Информатика (основы программирования)

Лекция 11.

Алгоритмы сортировки массивов

Автор: Бабалова И.Ф.

Доцент, каф.12

2023 г.

Задача сортировки (I)

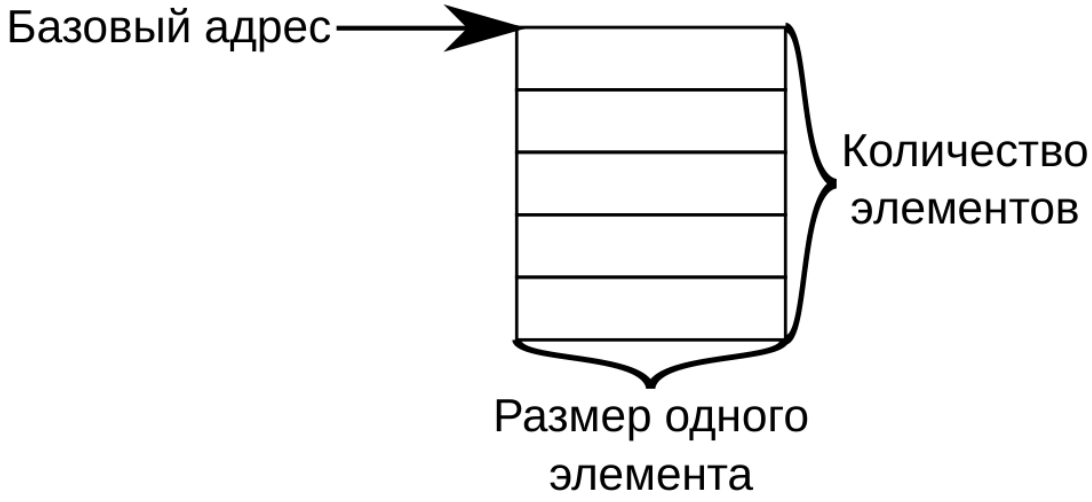
- Уровни представления данных:
  - Интуитивный
  - Логический (абстрактный)
  - Конкретный (физический)

- Задача сортировки решается на логическом уровне, а реализуется в виде программы на физическом уровне.

- Что мы сортируем? На логическом уровне – это массив, а на физическом – вектор.

Задача сортировки (II)

- Массив – множество значений (объектов любого типа), для которых упорядоченное множество целых чисел однозначно определяет позицию каждого элемента – это логическая структура.
- Вектор – конкретная структура данных:



Задача сортировки (III)

- Имеется последовательность из n ключей $k_1, k_2, \ldots, k_n$.
- **Требуется**: упорядочить ключи по не убыванию или не возрастанию.
- Это означает: найти перестановку ключей такую, что $k_1 \leq k_2 \leq \cdots \leq k_n$ или $k_1 \geq k_2 \geq \cdots \geq k_n$
- Элементами сортируемой последовательности могут быть любые типы данных. Обязательное условие — наличие **ключа**. Мы можем упорядочить эту последовательность несколькими способами, так как у нас есть одинаковые ключи. Но, как оказывается, не все способы одинаково эффективны и не всегда обеспечивают устойчивость сортировки.

Устойчивость сортировки

- Алгоритм сортировки устойчивый, если он сохраняет относительный порядок элементов с одинаковыми ключами.

Постараемся проанализировать методы сортировки:

- сравнением,
- с уменьшением числа перемещений,
- изменения направлений сортировки,
- увеличения расстояния между перемещаемыми элементами.

В данной лекции мы будем рассматривать методы сортировки только в оперативной памяти, без использования внешней памяти.

Основные правила сортировки (1)

- Сортировка – это упорядочивание информации по назначенному ключу.
- Массив сортируется на своем месте в памяти.
- Количество известных методов сортировки свыше 300.

- Оценка качества методов сортировки производится по двум показателям:
  - Количество сравнений элементов массива C (Compare) $\sim O(n^2)$
  - Количество перемещений элементов массива M (Move) $\sim O(n^2)$

Основные правила сортировки (2)

- Математический анализ методов сортировки показывает, что среднее количество перемещений элементов может быть $\sim n/3$.
- На этом выводе базируется многообразие методов сортировки.

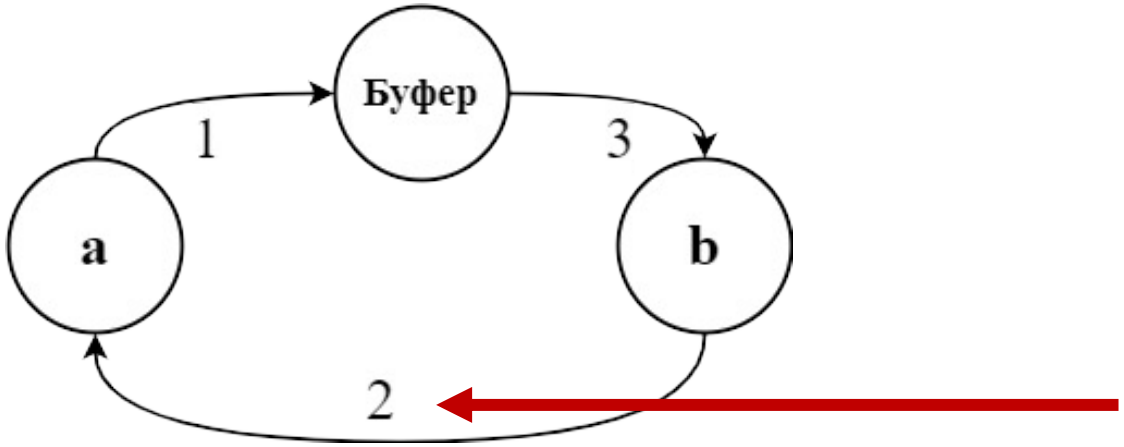
Наиболее часто используемые методы сортировки (1)

1. Простые обменные сортировки:
   A. Обменом.
   B. Вставками.
   C. Выбором.
2. Сортировка в разных направлениях (Шейкер).
3. Быстрая сортировка (по Хоару).
4. Сортировка с убывающим приращением (Шелла).
5. Сортировка слиянием.
6. Поразрядная сортировка.
7. Сортировка пирамидальная.

Наиболее часто используемые методы сортировки (2)

- Рассмотрим группу обменных сортировок:

- Основа всех сортировок – это обмен значениями через вспомогательную, буферную переменную.



Последовательность перемещений

Операция сравнения в задачах сортировки

- Очевидно, что для исполнения алгоритма сортировки необходимо определить операцию сравнения ключей:

$$\boldsymbol{a < b}$$

- Полагается, что $\boldsymbol{not\ (a < b) \wedge not\ (b < a) \rightarrow a = b}$ – это необходимое условие для соблюдения закона трихотомии: для любых **a**, **b** либо $\mathbf{a < b}$, либо $\mathbf{a = b}$, либо $\mathbf{a > b}$.

Понятие инверсии

- **Инверсия — пара ключей с нарушенным порядком следования.** Перестановка соседних элементов, расположенных в ненадлежащем порядке, уменьшает количество инверсий ровно на единицу.
- Количество инверсий в любом множестве конечно, в отсортированном же множестве оно равно нулю.
- Следовательно, количество обменов для сортировки конечно и не превосходит числа инверсий.
- Простейший алгоритм **обменной сортировки – сортировка «пузырьком»**. Он наиболее явно демонстрирует уменьшение количества инверсий при последовательной обработке ключей.

Сортировка обменом

- Лучший вариант исходной последовательности даёт оценку времени сортировки ~**O(n-1)**, а наихудший – **n*((n-1)/2)** или **~O(n^2)**
- После первого прохода хотя бы один элемент уже не будет перемещаться в следующем просмотре последовательности

Номера проходов						
	1	**2**	**3**	**4**	**5**	**6**
7	4	2	2	2	2	1
4	2	4	4	3	1	2
2	7	6	3	1	3	3
10	6	3	1	4	4	4
6	3	1	6	6	6	6
3	1	7	7	7	7	7
1	10	10	10	10	10	10

- Инвариант цикла для сортировки можно сформулировать так: после i-ого прохода не менее i элементов справа отсортированы.
- Усовершенствование метода возможно в связи с тем, что в результате каждого прохода последовательности самый большой элемент оказывается в конце последовательности (при упорядочивании по возрастанию).
- Так как количество сравнений конечно, то алгоритм обязательно завершается. Временная оценка такого метода соответствует **t ≈ O(n^2)**

Сортировка выбором (I)

- **Идея метода.** После первого просмотра последовательности наименьший элемент становится первым слева, а сравниваемый ключ становится на его место. Остальные элементы местами не меняются.

- $C = \frac{1}{2}(n^2-n)$; $M_{min} = 3(n-1)$; $M_{max} = trunc(n^2/4)+3(n-1)$.

- Среднее число перестановок определяется через эйлерову константу $Y=0.577216...$ следующим образом:

$M_{ср} = n \cdot (\ln(n)+Y)$ – это уже не $O(n^2)$

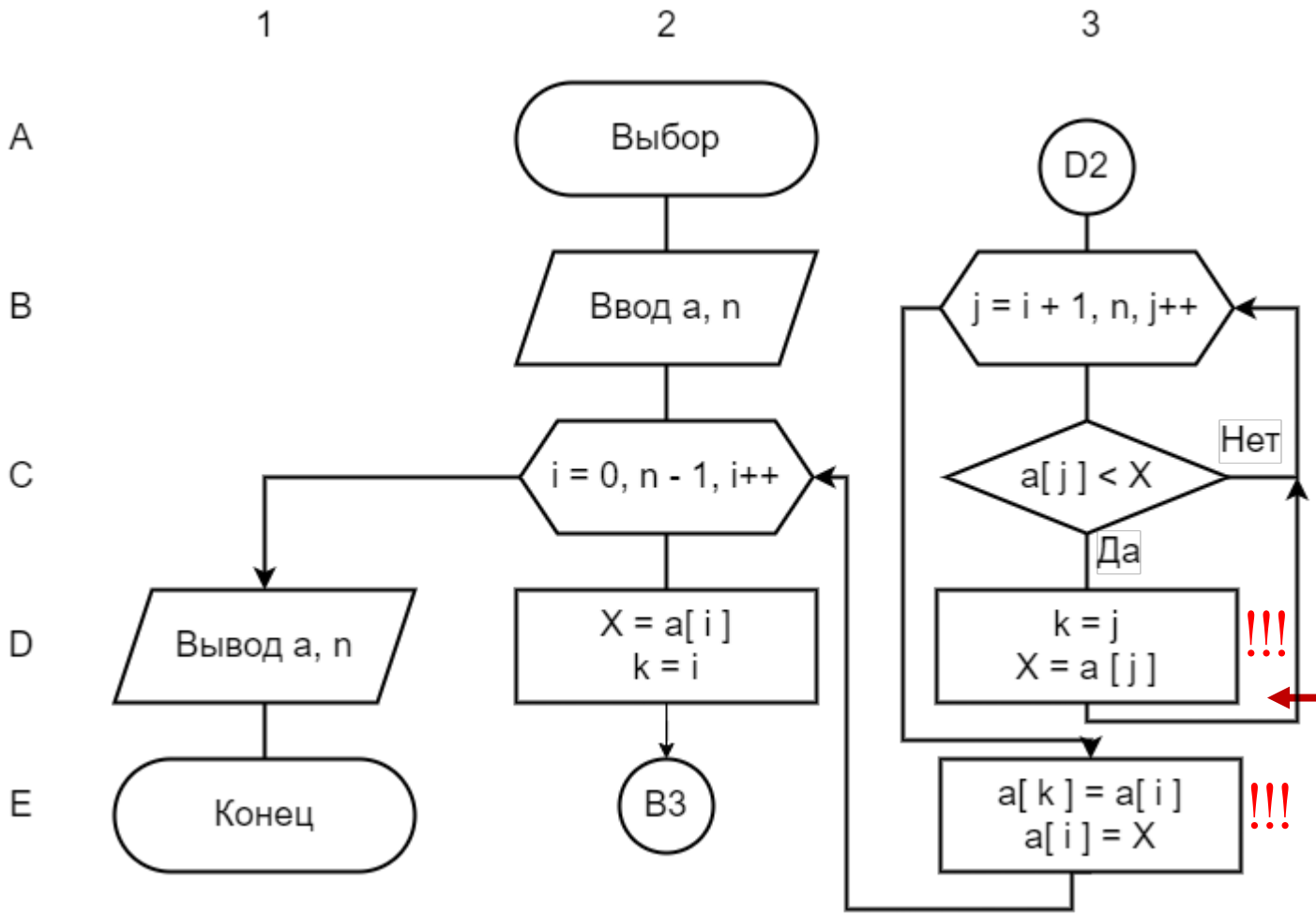
- Алгоритм позволяет уменьшить количество обменов элементов массива.

Сортировка выбором (II)

Номера проходов					
0	1	2	3	4	5
21	-1	-1	-1	-1	-1
18	18	18	18	18	18
31	31	21	21	21	21
75	75	75	28	28	28
44	44	44	44	31	31
28	28	28	75	75	33
-1	21	31	31	44	44
33	33	33	33	33	75

- Устойчивость сортировки.
- Количество перестановок – 5, а элементов в массиве – 8.
- При неизвестном порядке значений эта сортировка быстрее, чем сортировка простым обменом. Оценка времени сортировки в наихудшем случае $\mathbf{t \approx O(n^2)}$.
- Ищется наименьший элемент из оставшихся.

Алгоритм сортировки выбором



Цикл поиска наименьшего элемента

Сортировка вставками

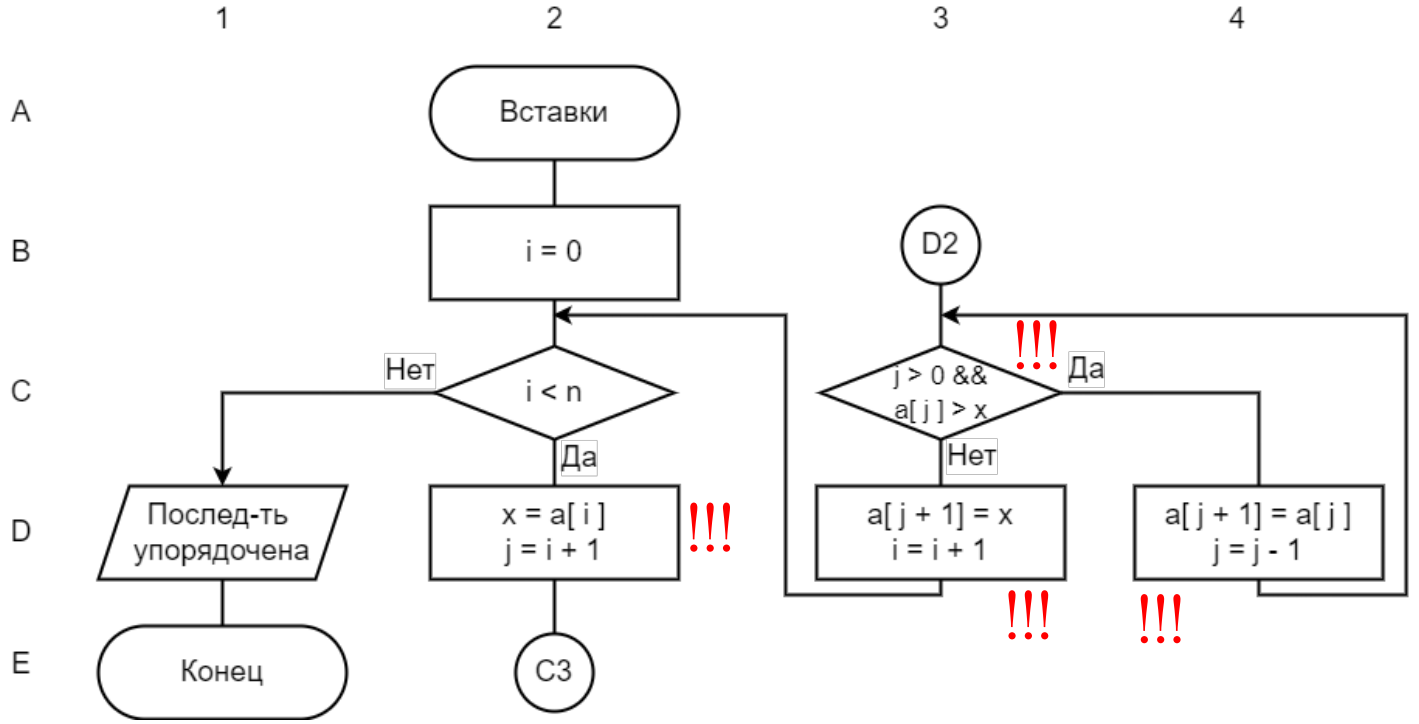
- Предполагаем, что $a_0, a_1, a_2, \dots, a_{n-1}$ не упорядочены.
- Назначается элемент массива a_0 буферным элементом.
- Элемент a_0 сравнивается со всеми имеющимися элементами до тех пор, пока не будет обнаружено между какими элементами он должен быть помещен: a_j и a_{j-1}.
- Все элементы, начиная с j-ого, сдвигаются вправо и вместо элемента j−1-го вставляется найденный элемент.
- Чтобы обеспечить контроль за длиной последовательности и не уйти за индекс, равный 1, вводится дополнительный контроль за длиной последовательности.
- Следовательно, окончание процесса сортировки обеспечивается двумя условиями:

1. Достигается барьер $a_0 < a_0$.
2. Достигается конец последовательности.

Сортировка вставками

- Исходная послед-ть:
$7,12,15,4,3$ - $a_0, a_1, \dots, a_4$

i = 1: x=12, j = 0
$7 < 12$ – порядок не поменяется
i = 2: x=15, j = 1
$12 < 15$ – порядок не поменяется
i = 3: x=4, j = 0, $4 < 15$ – начинаем перемещение элементов влево, пока j не станет отрицательным.



Последовательность будет иметь вид – $4, 7, 12, 15, 3$
Аналогично перемещается последний элемент – $3, 4, 7, 12, 15$
Обратите внимание на то, что количество присваиваний существенно сокращается

Оценка сходимости сортировки вставками

- Предполагаем, что $\mathbf{a_0, a_1, a_2, \dots, a_{n-1}}$ не упорядочены
- Количество присваиваний значений: $\mathbf{M_{ср} = \frac{1}{4} \cdot (n^2 + 9n - 10)}$
- Количество сравнений: $\mathbf{C_{ср} = \frac{1}{4} \cdot (n^2 + n - 2)}$
- Сортировка устойчива, так как элементы с одинаковыми ключами не перемещаются
- Наименьшие времена для подобной сортировки будут тогда, когда последовательность будет частично упорядоченной

$$\mathbf{C_{min} = n - 1}$$

$$\mathbf{M_{min} = 2(n - 1)}$$

Примеры записи функций для организации процесса сортировки

```
#define MAX 1000000
void swap(int *d1, int *d2) {
  int tmp = *d1;
 *d1 = *d2;    *d2 = tmp;
}
void array_sort(int *data, int len) {
 for (int i = 0; i < len - 1; ++i) {
    for (int j = 0; j < len - i - 1; ++j) {
       if (data[j] > data[j + 1]) {
          swap(&data[j], &data[j + 1]);
          }
      }
    }
}
```

```
void array_print(int *data, int len) {
   for (int i = 0; i < len; ++i)
 {
   printf(  " %d ", data[i]);
  }
 printf("\n");}
```

```
int *array_generate(int len, int max)
 {    int *data = calloc(len, sizeof(int));
      for (int i = 0; i < len; ++i)
{       data[i] = rand() % max;
 }    return data;
}
```