Алгоритмы и структуры данных

Лекция 3
Алгоритмы сортировки

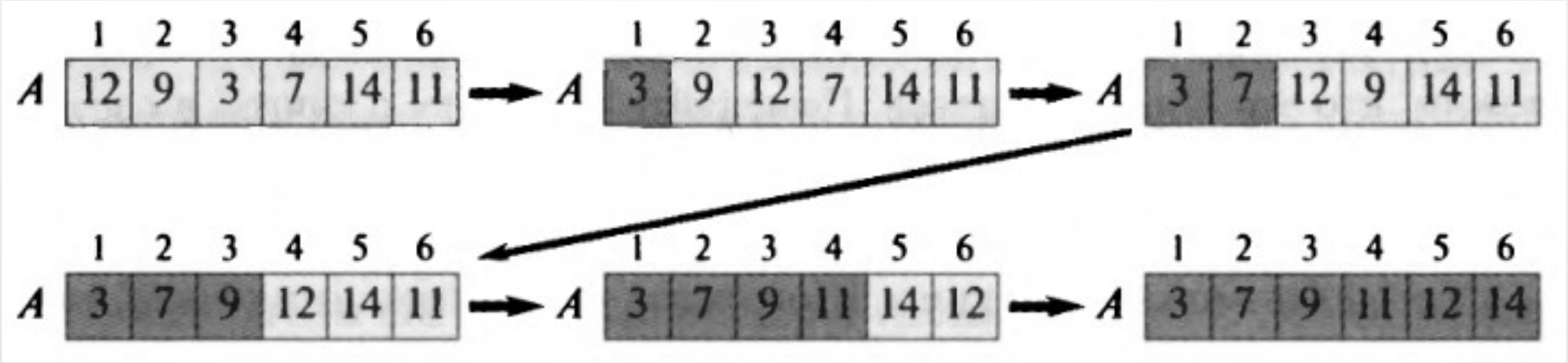
Алгоритмы сортировки

Сортировка — это упорядочение элементов массива по возрастанию или убыванию

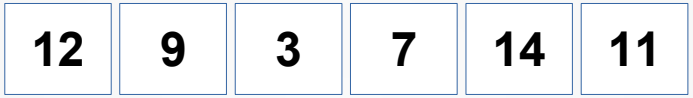
Алгоритм сортировки	Время работы в наихудшем случае	Время работы в наилучшем случае	Обменов в наихудшем случае	Выполняется ли сортировка на месте
Выбором	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	Да
Вставкой	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$	Да
Слиянием	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$	Нет
Быстрая	$\Theta(n^2)$	$\Theta(n \lg n)$	$\Theta(n^2)$	Да

Сортировка выбором

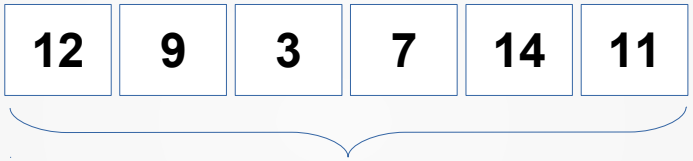
- Худшее время: $\Theta(n^2)$
- Лучшее время: $\Theta(n^2)$
- Среднее время: $\Theta(n^2)$
- Затраты памяти: $\Theta(n)$ всего, $\Theta(1)$ дополнительно



Сортировка выбором

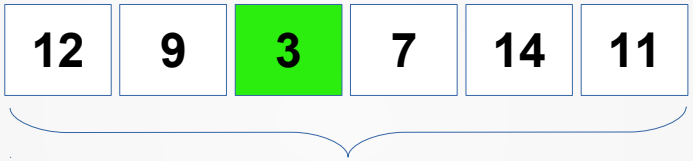


Сортировка выбором



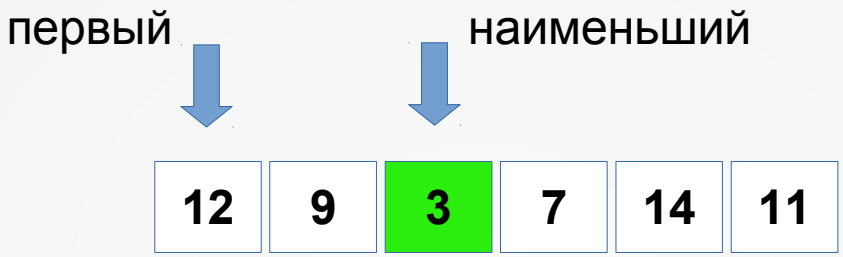
Поиск наименьшего элемента

Сортировка выбором



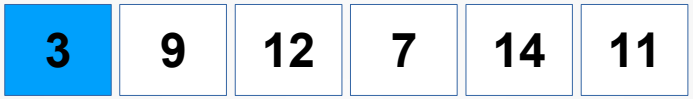
Поиск наименьшего элемента

Сортировка выбором



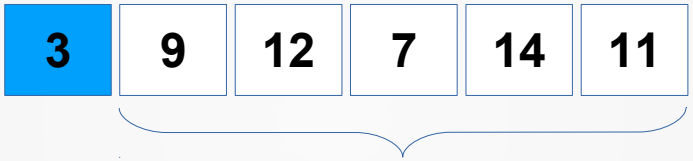
Обмен первого элемента с наименьшим

Сортировка выбором



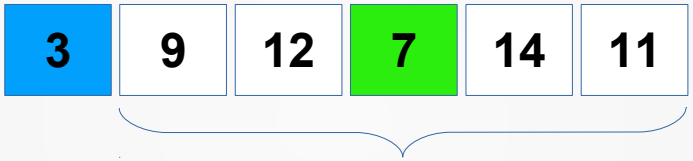
Синим отмечена отсортированная часть массива

Сортировка выбором



Поиск наименьшего элемента начиная со второго

Сортировка выбором



Поиск наименьшего элемента начиная со второго

Сортировка выбором

7 — наименьший элемент

Сортировка выбором

Поиск наименьшего элемента,
начиная с третьего

Сортировка выбором

Сортировка выбором

Сортировка выбором

Сортировка выбором

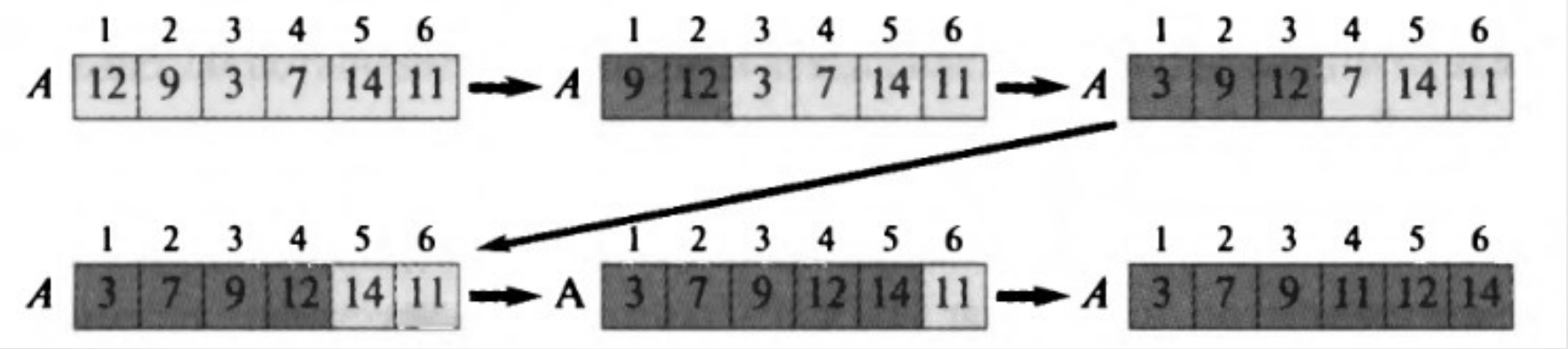
Сортировка выбором на C++

```
14  void selection_sort(int* a, int n)
15  {
16      for (int i=0; i<n; ++i)
17      {
18          int smallest = i;
19          for (int j=i+1; j<n; ++j)
20              if (a[j]<a[smallest])
21                  smallest = j;
22          swap(a, i, smallest);
23      }
24  }
```

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Сортировка вставками

- Худшее время: $\Theta(n^2)$
- Лучшее время: $\Theta(n)$
- Среднее время: $\Theta(n^2)$
- Затраты памяти: $\Theta(n)$ всего, $\Theta(1)$ дополнительно



Сортировка вставками

6 5 3 1 8 7 2 4

Сортировка вставками на C++

```
 5  void insertion_sort(int* a, int n)
 6  {
 7      for (int i=1; i<n; ++i)
 8      {
 9          int key = a[i];
10          int j = i-1;
11          while (j>=0 && a[j]>key)
12          {
13              a[j+1] = a[j];
14              --j;
15          }
16          a[j+1] = key;
17      }
18  }
```

Сортировка слиянием

- Во всех случаях время работы $\Theta(n \lg n)$
- Затраты памяти: $\Theta(n)$ вспомогательных

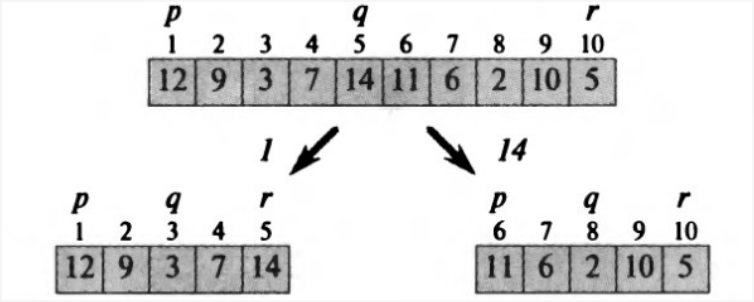
Алгоритм разработан
Джоном фон Нейманом
в 1945 году

Парадигма «разделяй и властвуй»

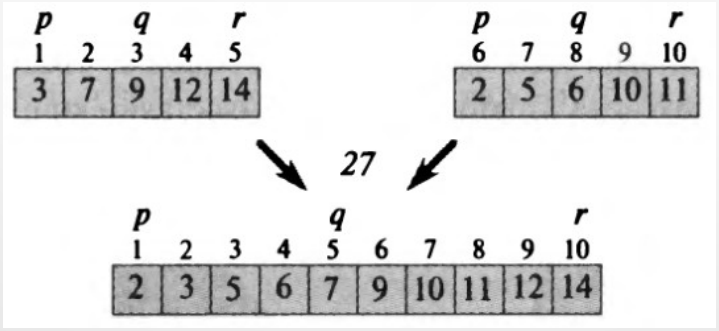
- **Разделение.** Задача разбивается на несколько подзадач, меньшего размера
- **Властвование.** Рекурсивно решаются подзадачи. Если они достаточно малы, они решаются как базовый случай.
- **Объединение.** Решения подзадач объединяются в решение исходной задачи.

Сортировка слиянием

- **Разделение.** Делим массив на две части q=[(p+r)/2]

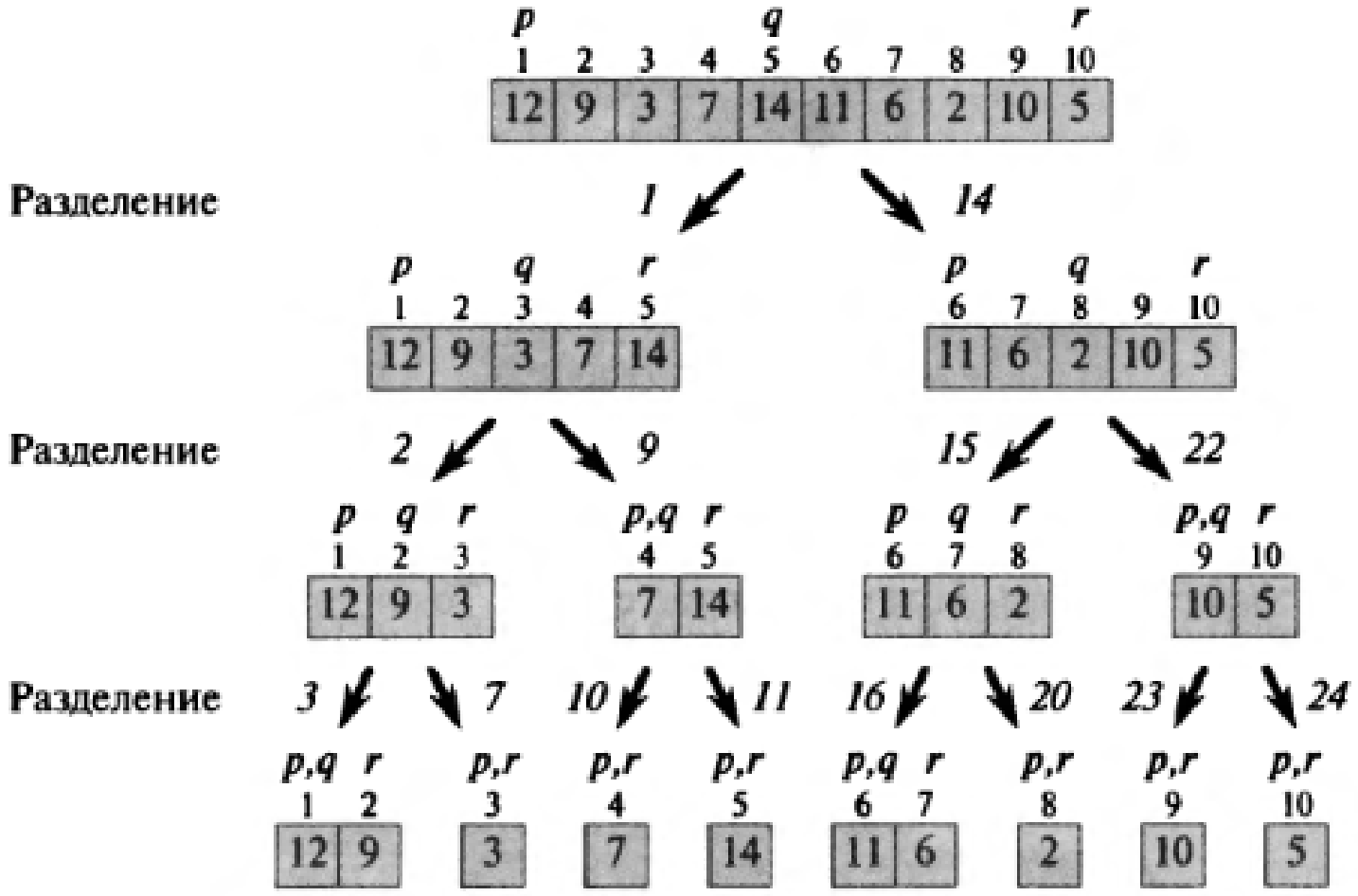


- **Властвование.** Рекурсивно сортируем книги в левой (от p до q) и правой (от q+1 до r) частях массива.
- **Объединение.** Объединение отсортированных книг в промежутках слотов от p до q и от q+1 до r так, чтобы книги в промежутке от p-го до r-го слотов были отсортированы.



Количество разделений

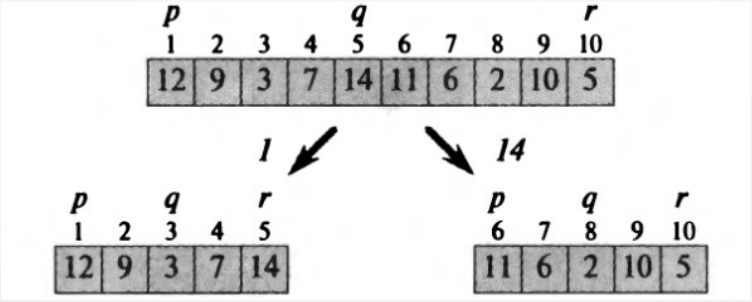
- Количество разделений O(lg n)

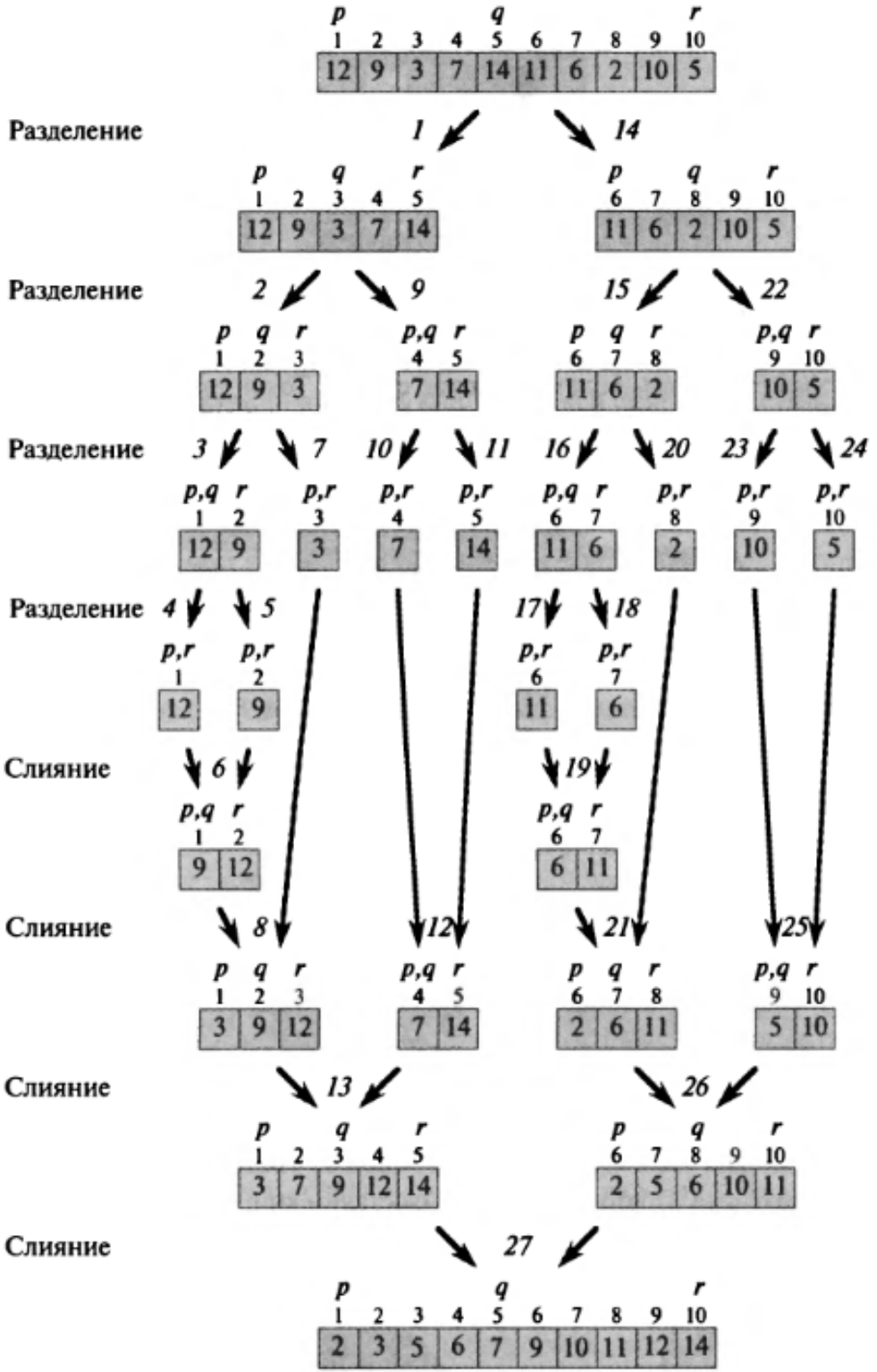


Разделение на C++

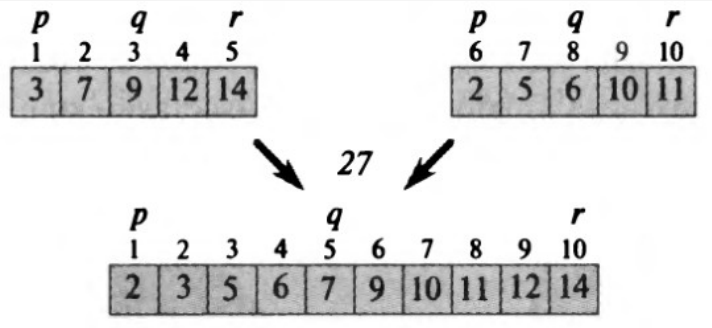
```
32  void merge_sort(int* a, int p, int r)
33  {
34      if (p>=r)
35          return;
36      int q = (p+r)/2;
37      merge_sort(a, p, q);
38      merge_sort(a, q+1, r);
39      merge(a, p, q, r);
40  }
```

Базовый случай — разделять дальше некуда




p q r
1 2 3 4 5 6 7 8 9 10
12 9 3 7 14 11 6 2 10 5
Разделение
1
14
p q r
1 2 3 4 5
12 9 3 7 14
p q r
6 7 8 9 10
11 6 2 10 5
Разделение
2
9
15
22
p q r
1 2 3
12 9 3
p,q r
4 5
7 14
p q r
6 7 8
11 6 2
p,q r
9 10
10 5
Разделение
3
7
10
11
16
20
23
24
p,q r
1 2
12 9
p,r
3
p,r
4
p,r
5
p,q r
6 7
11 6
p,r
8
p,r
9
p,r
10
2 10 5
Разделение
4
5
17
18
p,r
1
12
p,r
2
9
p,r
6
11
p,r
7
6
Слияние
6
19
p,q r
1 2
9 12
p,q r
6 7
6 11
Слияние
8
12
21
25
p q r
1 2 3
3 9 12
p,q r
4 5
7 14
p q r
6 7 8
2 6 11
p,q r
9 10
5 10
Слияние
13
26
p q r
1 2 3 4 5
3 7 9 12 14
p q r
6 7 8 9 10
2 5 6 10 11
Слияние
27
p q r
1 2 3 4 5 6 7 8 9 10
2 3 5 6 7 9 10 11 12 14

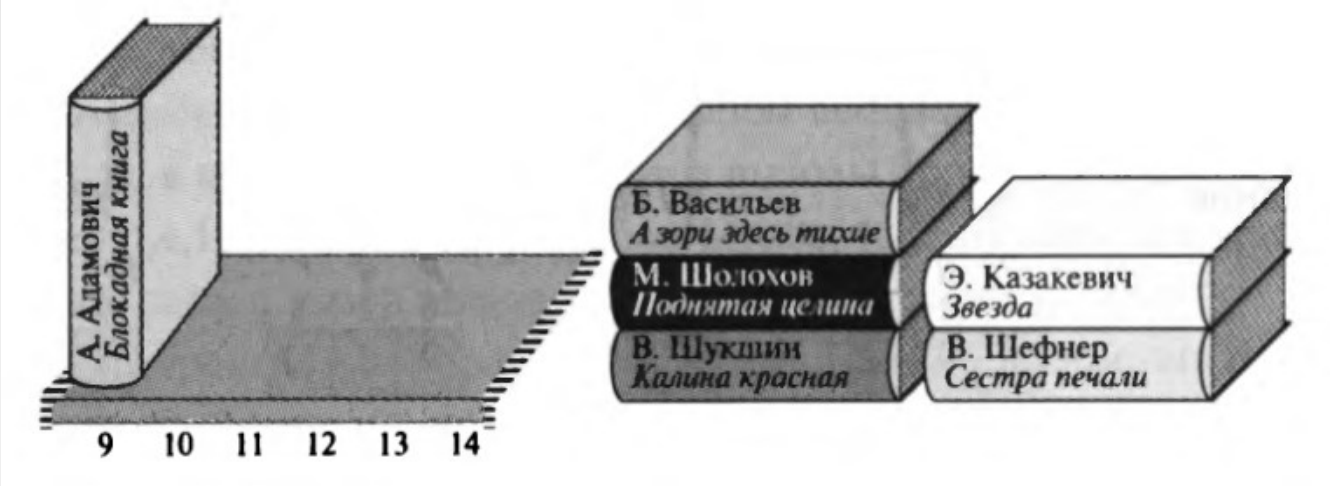
Объединение

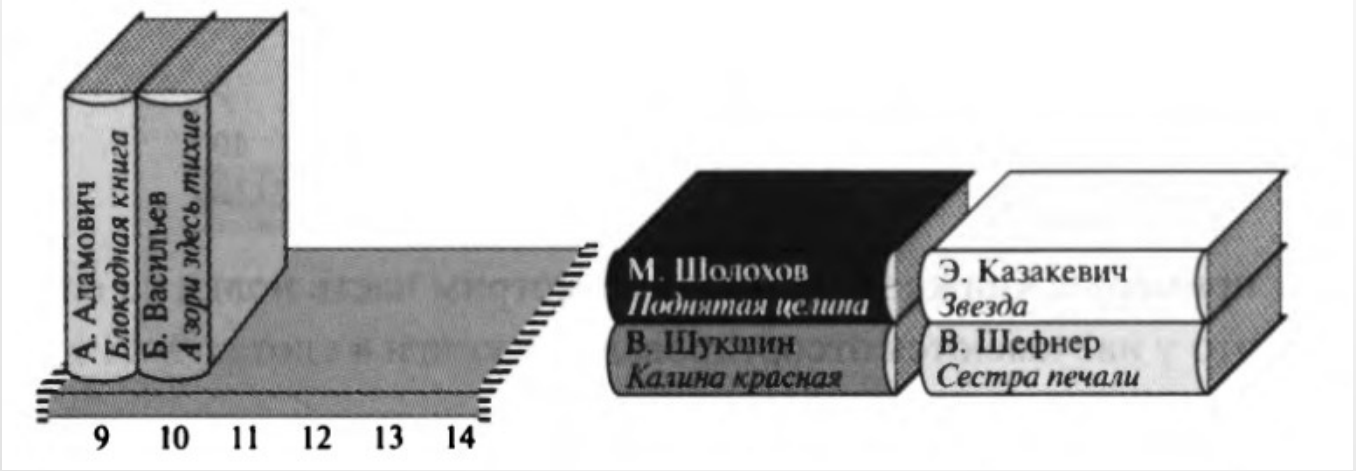


- Объединение — ключевой момент сортировки слиянием
- Нужно объединить два отсортированных массива так, чтобы снова получился отсортированный массив
- Лучшее теоретическое время объединения: Θ(n)

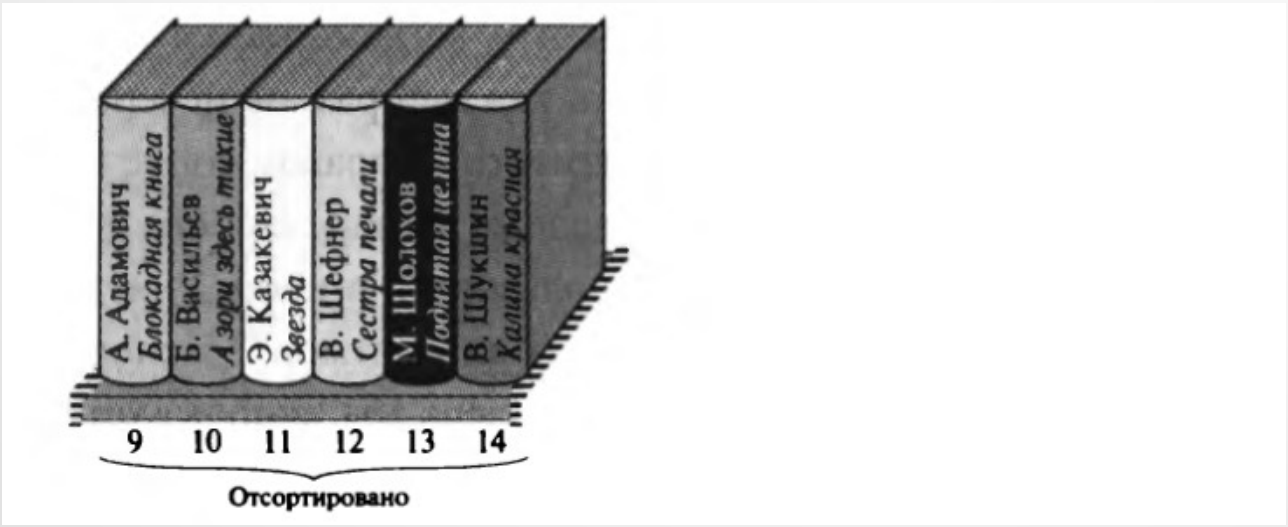
Объединение

Объединение

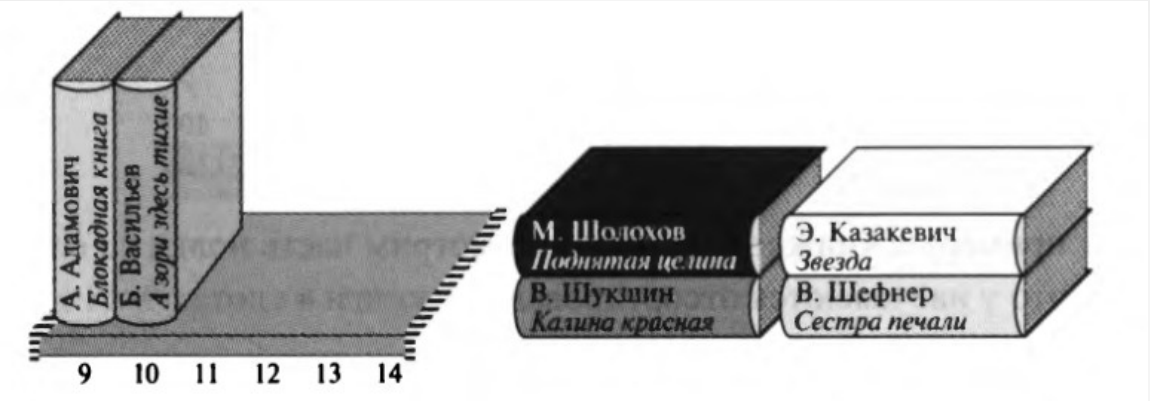




Объединение



Эффективность объединения



- Каждую книгу мы перемещаем ровно 2 раза: снимая её с книжной полки и вновь возвращая на полку
- Когда мы решаем, какая книга должны быть поставлена на полку, нам нужно сравнить только две книги

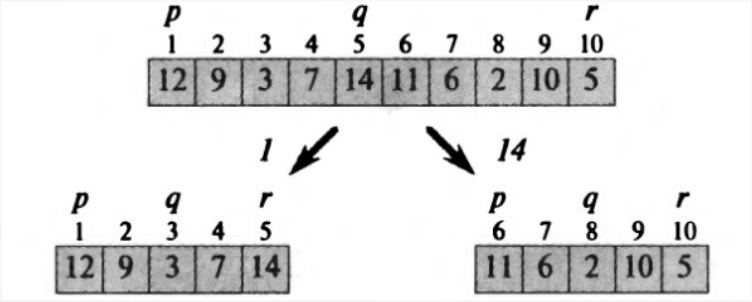
Количество перемещений: 2n
Количество сравнений: n

Копирование элементов при объединении

- Длины левого и правого массивов:

  n1=q-p+1

  n2=r-q



Копирование в массив b

```
11      for (int i=0; i<n1; ++i)
12          b[i] = a[p+i];
```

Копирование в массив c

```
13      for (int i=0; i<n2; ++i)
14          c[i] = a[q+1+i];
```

Объединение на C++

```
5  void merge(int* a, int p, int q, int r)
6  {
7      int n1 = q-p+1;
8      int n2 = r-q;
9      int* b = new int[n1+1];
10     int* c = new int[n2+1];
11     for (int i=0; i<n1; ++i)
12         b[i] = a[p+i];
13     for (int i=0; i<n2; ++i)
14         c[i] = a[q+1+i];
15     b[n1] = c[n2] = INT32_MAX;//Бесконечность в конце b и c
16     int i = 0;
17     int j = 0;
18     for (int k=p; k<=r; ++k)
19     {
20         if (b[i]<=c[j])
21         {
22             a[k] = b[i];
23             ++i;
24         } else
25         {
26             a[k] = c[j];
27             ++j;
28         }
29     }
30 }
```

n1 и n2 — длины массивов b и c

Выделение памяти под b и c

Копирование в массив b

Копирование в массив c

Ставим индексы в начало

Берём элемент из массива b
или из массива c
(какой больше)
и ставим в конец массива a.
Сдвигаем соответствующий
индекс.