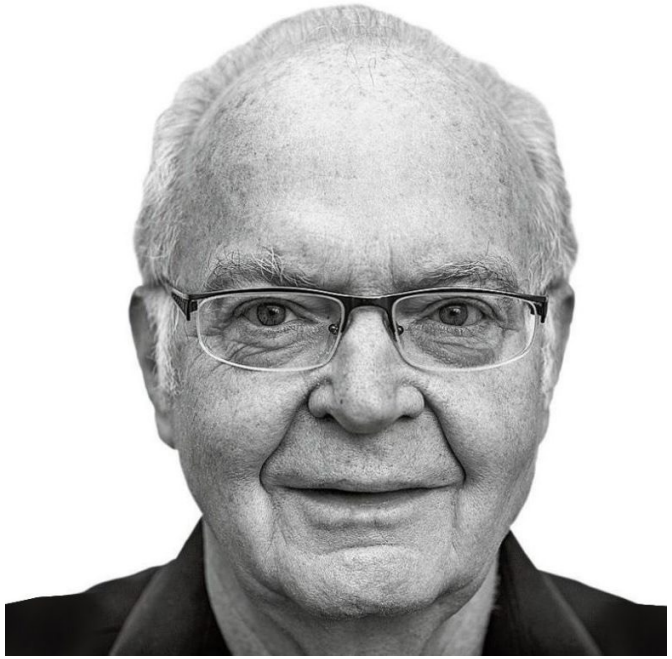
Итерационные сортировки

Горденко Мария Константиновна

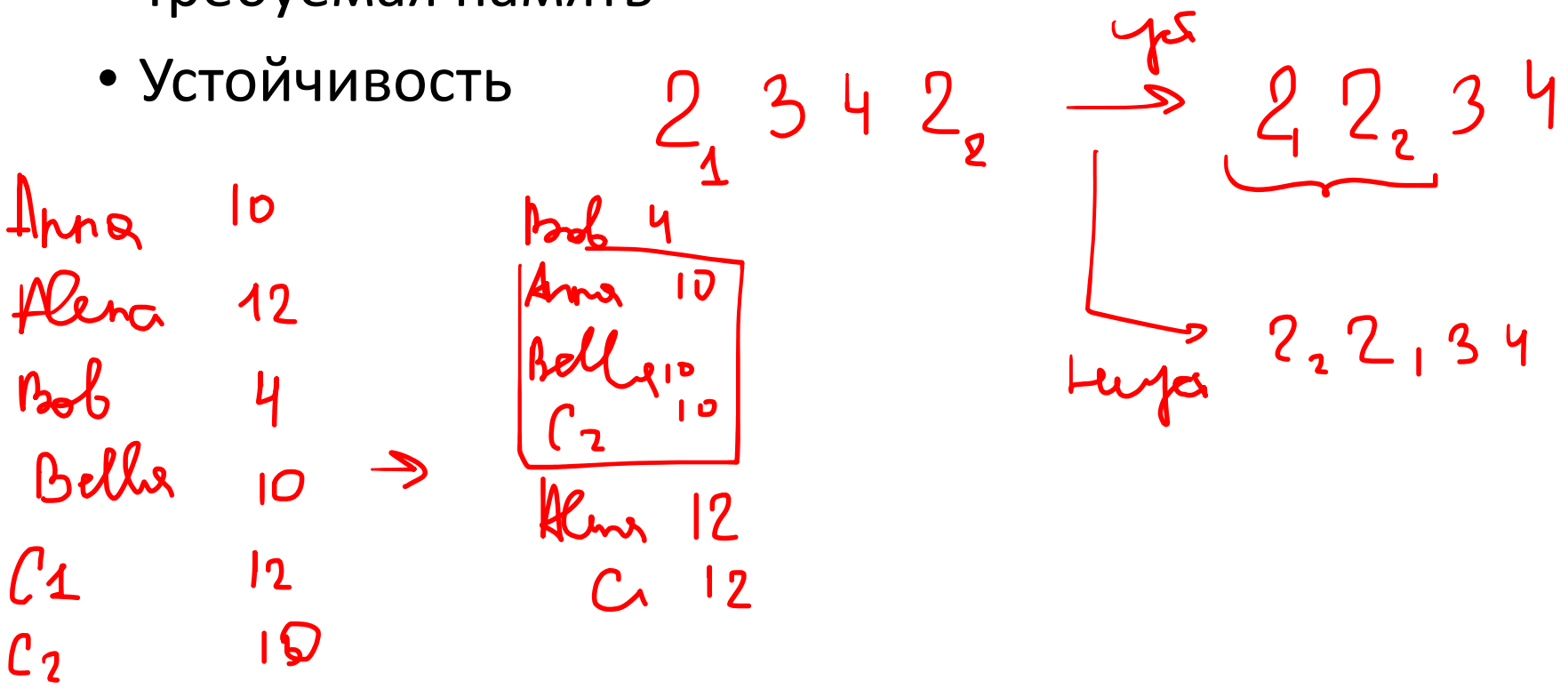
Задача сортировки

- Сортировка – это упорядочивание набора однотипных данных по возрастанию или убыванию.



Критерии оценки алгоритмов сортировки

- Время сортировки
- Требуемая память
- Устойчивость



Сфера применения

- Внутренние сортировки
- Внешние сортировки

Стратегии сортировки

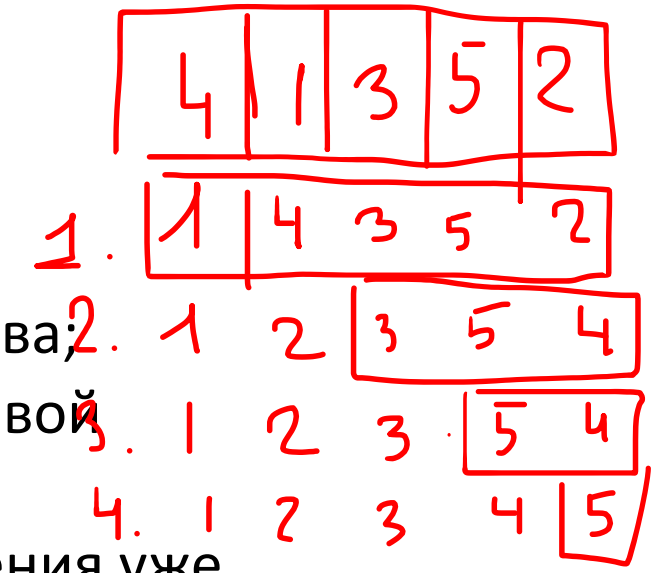
- Стратегия выборки
- Стратегия включения
- Стратегия распределения
- Стратегия слияния

Основные идеи итерационных сортировок

- Обмен
- Выбор
- Вставка
- Подсчет

Сортировка выбором (Selection sort)

- Сортировка выбором является одним из простейших алгоритмов сортировки
- Может быть как устойчивой, так и не устойчивой
- Шаги алгоритма:
  - находим минимальное значение в текущей части массива;
  - производим обмен этого значения со значением на первой неотсортированной позиции;
  - далее сортируем хвост массива, исключив из рассмотрения уже отсортированные элементы



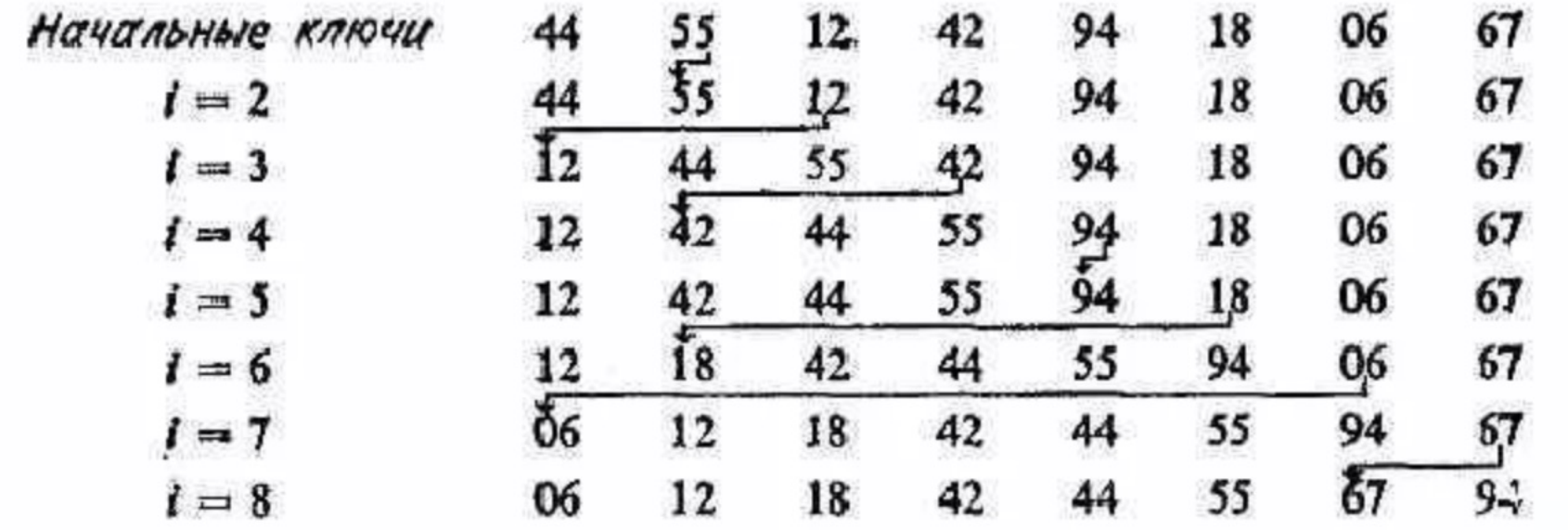
Сортировка выбором (Selection sort)

```
1. function selectionSort(T[n] a):
2.    for i = 0 to n - 2
3.      min = i
4.      for j = i + 1 to n - 1
5.        if a[j] < a[min]
6.          min = j
7.      swap(a[i], a[min])
```

- Асимптотическая сложность - $O(n^2)$
- Время работы в худшем, лучшем и среднем случае - $O(n^2)$
- Дополнительная память - $O(1)$

Сортировка вставками (Insertion sort)

- Сортировка вставками — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов



44 55 12 42 94 18 06 67

0: 44

1: 44 55

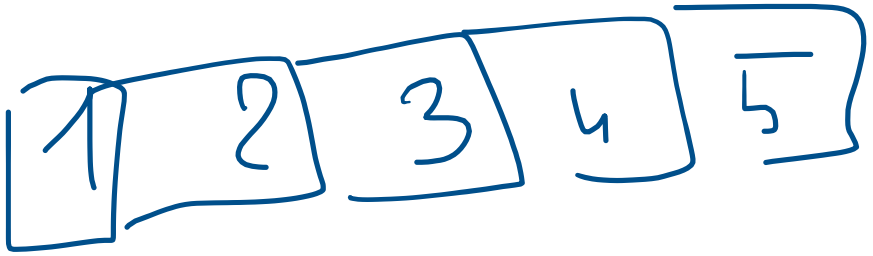
2: 12 44 55

3: 12 42 44 55

4: 12 42 44 55 94

5: 12 18 42 44 55 94

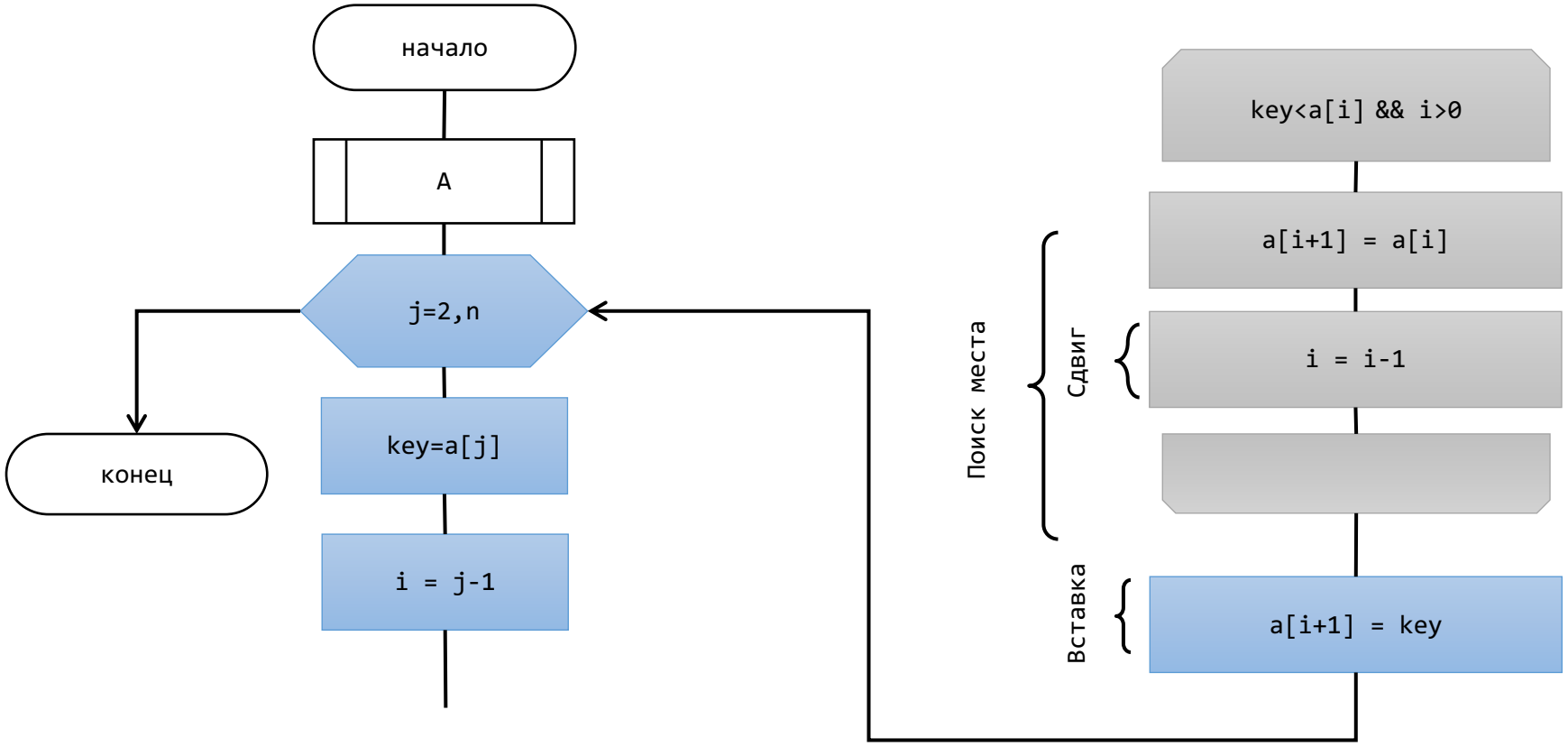
$O(n^2)$



$1 = n$

$n = n^2$

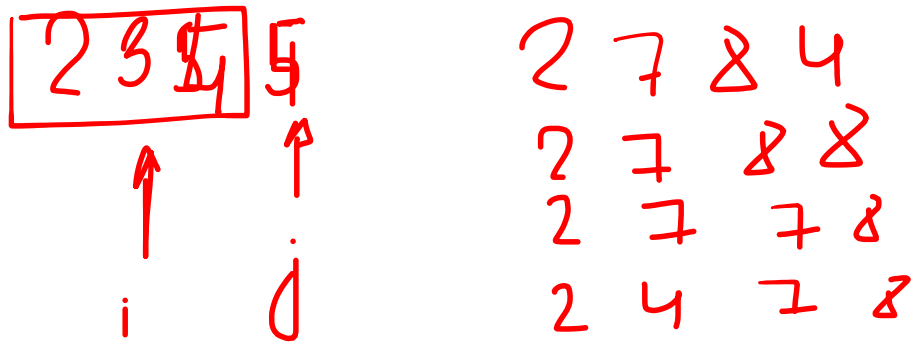
Сортировка вставками (Insertion sort)



Сортировка вставками (Insertion sort)

- Асимптотическая сложность - $O(n^2)$
- Время работы в худшем и среднем случае - $O(n^2)$
- Время работы в лучшем случае - $O(n)$
- Дополнительная память - $O(1)$

```
1. for j = 2 to A.length do
2.     key = A[j]
3.     i = j-1
4.     while (i > 0 and A[i] > key) do
5.         A[i + 1] = A[i]
6.         i = i - 1
7.     end while
8.     A[i+1] = key
9. end for
```



Сортировка бинарными вставками

- Мы можем использовать бинарный поиск, чтобы уменьшить количество сравнений в обычной сортировке вставкой . Бинарная сортировка вставок использует бинарный поиск, чтобы найти правильное место для вставки выбранного элемента на каждой итерации.
- При обычной сортировке вставкой в худшем случае требуется $O\ (n)$ сравнений (на n-й итерации). Мы можем уменьшить его до $O\ (\log n)$, используя бинарный поиск .

Сортировка бинарными вставками

- Асимптотическая сложность - $O(n^2)$
- Время работы в худшем и среднем случае - $O(n^2)$
- Время работы в лучшем случае - $O(n)$
- Дополнительная память - $O(1)$

```
1. function insertionSort(a):
2.   for i = 1 to n - 1
3.     j = i - 1
4.     k = binSearch(a, a[i], 0, j)
5.     for m = j downto k
6.       swap(a[m], a[m+1])
```

Сортировка вставками. Достоинства и недостатки

- эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказаться лучшим;
- эффективен на наборах данных, которые уже частично отсортированы;
- это устойчивый алгоритм сортировки (не меняет порядок элементов, которые уже отсортированы);
- может сортировать список по мере его получения;
- не требует временной памяти, даже под стек.
- **Минусом** же является высокая сложность алгоритма: $O(n^2)$.

Сортировка пузырьком (bubble sort)

- Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма).

Сортировка пузырьком

- Асимптотическая сложность - $O(n^2)$
- Время работы в худшем и среднем случае - $O(n^2)$
- Время работы в лучшем случае - $O(n)$
- Дополнительная память - $O(1)$

```
1. function bubbleSort(a):
2.   for i = 0 to n - 2
3.     for j = 0 to n – i - 2
4.       if a[j] > a[j + 1]
5.         swap(a[j], a[j + 1])
```

Пример

i	*j*		*1*	*2*	*3*	*4*	*5*	*6*
1	1		4	5	9	1	3	6
	2		4	5	9	1	3	6
	3	обмен	4	5	9	1	3	6
	4	обмен	4	5	1	9	3	6
	5	обмен	4	5	1	3	9	6
2	1		4	5	1	3	6	**9**
	2	обмен	4	5	1	3	6	**9**
	3	обмен	4	1	5	3	6	**9**
	4		4	1	3	5	6	**9**
3	1	обмен	4	1	3	5	**6**	**9**
	2	обмен	1	4	3	5	**6**	**9**
	3		1	3	4	5	**6**	**9**
4	1		1	3	4	**5**	**6**	**9**
	2		1	3	4	**5**	**6**	**9**
5	1		1	3	**4**	**5**	**6**	**9**

Оптимизация (условие Айверсона 1)

- Если после выполнения внутреннего цикла не произошло ни одного обмена, то массив уже отсортирован, и продолжать что-то делать бессмысленно. Поэтому внутренний цикл можно выполнять не n−1 раз, а до тех пор, пока во внутреннем цикле происходят обмены.

```
1. function bubbleSort(a):
2.   i = 0
3.   t = true
4.   while t
5.     t = false
6.     for j = 0 to n - i - 2
7.       if a[j] > a[j + 1]
8.         swap(a[j], a[j + 1])
9.         t = true
10.    i = i + 1
```

Оптимизация (условие Айверсона 2)

- Запоминаем, в какой позиции t был последний обмен на предыдущей итерации внешнего цикла.
- Это – верхняя граница просмотра массива Bound на следующей итерации
- Если t = 0 после выполнения внутреннего цикла, значит, обменов не было, алгоритм заканчивает работу.
- Основная идея – **уменьшаем количество проходов внутреннего цикла**

Модификации

- Сортировка чет-нечет
- Сортировка расческой
- Сортировка перемешиванием (шейкерная сортировка)