ФГОБУ ВПО "СибГУТИ"
Кафедра вычислительных систем

Дисциплины
"ЯЗЫКИ ПРОГРАММИРОВАНИЯ"
"ПРОГРАММИРОВАНИЕ"

Примеры алгоритмов обработки массивов

Преподаватель:

Доцент Кафедры ВС, к.т.н.

Поляков Артем Юрьевич

План лекции

1. Задача сортировки и алгоритмы ее решения

- 1.1. Мера упорядоченности последовательности
- 1.2. Задача сортировки
- 1.3. Алгоритм сортировки выбором
- 1.4. Алгоритм сортировки вставками
- 1.5. Алгоритм сортировки методом пузырька

2. Задача поиска простых чисел в заданном диапазоне

- 2.1. Линейный алгоритм
- 2.2. Алгоритм Эратосфена

Задача сортировки и алгоритмы ее решения

Мера упорядоченности последовательности

8	15	4	30	3	17	6	29	12	27

3	4	6	8	12	15	17	27	29	30

3	4	6	29	12	15	17	27	8	30

15	4	30	29	12	3	17	27	8	6

30	29	27	17	15	12	8	6	4	3

Количественная мера упорядоченности

Инверсией элементов i и j последовательности a называется ситуация, когда $a[i] > a[j]$ и $i < j$.

1	2	3	4	5	6	7	8	9	10
3	**4**	**6**	**29**	**12**	**15**	**17**	**27**	**8**	**30**

$a[4] = 29$, $a[9] = 8$: $4 < 9$, $29 > 8$.

Общее количество инверсией определяет степень упорядоченности последовательности. Для вычисления этого показателя элементы последовательности перебираются слева направо, для каждого элемента вычисляется количество его инверсий с элементами, расположенными правее.

Количественная мера упорядоченности (2)

1	2	3	4	5	6	7	8	9	10
3	**4**	**6**	**29**	**12**	**15**	**17**	**27**	**8**	**30**

1) *a*[1] – a[3] : количество инверсий 0;

2) *a*[4] = 29: инверсии с элементами a[5] – a[9] = 5;

3) a[5] – a[8]: инверсия с элементом a[9], в сумме = 4;

4) a[9]: инверсий нет.

5) a[10]: инверсий нет.

Итого: 9 инверсий.

Мера упорядоченности последовательности

8	15	4	30	3	17	6	29	12	27

абсолютное: 16 инверсий
относит: 0,356

абсолютное: 0 инверсий
относит: 0

3	4	6	8	12	15	17	27	29	30

3	4	6	29	12	15	17	27	8	30

абсолютное: 9 инверсий
относит: 0,2

абсолютное: 24 инверсий
относит: 0,533

15	4	30	29	12	3	17	27	8	6

30	29	27	17	15	12	8	6	4	3

абсолютное: 45 инверсий
относит: 1

Задача сортировки (sorting problem)

Дано: последовательность из n чисел $\langle a_1, a_2, a_3, \ldots, a_n \rangle$

Необходимо: переставить элементы последовательности так, чтобы для любых элементов новой последовательности $\langle a'_1, a'_2, a'_3, \ldots, a'_n \rangle$ выполнялось соотношение:

$$a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$$ (сортировка по возрастанию)

Алгоритм сортировки выбором

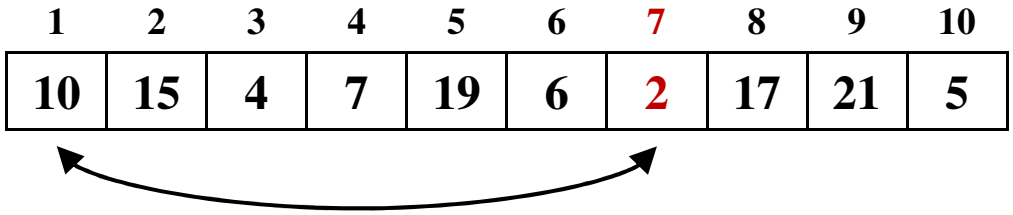
5	3	8	9	4
5	3	8	4	**9**
5	3	4	**8**	**9**
4	3	**5**	**8**	**9**

3	**4**	**5**	**8**	**9**
3	**4**	**5**	**8**	**9**

Алгоритм сортировки выбором

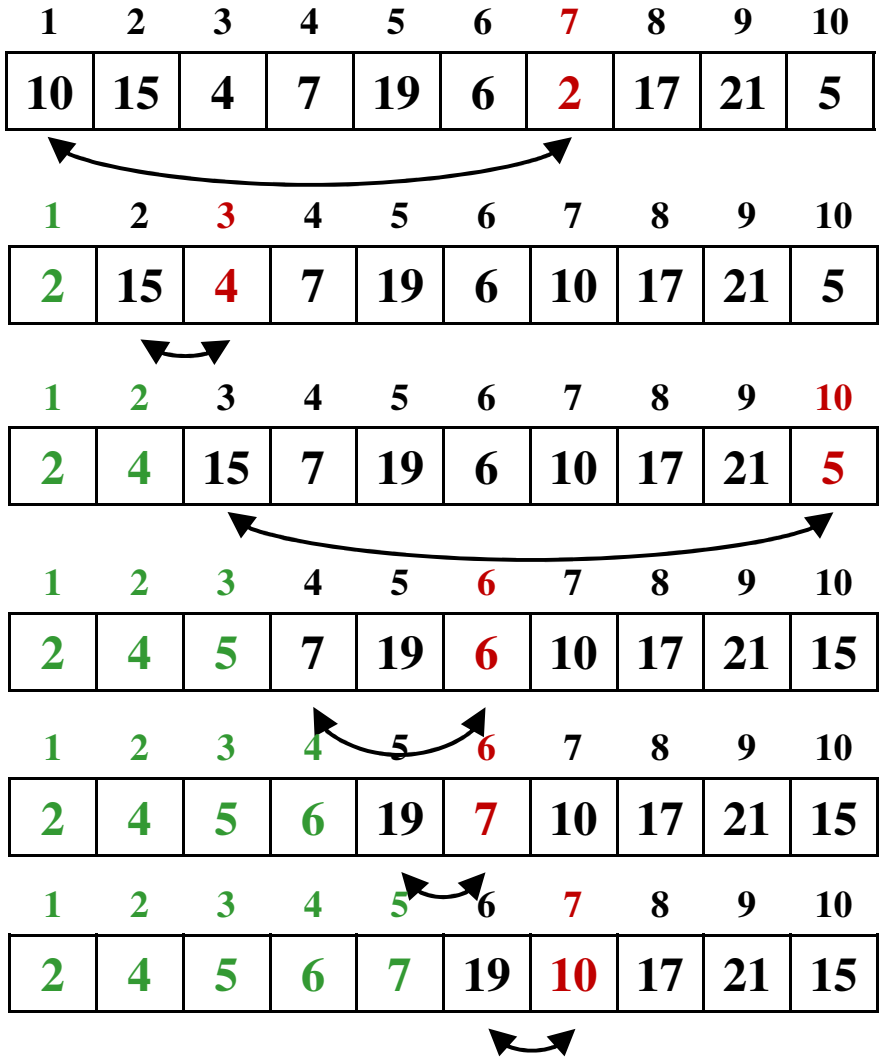
Суть алгоритма заключается в последовательном формировании упорядоченной последовательности слева направо.

На каждом шаге рассматривается фрагмент массива с i по n-й элементы. Среди них выбирается наименьший, который занимает первое место диапазона (на место i-го элемента). При этом i-й элемент перемещается на позицию, в которой найден минимум:



После этого считается, что элементы массива с 1 по i отсортированы. Поэтому далее процедура повторяется для диапазона элементов с i+1 по n.

Алгоритм сортировки выбором (2)



$i = 1$, **min**$(a[k], k = i \ldots n) = 2$

inv = 22

$i = 2$, **min**$(a[k], k = i \ldots n) = 4$

inv = 15

$i = 3$, **min**$(a[k], k = i \ldots n) = 5$

inv = 14

$i = 4$, **min**$(a[k], k = i \ldots n) = 6$

inv = 6

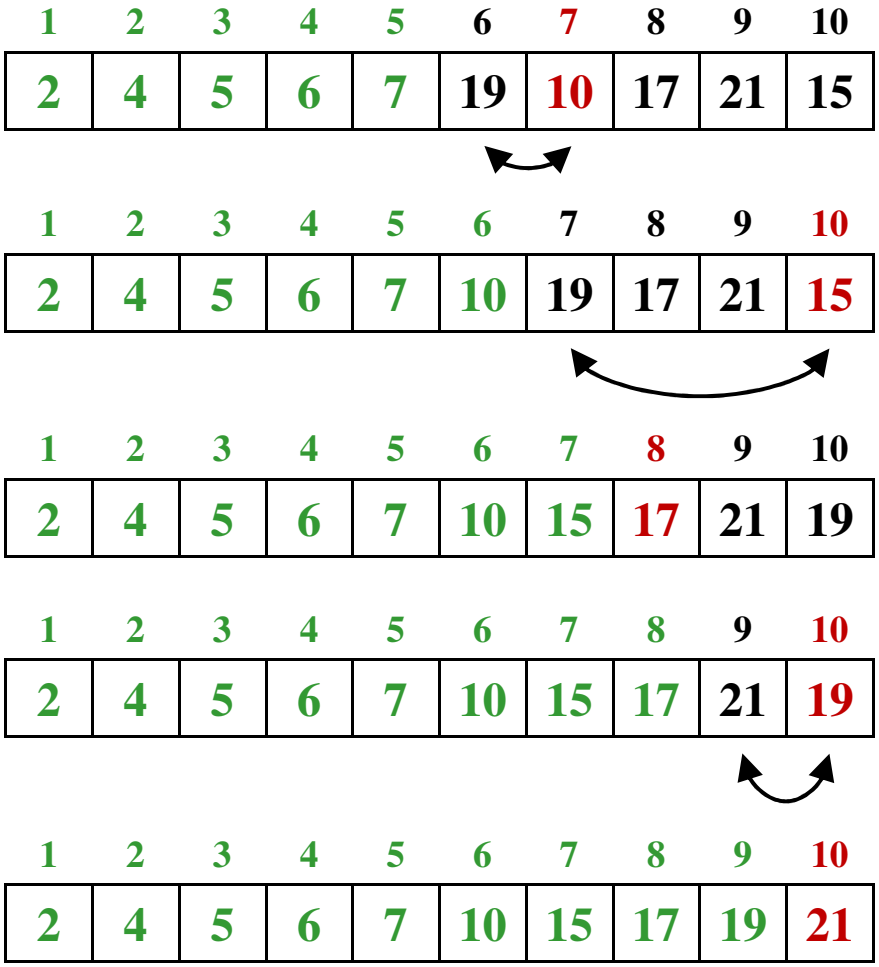
$i = 5$, **min**$(a[k], k = i \ldots n) = 7$

inv = 5

$i = 6$, **min**$(a[k], k = i \ldots n) = 10$

inv = 4

Алгоритм сортировки выбором (3)



$i = 6$, **min**$(a[k], k = i \ldots n) = 10$

inv = 4

$i = 7$, **min**$(a[k], k = i \ldots n) = 15$

inv = 1

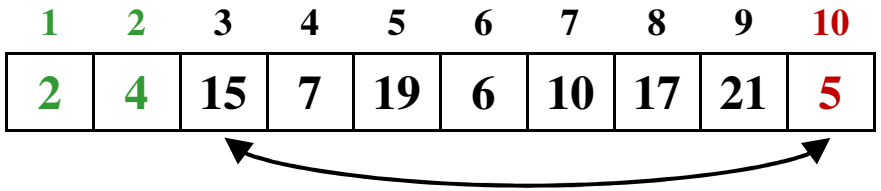
$i = 8$, **min**$(a[k], k = i \ldots n) = 17$

inv = 1

$i = 9$, **min**$(a[k], k = i \ldots n) = 19$

inv = 0

Проход алгоритма сортировки выбором



$i = 3$, **min**$(a[k], k = i \ldots n) = 5$

Основной подзадачей, возникающей в процессе сортировки, является поиск **индекса** минимального элемента в диапазоне с i по n (в примере на рис. диапазон – с 3 по 10).

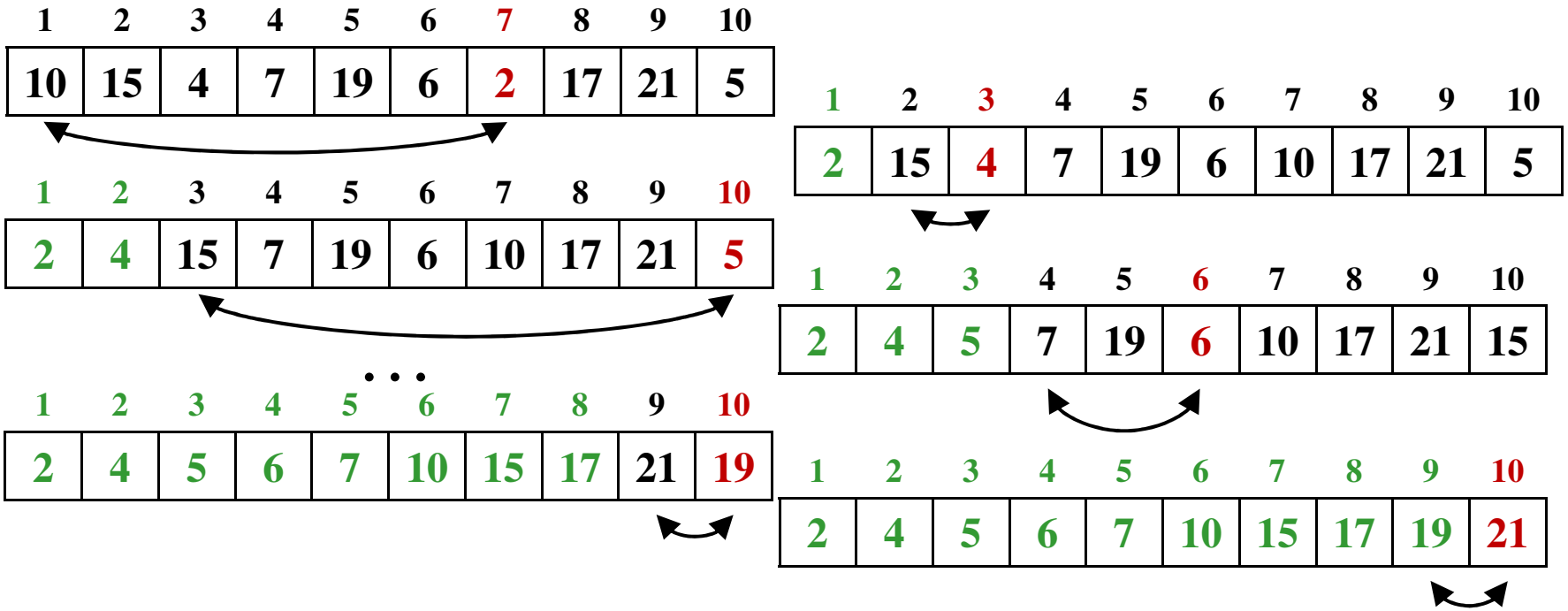
Для решения этой задачи можно воспользоваться уже известным алгоритмом поиска минимального элемента:

$m \leftarrow i, k \leftarrow i + 1$
while $k \le n$ **do**
  if $a[k] < a[m]$ **then**
    $m \leftarrow k$
  $k \leftarrow k + 1$
$t \leftarrow a[i], a[i] \leftarrow a[m], a[m] \leftarrow t$

Приведенный фрагмент называется "проходом" данного алгоритма.

Алгоритм сортировки выбором (4)

После каждого прохода алгоритма сортировки выбором размер отсортированной части увеличивается на 1 элемент:



Поэтому, учитывая, что на последнем проходе рассматривается один элемент, количество проходов составляет $n - 1$.

Алгоритм сортировки выбором (псевдокод)

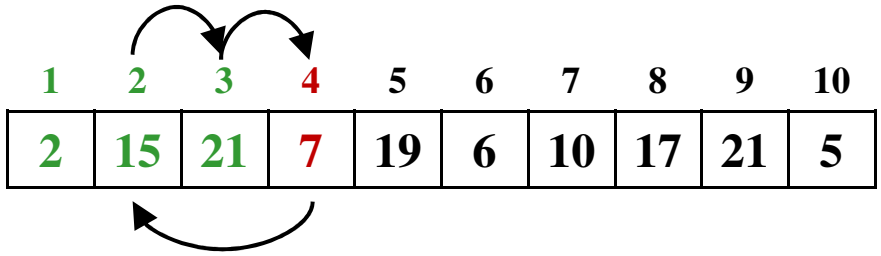
Учитывая, что на последнем проходе рассматривается один элемент, количество проходов для получения полностью отсортированной последовательности составляет $n - 1$.

```
i ← 1
while i < n do
    m ← i, k ← i + 1
    while k ≤ n do
        if a[k] < a[m] then
            m ← k
        k ← k + 1
    t ← a[i], a[i] ← a[m], a[m] ← t
    i ← i + 1
```

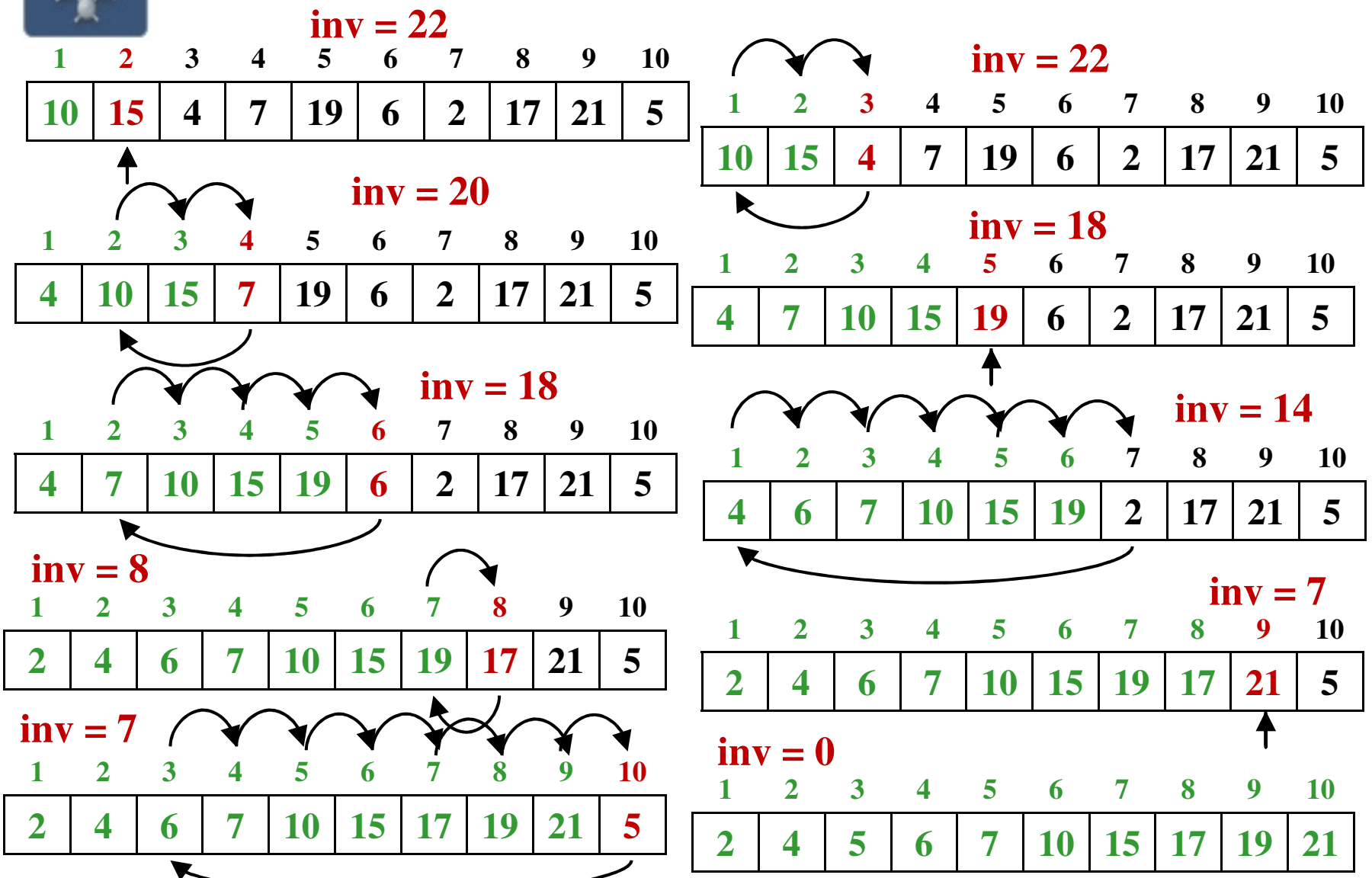
Алгоритм сортировки вставками

Суть данного алгоритма (аналогично предыдущему) заключается в последовательном формировании упорядоченной последовательности слева направо. Однако, в отличие от сортировки вставками, основная работа осуществляется в отсортированной части массива.

На каждом шаге имеется отсортированная часть последовательности с 1 по $(i-1)$ и не отсортированная – с i по n элементы. Для первого не отсортированного элемента с номером i в отсортированной части ищется подходящая позиция, как показано на рисунке:



Алгоритм сортировки вставками (2)

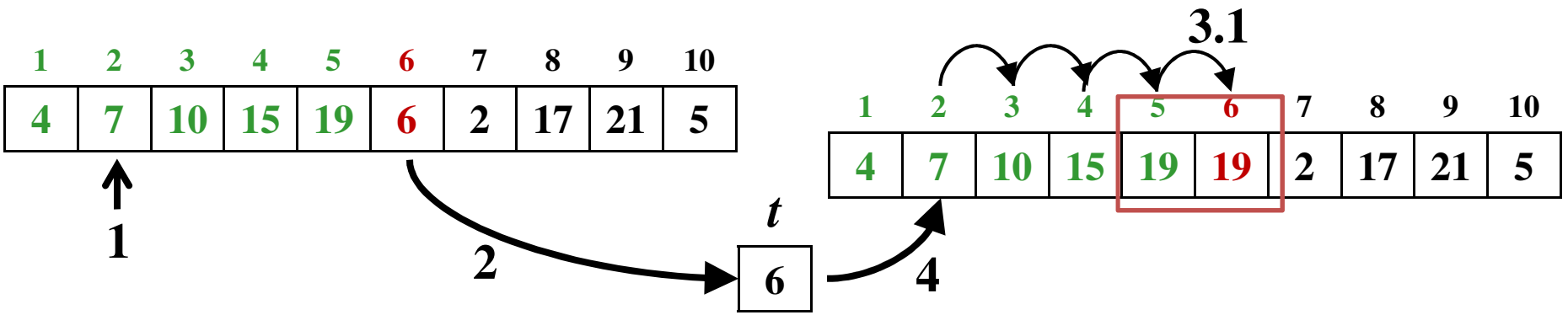


Проход алгоритма сортировки вставками

Основной подзадачей, возникающей в процессе сортировки, является поиск для очередного элемента правильного места в уже отсортированной части массива и вставка его на эту позицию.

Решение этой задачи можно разбить на следующие шаги:

1. Поиск индекса подходящей позиции k для элемента $a[i]$.
2. Запоминание элемента $a[i]$ во временной ячейке t: $t = a[i]$.
3. Сдвиг элементов $a[k]$, $a[k+1]$, …, $a[i-1]$ на один элемент вправо, т.е. в ячейки $a[k+1]$, $a[k+2]$, …, $a[i]$ соответственно.
4. Поместить значение из ячейки t в $a[k]$: a[k] = t.



Проход алгоритма сортировки вставками (2)

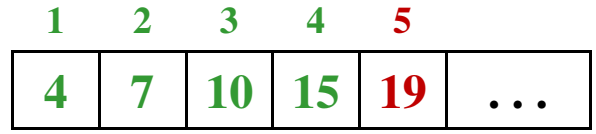
1. Поиск индекса подходящей позиции k для элемента $a[i]$.

2. Запоминание элемента $a[i]$ во временной ячейке t: $t = a[i]$.

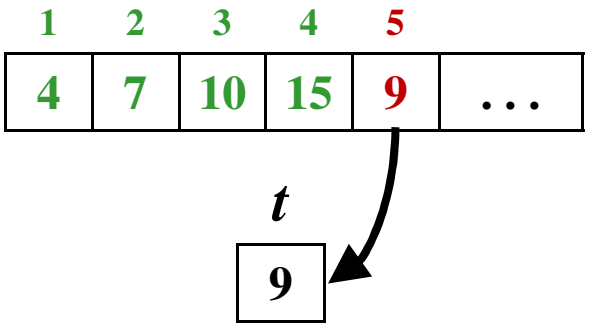
3. Сдвиг элементов $a[k]$, $a[k+1]$, …, $a[i-1]$ на один элемент вправо, т.е. в ячейки $a[k+1]$, $a[k+2]$, …, $a[i]$ соответственно.

4. Поместить значение из ячейки t в $a[k]$: a[k] = t.

На практике поиск элемента часто объединяют со сдвигом элементов:



- $a[5] > a[4]$ => сдвиг не нужен



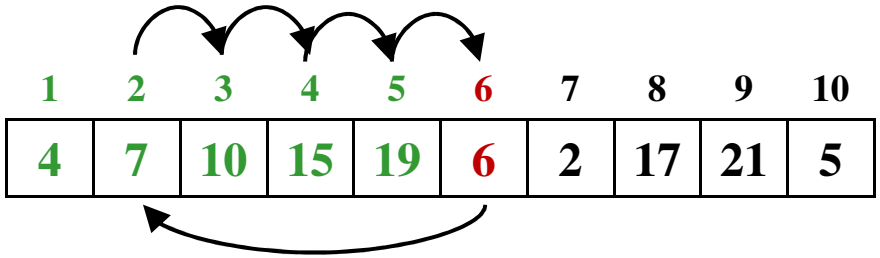
$t = a[5]$

1) $t < a[4]$, сдвиг: $a[5] = a[4]$;

2) $t < a[3]$, сдвиг: $a[3] = a[4]$;

3) $t > a[2]$, стоп: $a[3] = t$;

Проход алгоритма сортировки вставками (3)



Псевдокод одного прохода алгоритма сортировки вставками выглядит следующим образом:

$t \leftarrow a[i],\ k \leftarrow i - 1$

while $k \geq 1$ **и** $a[k] > t$ **do**

$a[k+1] \leftarrow a[k]$

$k \leftarrow k - 1$

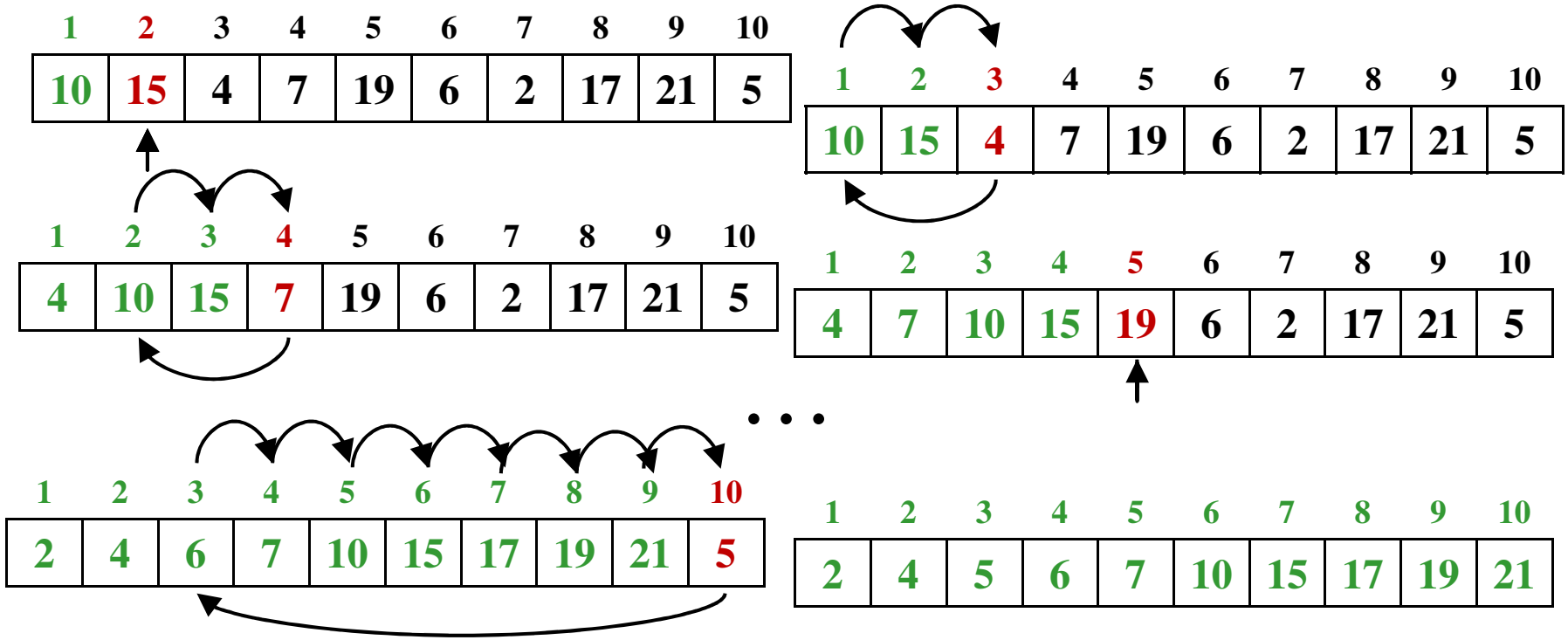
$a[k+1] \leftarrow t$

1	2	3	4	5	6		
4	7	10	15	19	19	. . .	$k = 5\ a[6] = a[5]$
1	2	3	4	5	6		
4	7	10	15	15	19	. . .	$k = 4\ a[5] = a[4]$
1	2	3	4	5	6		
4	7	10	10	15	19	. . .	$k = 3\ a[4] = a[3]$
1	2	3	4	5	6		
4	7	9	10	15	19	. . .	$k = 2\ a[3] = t$

Приведенный фрагмент называется "проходом" данного алгоритма.

Алгоритм сортировки вставками (3)

После каждого прохода алгоритма сортировки выбором размер отсортированной части увеличивается на 1 элемент:



Таким образом для сортировки всей последовательности требуется $n - 1$ проход.

Алгоритм сортировки вставками (псевдокод)

Таким образом для сортировки всей последовательности требуется $n - 1$ проход.

$t \leftarrow \mathrm{a}[i],\ k \leftarrow i - 1$

while $k \geq 1$ **и** $a[k] > t$ **do**

$a[k+1] \leftarrow a[k]$

$k \leftarrow k - 1$

$a[k+1] \leftarrow t$

$i \leftarrow 2$

while $i < n$ **do**

$t \leftarrow \mathrm{a}[i],\ k \leftarrow i - 1$

while $k \geq 1$ **и** $a[k] > t$ **do**

$a[k+1] \leftarrow a[k]$

$k \leftarrow k - 1$

$a[k+1] \leftarrow t$

$i \leftarrow i + 1$

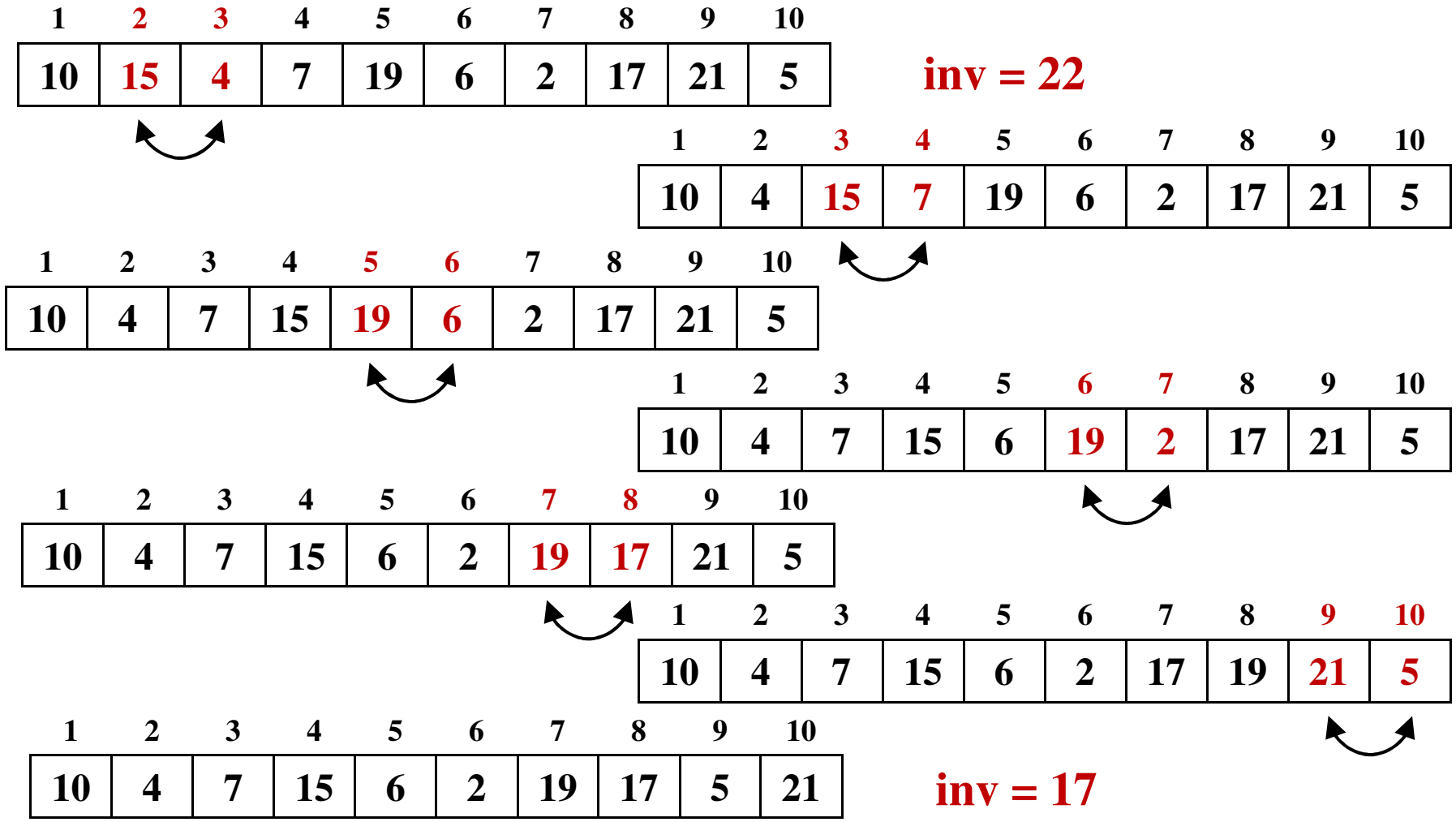
Алгоритм сортировки методом пузырька

Суть данного заключается в том, что на каждом проходе определяется наибольший элемент, который выводится на последнее место рассматриваемой подпоследовательности и исключается из дальнейшего рассмотрения. Важным отличием данного алгоритма от ранее рассмотренных является то, что в процессе выведения наибольшего элемента участвуют все элементы и количество инверсий сокращается более, чем на вклад наибольшего элемента.

Проход алгоритма заключается в попарном сравнении соседних элементов и устранении "локальных" инверсий. В результате такого прохода наибольший элемент (пузырек наибольшего размера) выдвигается в крайнюю правую позицию.

1	2	3	4	5	6	7	8	9	10
10	15	4	7	19	6	2	17	21	5

Проход алгоритма сортировки методом пузырька



Алгоритм сортировки методом пузырька (3)

1	2	3	4	5	6	7	8	9	10	
10	15	4	7	19	6	2	17	21	5	**inv = 22**
1	2	3	4	5	6	7	8	9	10	
10	4	7	15	6	2	19	17	5	21	**inv = 17**
1	2	3	4	5	6	7	8	9	10	
4	7	10	6	2	15	17	5	19	21	**inv = 11**
1	2	3	4	5	6	7	8	9	10	
4	7	6	2	10	15	5	17	19	21	**inv = 8**
1	2	3	4	5	6	7	8	9	10	
4	6	2	7	10	5	15	17	19	21	**inv = 5**
1	2	3	4	5	6	7	8	9	10	
4	2	6	7	5	10	15	17	19	21	**inv = 3**
1	2	3	4	5	6	7	8	9	10	
2	4	6	5	7	10	15	17	19	21	**inv = 1**

Алгоритм сортировки методом пузырька (4)

1	2	3	4	5	6	7	8	9	10	
4	2	6	7	5	10	15	17	19	21	**inv = 3**

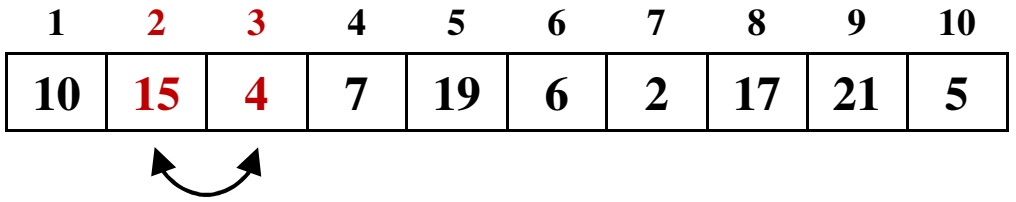
1	2	3	4	5	6	7	8	9	10	
2	4	6	5	7	10	15	17	19	21	**inv = 1**

1	2	3	4	5	6	7	8	9	10	
2	4	5	6	7	10	15	17	19	21	**inv = 0**

1	2	3	4	5	6	7	8	9	10	
2	4	5	6	7	10	15	17	19	21	**inv = 0**

Особенностью алгоритма сортировки методом пузырька является то, что существует возможность *досрочного прекращения* сортировки при условии, что на очередном проходе не было выполнено ни одного обмена.

Проход алгоритма сортировки методом пузырька (2)



$j \leftarrow 1$

while $j \leq (n - i)$ **do**

  if $a[j+1] < a[j]$ **then**

    $t \leftarrow a[j+1]$

    $a[j+1] \leftarrow a[j]$

    $a[j] \leftarrow t$

  $j \leftarrow j + 1$

Алгоритм сортировки методом пузырька (5)

После каждого прохода алгоритма сортировки методом пузырька наибольший размер отсортированной правой части увеличивается на 1 элемент:

1	2	3	4	5	6	7	8	9	10	
10	15	4	7	19	6	2	17	21	5	**inv = 22**
1	2	3	4	5	6	7	8	9	10	
10	4	7	15	6	2	19	17	5	21	**inv = 17**

• • •

1	2	3	4	5	6	7	8	9	10	
2	4	5	6	7	10	15	17	19	21	**inv = 0**

Таким образом для сортировки всей последовательности требуется $n - 1$ проход.

Алгоритм сортировки методом пузырька (псевдокод)

Таким образом для сортировки всей последовательности требуется $n - 1$ проход.

```
i = 1
while i < n – 1 do
    j ← 1
    while j ≤ (n – i) do
        if a[j+1] < a[j] then
            t ← a[j+1]
            a[j+1] ← a[j]
            a[j] ← t
        j ← j + 1
    i ← i + 1
```

```
j ← 1
while j ≤ (n – i) do
    if a[j+1] < a[j] then
        t ← a[j+1]
        a[j+1] ← a[j]
        a[j] ← t
    j ← j + 1
```

Алгоритм сортировки методом пузырька с досрочным выходом (псевдокод)

Таким образом для сортировки всей последовательности требуется $n - 1$ проход.

$i = 1, f \leftarrow 1$
while $i < n - 1$ и $f > 0$ **do**
  $j \leftarrow 1, f \leftarrow 0$
  while $j \leq (n - i)$ **do**
    if $a[j+1] < a[j]$ **then**
      $t \leftarrow a[j+1]$
      $a[j+1] \leftarrow a[j]$
      $a[j] \leftarrow t$
      $f \leftarrow 1$
    $j \leftarrow j + 1$
  $i \leftarrow i + 1$

Задача поиска простых чисел в заданном диапазоне

Простые числа

Простое число – это натуральное число, имеющее ровно два различных натуральных делителя: единицу и само себя.

Примеры простых чисел:

2, 13, 59, 101, 431, 733, 1153, 2069 и др.

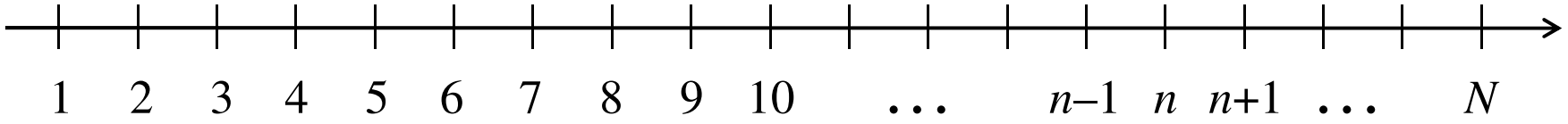
Все остальные натуральные числа, кроме единицы, называются *составными*.

Все натуральные числа больше единицы разбиваются на простые и составные.

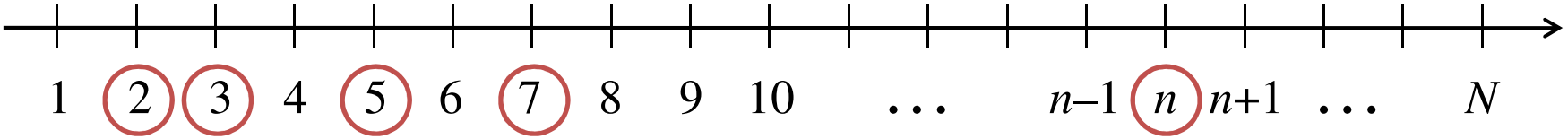
Изучением свойств простых чисел занимается теория чисел.

Список первых 500 простых чисел:
http://ru.wikipedia.org/wiki/Список_простых_чисел.

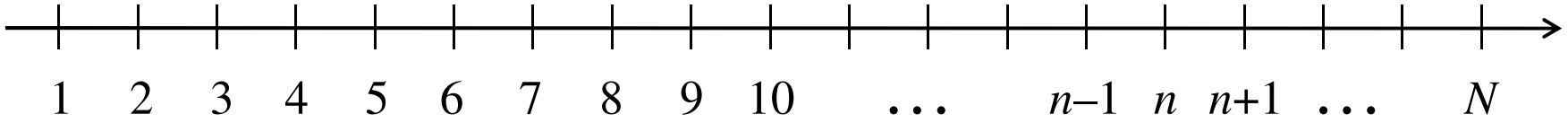
Задача поиска простых чисел в заданном диапазоне



Дан диапазон натуральных чисел [2, *N*]. Требуется выбрать из него только те числа, которые являются простыми:



Линейный алгоритм поиска простых чисел в заданном диапазоне



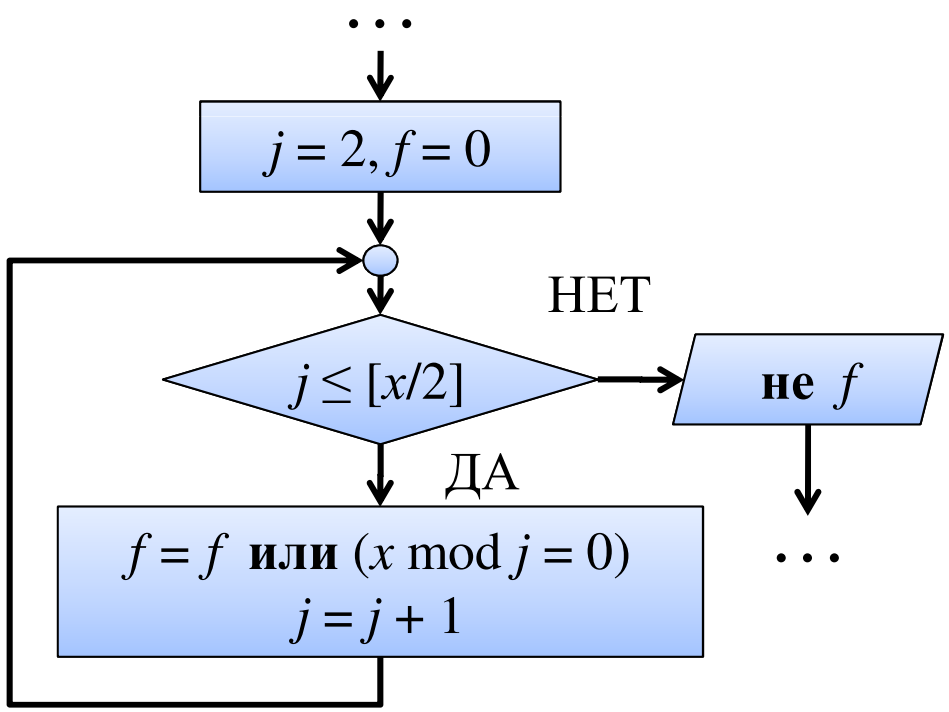
Наиболее простым алгоритмом решения задачи является перебор всех чисел диапазона [2, *N*] и проверка каждого из них на простоту линейным алгоритмом.

Для того, чтобы проверить, является ли число *x* простым достаточно проверить, делится ли оно на одно из чисел диапазона [2, *x*/2]. Упрощенный вариант этого алгоритма уже рассматривался при *изучении циклов*.

Числа, прошедшие проверку на простоту могут быть помещены в специально отведенный для этого массив, размерность которого совпадает с исходным (пессимистическое выделение памяти).

Проверка на простоту

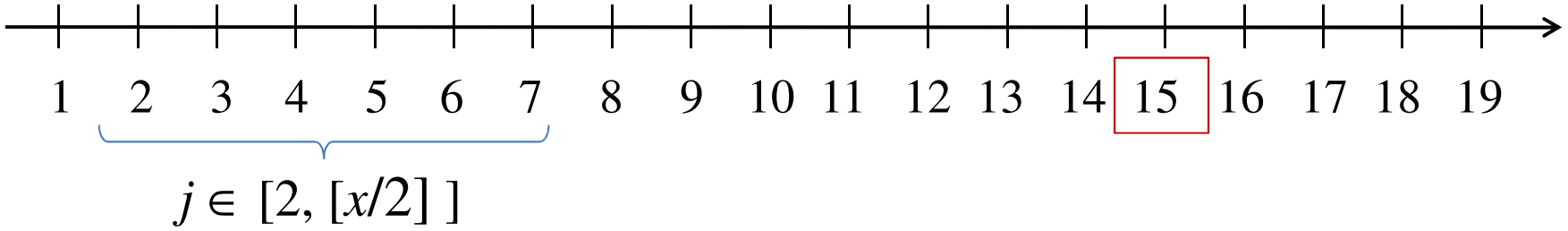
x – целое число, которое проверяется на простоту.
На каждой итерации проверяется потенциальный делитель
$j \in [2, [x/2]]$.
Если x **mod** $j = 0$ (делится нацело), то *переменная-флаг* $f = 1$.

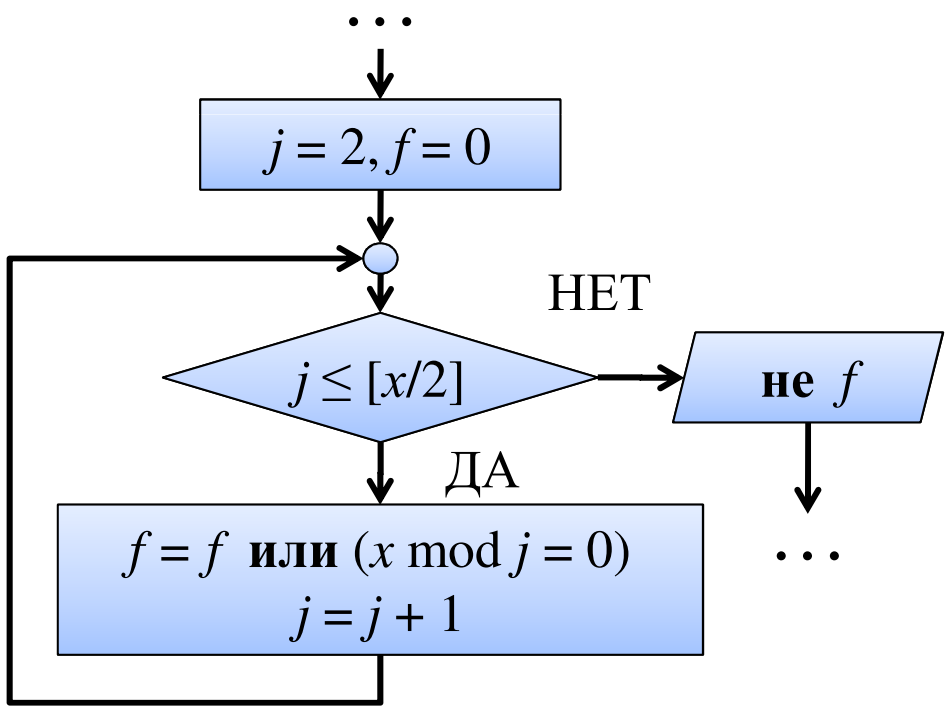


На выходе значение флага f определяет результат:

- если $f = 0$ ((**не** f) = 1), то среди чисел диапазона [2, [x/2]] не нашлось ни одного делителя – число простое.
- иначе $f = 1$ ((**не** f)= 0) – число составное.

Проверка на простоту (2)





На выходе значение флага f определяет результат:

- если $f = 0$ ((**не** f) = 1), то среди чисел диапазона [2, [x/2]] не нашлось ни одного делителя – число простое.
- иначе $f = 1$ ((**не** f)= 0) – число составное.

Линейный алгоритм поиска простых чисел в заданном диапазоне

$x \leftarrow 2$, $n \leftarrow 0$ // x – проверяемые числа, n – счетчик простых чисел

while $x \le N$ **do**

// j – потенциальные делители, f – флаг, 0 => простое число

$j \leftarrow 2, f \leftarrow 0$

while $j \le (x$ **div** $2)$ **do**

if $(x$ **mod** $j = 0)$ **then** // если x делится на j – x не простое!

$f \leftarrow 1$

$j \leftarrow j + 1$ // перейти к следующему делителю

if $f = 0$ **then** // если флаг $f = 0$, то x – простое число

$n \leftarrow n + 1$ // счетчик n – первый свободный элемент

$primes[n] \leftarrow x$

$x \leftarrow x + 1$

Как исключить ненужные операции?

Пути оптимизации

1. Если обнаружен хотя бы один делитель, число уже не является простым и дальнейшую проверку проводить не требуется.

2. Чтобы проверяемое число было простым достаточно, чтобы оно не делилось ни на одно из ранее найденных простых чисел.

Линейный алгоритм поиска простых чисел в заданном диапазоне (версия 2)

$x \leftarrow 2, n \leftarrow 0$ // x – проверяемые числа, n – счетчик простых чисел

while $x \le N$ **do**

// j – потенциальные делители, f – флаг, 0 => простое число

$j \leftarrow 2, f \leftarrow 0$

while $j \le$ (x **div** 2) **и** $f \ne 1$ **do**

if (x **mod** $j = 0$) **then** // если x делится на j – x не простое!

$f \leftarrow 1$

$j \leftarrow j + 1$ // перейти к следующему делителю

if $f = 0$ **then** // если флаг $f = 0$, то x – простое число

$n \leftarrow n + 1$ // счетчик n – первый свободный элемент

$primes[n] \leftarrow x$

$x \leftarrow x + 1$

Линейный алгоритм поиска простых чисел в заданном диапазоне (версия 3)

$x \leftarrow 2$, $n \leftarrow 0$ // x – проверяемые числа, n – счетчик простых чисел

while $x \leq N$ **do**

// j – потенциальные делители, f – флаг, 0 => простое число

$j \leftarrow 1, f \leftarrow 0$

while $j \leq n$ **и** $f \neq 1$ **do**

if (x **mod** $primes[j] = 0$) **then**

$f \leftarrow 1$

$j \leftarrow j + 1$ // перейти к следующему делителю

if $f = 0$ **then** // если флаг $f = 0$, то x – простое число

$n \leftarrow n + 1$ // счетчик n – первый свободный элемент

$primes[n] \leftarrow x$

$x \leftarrow x + 1$

Решето Эратосфена

W http://ru.wikipedia.org/wiki/Решето_Эратосфена.

Решето Эратосфена — алгоритм нахождения всех простых чисел до некоторого целого числа *N*, который приписывают древнегреческому математику Эратосфену Киренскому.

1. Выписать подряд все целые числа от двух до *N* (2, 3, 4, …, *N*).

2. Пусть переменная *p* изначально равна двум – первому простому числу.

3. Считая от *p* шагами по *p*, зачеркнуть в списке все числа от 2*p* до *n* кратные *p* (то есть числа 2*p*, 3*p*, 4*p*, …).

4. Найти первое незачеркнутое число в списке, большее чем *p*, и присвоить значению переменной *p* это число.

5. Повторять шаги 3 и 4, пока незачеркнутое число есть.

Решето Эратосфена. Алгоритм

1. Выписать подряд все целые числа от двух до N (2, 3, 4, …, N).

2 3 4 5 6 7 8 9 10 11 12 13 14 15 … *N*

2. Пусть переменная $p = 2$ (первому простому числу).

2 3 4 5 6 7 8 9 10 11 12 13 14 15 … *N*

step=2

3. Считая от p шагами по p, зачеркнуть в списке все числа от $2p$ до n кратные p (то есть числа $2p$, $3p$, $4p$, …).

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 … *N*

2*step 3*step 4*step 5*step 6*step 7*step

4. Найти первое незачеркнутое число в списке, большее чем p, и присвоить значению переменной p это число: $p = 3$.

5. Повторять шаги 3 и 4, пока незачеркнутое число есть.

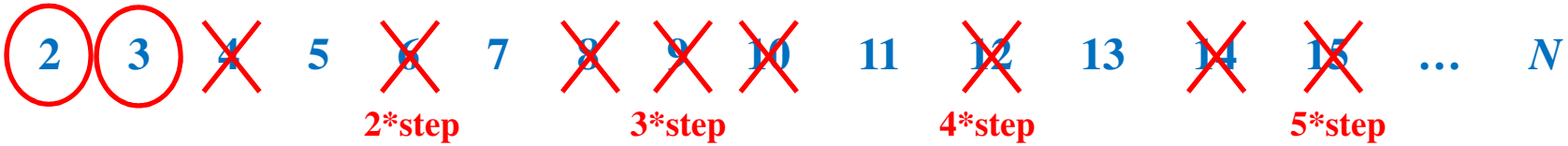
Решето Эратосфена. Алгоритм

1. Выписать подряд все целые числа от двух до N (2, 3, 4, …, N).

2 3 4 5 6 7 8 9 10 11 12 13 14 15 … *N*

2. $p = 3$

3. Считая от p шагами по p, зачеркнуть в списке все числа от $2p$ до n кратные p (то есть числа $2p$, $3p$, $4p$, …).



4. Найти первое незачеркнутое число в списке, большее чем p, и присвоить значению переменной p это число: $p = 5$.

5. Повторять шаги 3 и 4, пока незачеркнутое число есть.

Иллюстрация алгоритма Эратосфена

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

W http://ru.wikipedia.org/wiki/Решето_Эратосфена.

Псевдокод алгоритма Эратосфена

```
for i ← 1 to (N – 1) do // Заполнить массив числами диапазона
    primes[i] ← i + 1
i ← 1
while i < N do // основной цикл
    s ← primes[i]
    p ← i + s
    while p < N do // Вычеркивание (обнуление) не простых чисел
        primes[p] ← 0
        p ← p + s
    i ← i + 1
    while i < N и primes[i] = 0 do // Первое невычеркнутое
        i ← i + 1
```

Псевдокод алгоритма Эратосфена (оптимизир.)

for $i \leftarrow 1$ **to** $(N - 1)$ **do** // Заполнить массив числами диапазона
    $primes[i] \leftarrow i + 1$
$i \leftarrow 1$
while $i^2 < N$ **do** // основной цикл
    $s \leftarrow primes[i]$
    $p \leftarrow i + s$
    while $p < N$ **do** // Вычеркивание (обнуление) не простых чисел
        $primes[p] \leftarrow 0$
        $p \leftarrow p + s$
    $i \leftarrow i + 1$
    while $i < N$ и $primes[i] = 0$ **do** // Первое невычеркнутое
        $i \leftarrow i + 1$

Другие алгоритмы поиска простых чисел

W http://ru.wikipedia.org/wiki/Решето_Сундарама

В математике решето Сундарама — детерминированный алгоритм нахождения всех простых чисел до некоторого целого числа N. Разработан индийским студентом С. П. Сундарамом в 1934 году.

Из ряда натуральных чисел исключаются числа вида $i + j + 2ij$, где $i < j$.

W http://ru.wikipedia.org/wiki/Решето_Аткина

В математике решето Аткина — быстрый современный алгоритм нахождения всех простых чисел до заданного целого числа N. Основная идея алгоритма состоит в использовании неприводимых квадратичных форм (представление чисел в виде $ax^2 + by^2$).