хороших алгоритмов поиска и сортировки
 $O(N \log N)$

Сравнение сложностей алгоритмов (1)

График роста функции «О-большое»

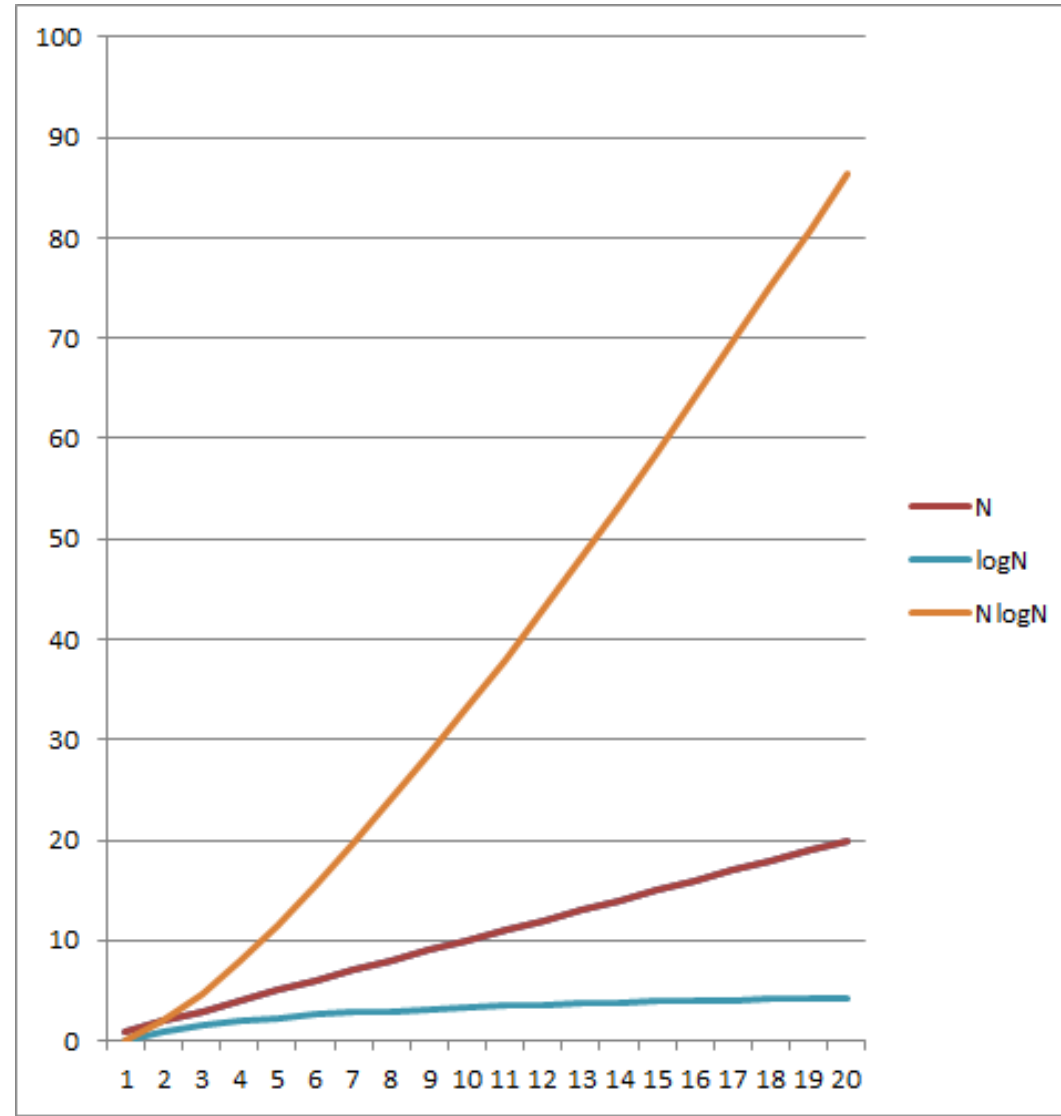
Число	N	N ²	N ³	logN	NlogN
1	1	1	1	0,00000	0,00000
2	2	4	8	1,00000	2,00000
3	3	9	27	1,5849625	4,7548875
4	4	16	64	2,00000	8,00000
5	5	25	125	2,3219281	11,60964
6	6	36	216	2,5849625	15,509775
7	7	49	343	2,8073549	19,651484
8	8	64	512	3,00000	24,00000
9	9	81	729	3,169925	28,529325
10	10	100	1000	3,3219281	33,219281
11	11	121	1331	3,4594316	38,053748
12	12	144	1728	3,5849625	43,01955
13	13	169	2197	3,7004397	48,105716
14	14	196	2744	3,8073549	53,302969
15	15	225	3375	3,9068906	58,603359
16	16	256	4096	4,00000	64,00000
17	17	289	4913	4,0874628	69,486868
18	18	324	5832	4,169925	75,05865
19	19	361	6859	4,2479275	80,710623
20	20	400	8000	4,3219281	86,438562



Сравнение сложностей алгоритмов (2)

График роста функции «О-большое»

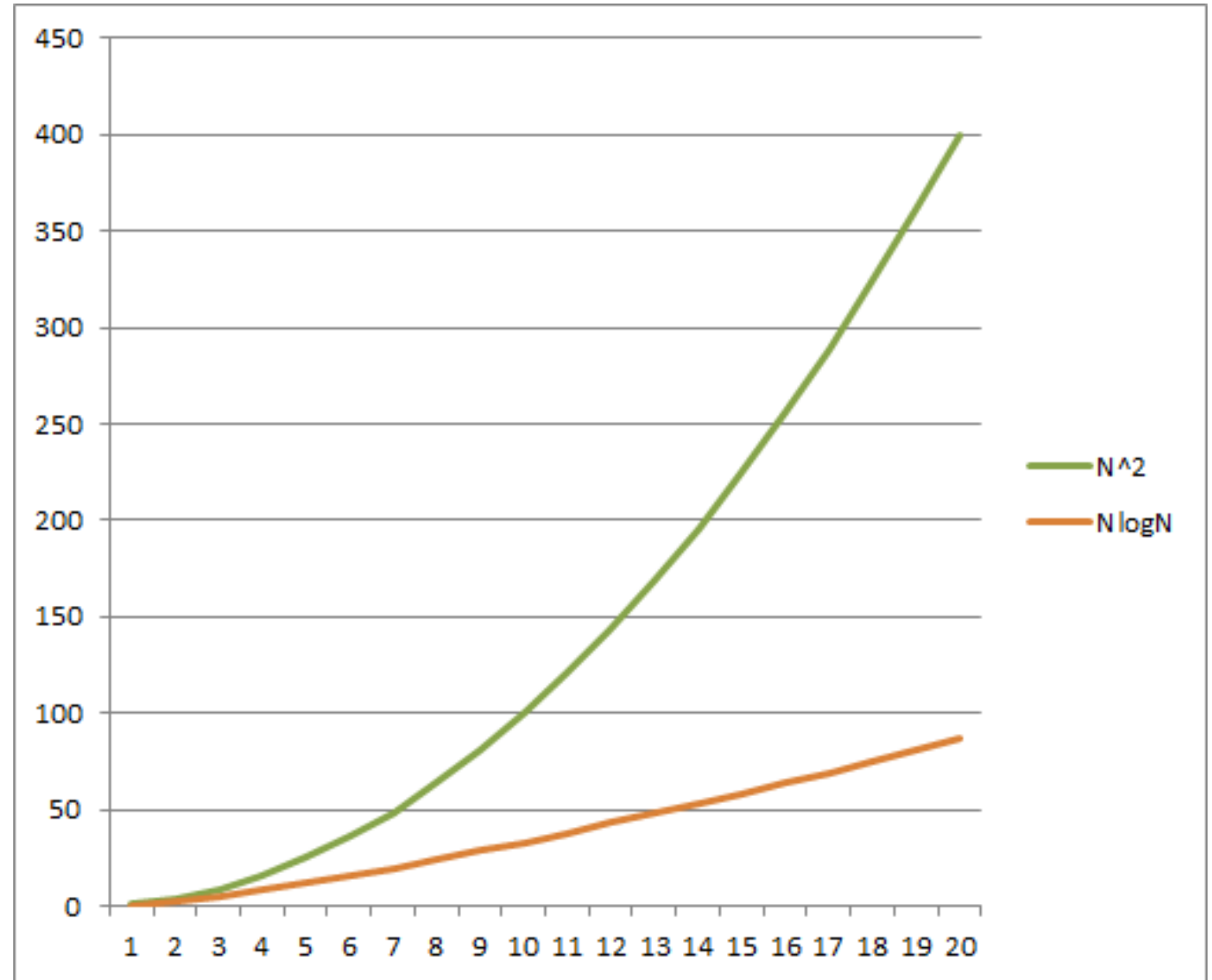
Число	N	logN	NlogN
1	1	0,00000	0,00000
2	2	1,00000	2,00000
3	3	1,5849625	4,7548875
4	4	2,00000	8,00000
5	5	2,3219281	11,60964
6	6	2,5849625	15,509775
7	7	2,8073549	19,651484
8	8	3,00000	24,00000
9	9	3,169925	28,529325
10	10	3,3219281	33,219281
11	11	3,4594316	38,053748
12	12	3,5849625	43,01955
13	13	3,7004397	48,105716
14	14	3,8073549	53,302969
15	15	3,9068906	58,603359
16	16	4,00000	64,00000
17	17	4,0874628	69,486868
18	18	4,169925	75,05865
19	19	4,2479275	80,710623
20	20	4,3219281	86,438562



Сравнение сложностей алгоритмов (3)

График роста функции «О-большое»

Число	N^2	$N \log N$
1	1	0,00000
2	4	2,00000
3	9	4,7548875
4	16	8,00000
5	25	11,60964
6	36	15,509775
7	49	19,651484
8	64	24,00000
9	81	28,529325
10	100	33,219281
11	121	38,053748
12	144	43,01955
13	169	48,105716
14	196	53,302969
15	225	58,603359
16	256	64,00000
17	289	69,486868
18	324	75,05865
19	361	80,710623
20	400	86,438562



Волновой алгоритм (алгоритм Ли) нахождения кратчайшего пути

Задача: найти кратчайшее расстояние между двумя точками на плоскости

A	1	2	3	4	5
1	2		4	5	6
	3		5	6	7
5	4		6		
6	5	6	7	8	B

Шаг 1:

нанести на поле сетку, препятствия, начальную (A) и конечную (B) точки

Шаг 2:

проставить первую волну вокруг начальной точки A

Шаг 3:

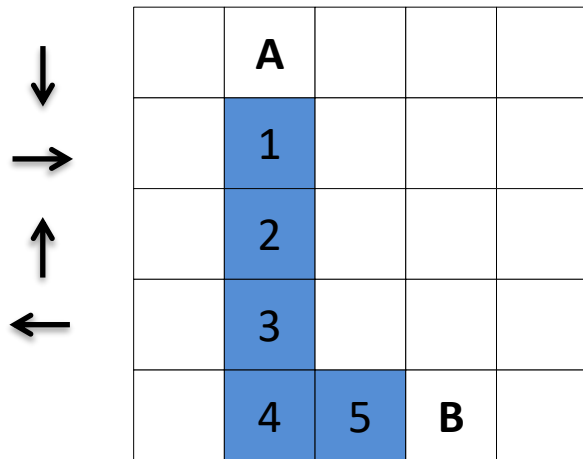
запустить цикл обхода всей матрицы, проставляя следующие номера волны вокруг текущего номера, пока не дойдём до финиша (B)

Шаг 4:

найти обратный путь по наименьшим номерам, начиная с точки финиша (B)

Реализация алгоритма «в глубину»

Поиск в глубину. Вариант 1



Поиск в глубину. Вариант 2

