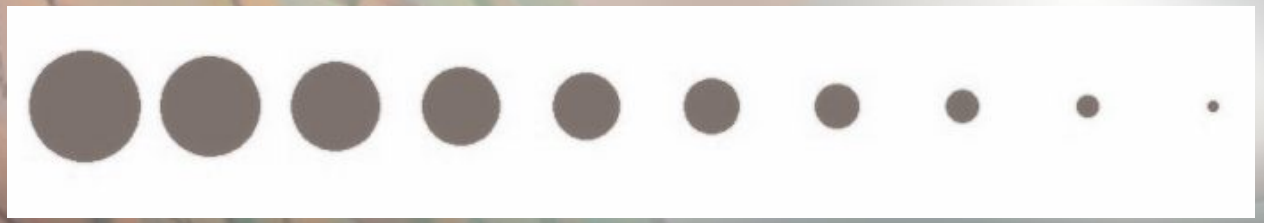
Структуры и алгоритмы обработки данных

Лекция 6

Алгоритмы сортировки на массивах



Сортировка объектов

Эта тема посвящена сугубо алгоритмической проблеме упорядочения данных

Сортировка *применяется во всех без исключения областях программирования, будь то базы данных или математические программы*

К примеру, входные данные подаются "вперемешку", а вашей программе удобнее обрабатывать упорядоченную последовательность

Сортировка **-** в информатике переупорядочение рассматриваемых объектов по некоторому признаку или системе признаков

Например, упорядочение слов по алфавиту называется ***лексикографической*** сортировкой

Алгоритм сортировки

- *АЛГОРИТМ ДЛЯ УПОРЯДОЧЕНИЯ НЕКОТОРОГО МНОЖЕСТВА ЭЛЕМЕНТОВ*

Обычно под алгоритмом сортировки подразумевают алгоритм упорядочивания множества элементов по возрастанию или убыванию

В случае наличия элементов с одинаковыми значениями, в упорядоченной последовательности они располагаются рядом друг за другом в любом порядке. Однако иногда бывает полезно сохранять первоначальный порядок элементов с одинаковыми значениями

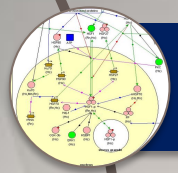
Часть данных используется в качестве ключа сортировки

Ключом сортировки называется атрибут , по значению которого определяется порядок элементов

При написании алгоритмов сортировок массивов следует учесть, что ключ полностью или частично совпадает с данными

Классификация алгоритмов сортировок

по устойчивости



по поведению

по использованию операций сравнения

по потребности в дополнительной памяти

по потребности в знаниях о структуре данных, выходящих за рамки операции сравнения, и др.

Классификация алгоритмов сортировок

по устойчивости

Устойчивая сортировка не меняет взаимного расположения равных элементов

Исходные данные

1	*a*	3	*q*	1	*b*	1	*c*	2	*z*

Пример работы устойчивой сортировки

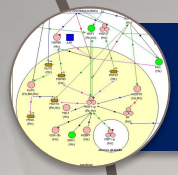
1	*a*	1	*b*	1	*c*	2	*z*	3	*q*

Пример работы неустойчивой сортировки

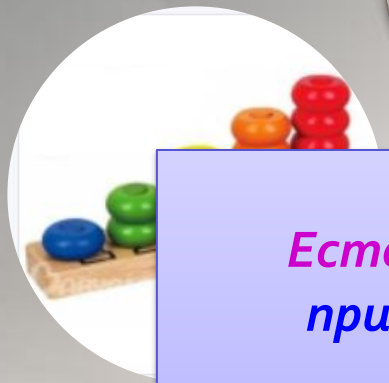
1	*c*	1	*b*	1	*a*	2	*z*	3	*q*

Классификация алгоритмов сортировок

по устойчивости



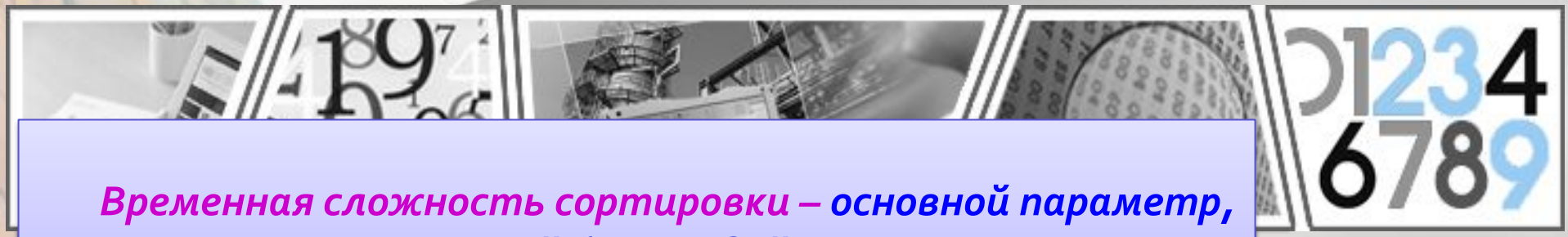
по поведению



Естественность поведения – *эффективность метода при обработке уже отсортированных, или частично отсортированных данных*

Алгоритм ведет себя естественно, если учитывает эту характеристику входной последовательности и работает лучше

Классификация алгоритмов сортировок



Временная сложность сортировки – основной параметр, характеризующий быстродействие алгоритма

- *по использованию операций сравнения*
- *по потребности в дополнительной памяти*
- *по потребности в знаниях о структуре данных, выходящих за рамки операции сравнения, и др.*

Алгоритмы внешней и внутренней сортировки

КЛАССИФИКАЦИЯ АЛГОРИТМОВ СОРТИРОВКИ ПО СФЕРЕ ПРИМЕНЕНИЯ

Алгоритмы **внутренней** сортировки оперируют сравнительно небольшими объемами данных. Они могут "видеть" любой элемент сортируемого множества

Алгоритмы **внешней** сортировки применяются тогда, когда количество элементов велико и нет возможности "разложить их на столе"
(в оперативной памяти)

Классификация алгоритмов сортировок

КЛАССИФИКАЦИЯ АЛГОРИТМОВ СОРТИРОВКИ ПО СФЕРЕ ПРИМЕНЕНИЯ

Сортировка

Внутренняя сортировка
или
сортировка массивов

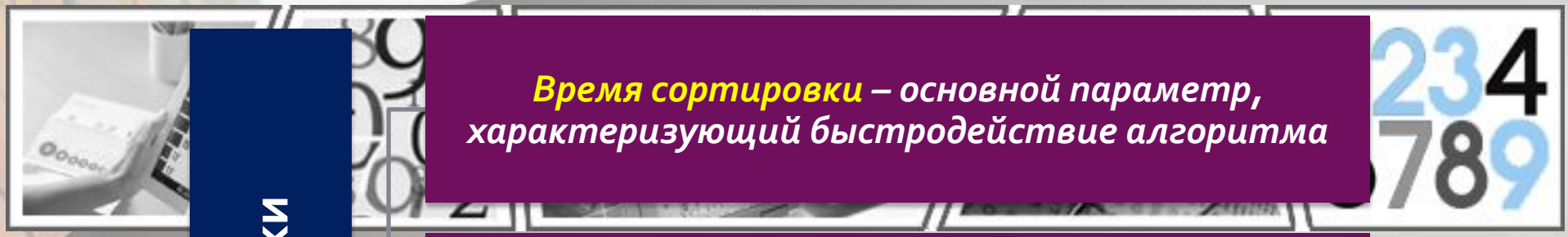
- все рассматриваемые объекты находятся в оперативной памяти, то есть в любой момент времени «видны», доступны любые элементы массива (имеется прямой доступ к сортируемым элементам)

Внешняя сортировка
или
сортировка файлов

- выполняется над объектами, находящимися во внешней памяти.

У файла «виден», доступен только один элемент, попавший в буфер файла (имеется последовательный доступ к сортируемым элементам)

Оценка алгоритмов сортировки



Алгоритм сортировки

Время сортировки *– основной параметр, характеризующий быстродействие алгоритма*

Память *– ряд алгоритмов сортировки требуют выделения дополнительной памяти под временное хранение данных*

Устойчивость *– сортировка не меняет взаимного расположения равных элементов*

Естественность поведения *– эффективность метода при обработке уже отсортированных, или частично отсортированных данных*

Этапы алгоритма сортировки

Алгоритм сортировки

- *сравнение, определяющее упорядоченность пары элементов*
- *перестановка, меняющая местами пару элементов*
- *сортирующий алгоритм, который осуществляет сравнение и перестановку элементов до тех пор, пока все элементы множества не будут упорядочены*

Внутренняя сортировка

Внутренняя сортировка

- выполняется «на том же самом месте», то есть без использования вспомогательных массивов

- ► сортируемые массивы могут иметь огромные размерности, сортируются сотни миллионов элементов
- ► эффективное использование оперативной памяти

Внутренняя сортировка

- **Методы сортировки**
  - **Сортировка в линейных структурах**
    - **Вставкой**
      - Простая вставка
      - Бинарная вставка
    - **Выбором**
      - Простой выбор
    - **Обменом**
      - Стандартный обмен
      - Метод Шелла
      - Метод Хоара
  - **Сортировка в нелинейных структурах**
    - Турнирная
    - Пирамидальная

Внутренняя сортировка

Методы внутренней сортировки

Прямые методы

- *вставкой (включением)*
- *выбором (выделением)*
- *обменом («пузырьковая»)*

Улучшенные методы

- *быстрая*
- *Шелла*

Оценка сложности сортировки

Постановка задачи сортировки в общем виде предполагает, что существуют только два типа действий с данными сортируемого типа:

- сравнение двух элементов (***x<y***)
- пересылка элемента (***x:=y***)

Поэтому удобная мера сложности алгоритма сортировки массива ***a[1..n]***:

- по времени – количество сравнений ***C(n)***
- количество пересылок ***M(n)***

Простые сортировки

К **простым внутренним сортировкам** относят методы, сложность которых пропорциональна квадрату размерности входных данных

Иными словами, при сортировке массива, состоящего из N компонент, такие алгоритмы будут выполнять C^*N^2 действий, где C - некоторая константа. Этот факт принято обозначать следующей символикой: $O(N^2)$

Сортировка

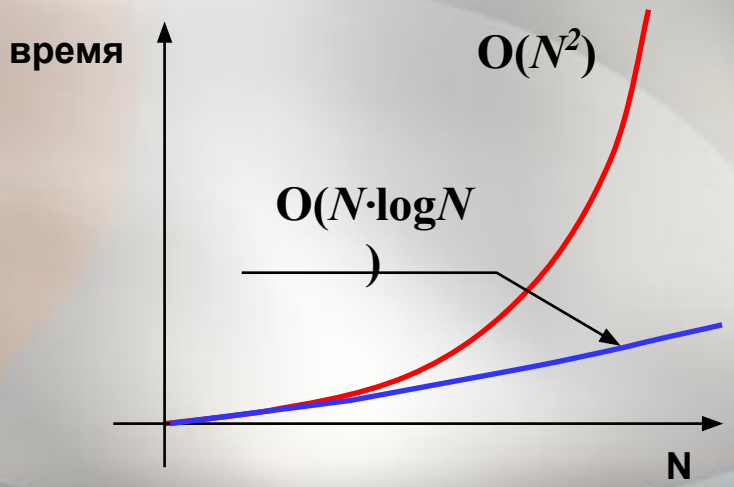
Алгоритмы:

- простые и понятные, но неэффективные для больших массивов
  - **метод пузырька**
  - **метод выбора**
  - **метод прямой вставки**

сложность $O(N^2)$

сложность $O(N \cdot \log N)$

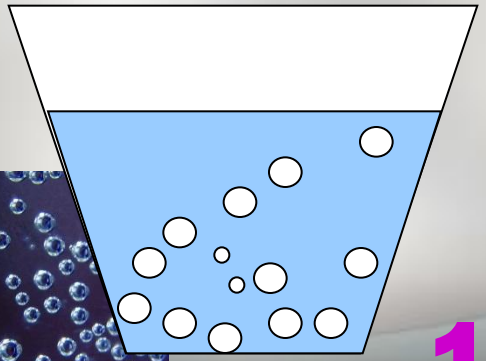
- сложные, но эффективные
  - **«быстрая сортировка» (*Quick Sort*)**
  - **сортировка «кучей» (*Heap Sort*)**
  - **сортировка слиянием**
  - **пирамидальная сортировка**



НЕ СУЩЕСТВУЕТ УНИВЕРСАЛЬНОГО, НАИБОЛЕЕ ЭФФЕКТИВНОГО СПОСОБА СОРТИРОВКИ

Метод пузырька (сортировка обменом)

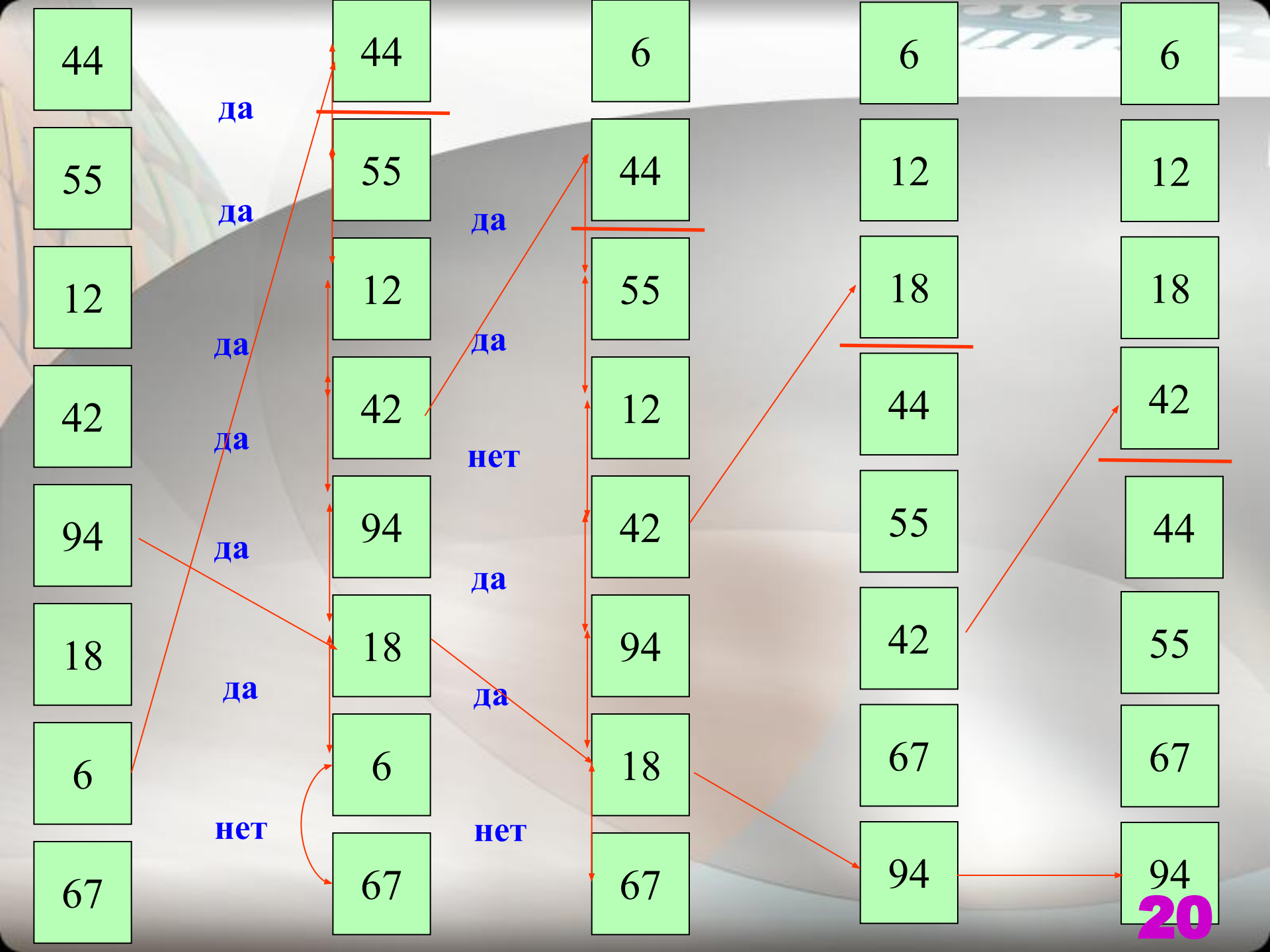
- Последовательно просматривается массив и сравнивается каждая пара элементов между собой
- При этом "неправильное" расположение элементов устраняется путем их перестановки
- Процесс просмотра и сравнения элементов повторяется до просмотра всего массива



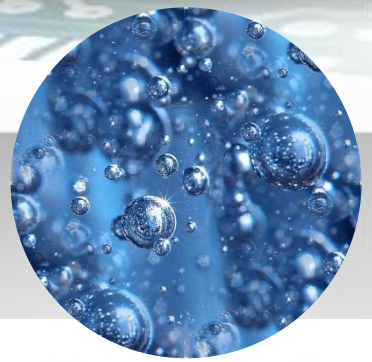
Метод пузырька / сортировка обменом

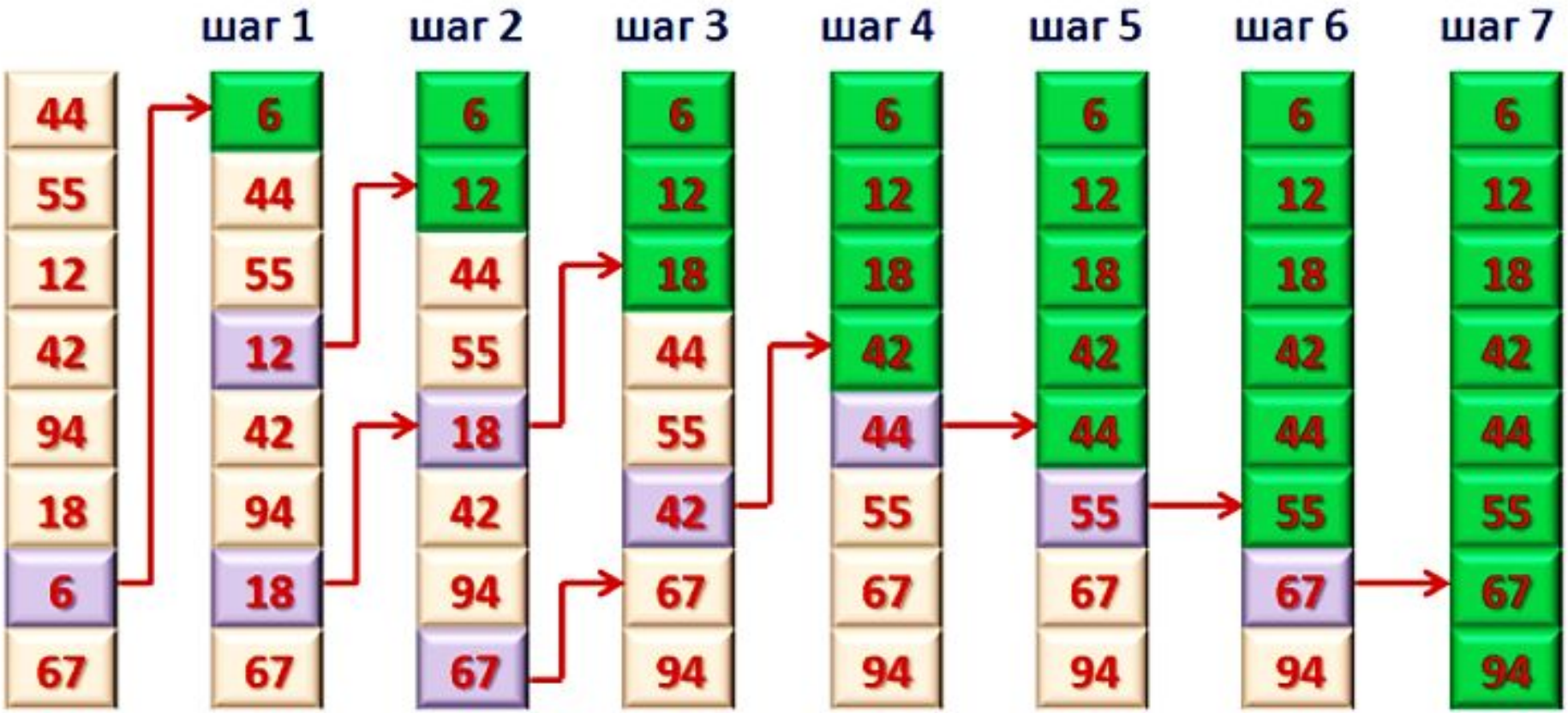
по возрастанию

Массив отсортирован по возрастанию

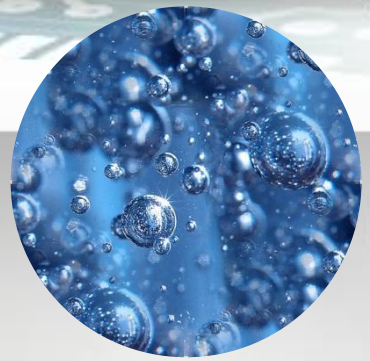

44
55
12
42
94
18
6
67
да
да
да
да
да
да
нет
44
55
12
42
94
18
6
67
да
да
да
нет
нет
6
44
55
12
42
94
18
67
6
12
18
44
55
42
67
94
6
12
18
42
44
55
67
94

Результаты работы алгоритма сортировки обменом



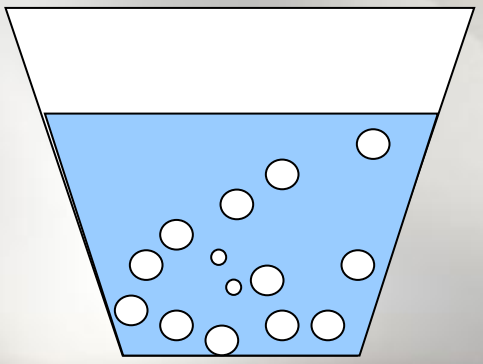


Сортировка обменом, метод пузырька

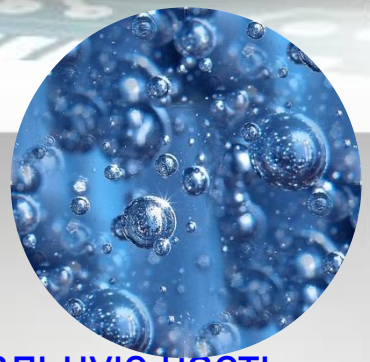


- Для осуществления сортировки нужно выполнить несколько шагов, несколько *проходов* по массиву
- На первом шаге на своем месте оказывается *самый маленький* элемент. На втором — следующий по величине и т.д.
- Вообще, на каждом шаге на «свое место» попадает, по крайней мере, один элемент массива
- **Для полного упорядочения нужно выполнить *N-1* шаг сортировки**

Так как при вертикальном расположении массива на каждом проходе в начальную часть массива, наверх «поднимаются», «всплывают» маленькие, «лёгкие» элементы, то обсуждаемый тип сортировки называют «пузырьковой», по аналогии с всплывающими к поверхности воды пузырьками воздуха

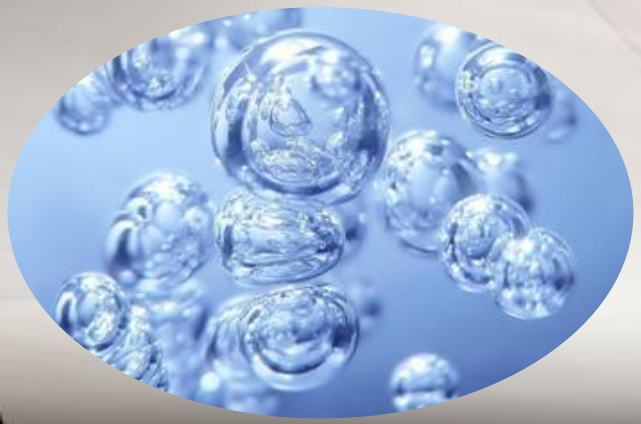


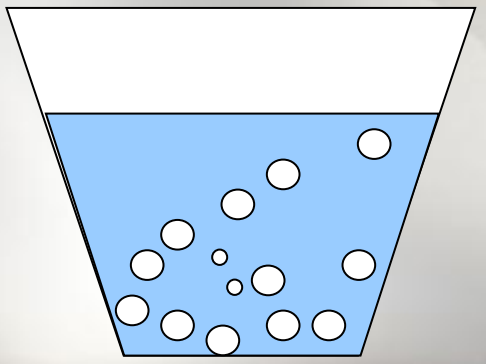
Сортировка обменом, метод пузырька



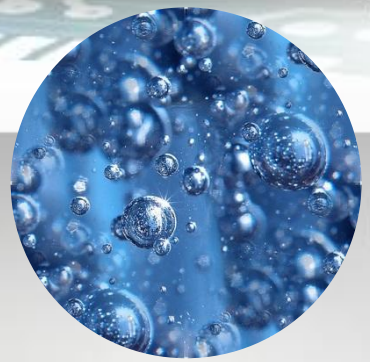
- Следует обратить внимание на то, что за один проход в начальную часть массива (в его уже упорядоченную часть) попадает самый маленький элемент из неупорядоченной части, а самый большой элемент перемещается в конец массива, «опускается вниз» *всего на одну* позицию

- В методе обменной сортировки *нужный элемент отыскивается* с помощью действий *в неупорядоченной части* и найденный элемент *добавляется в конец уже упорядоченной части*

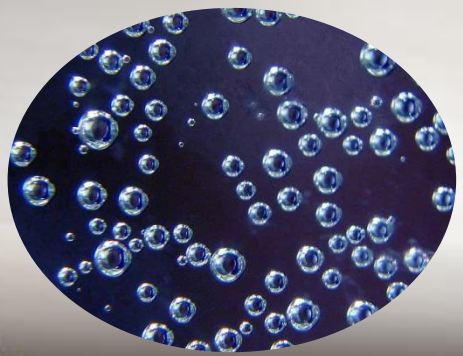


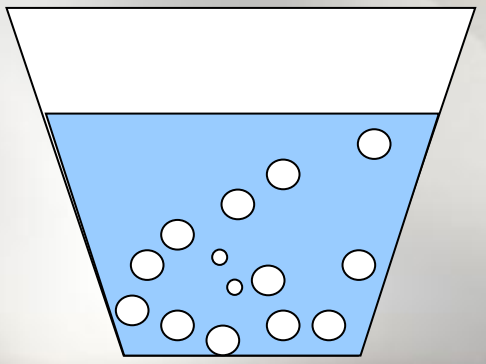


Сортировка обменом, метод пузырька

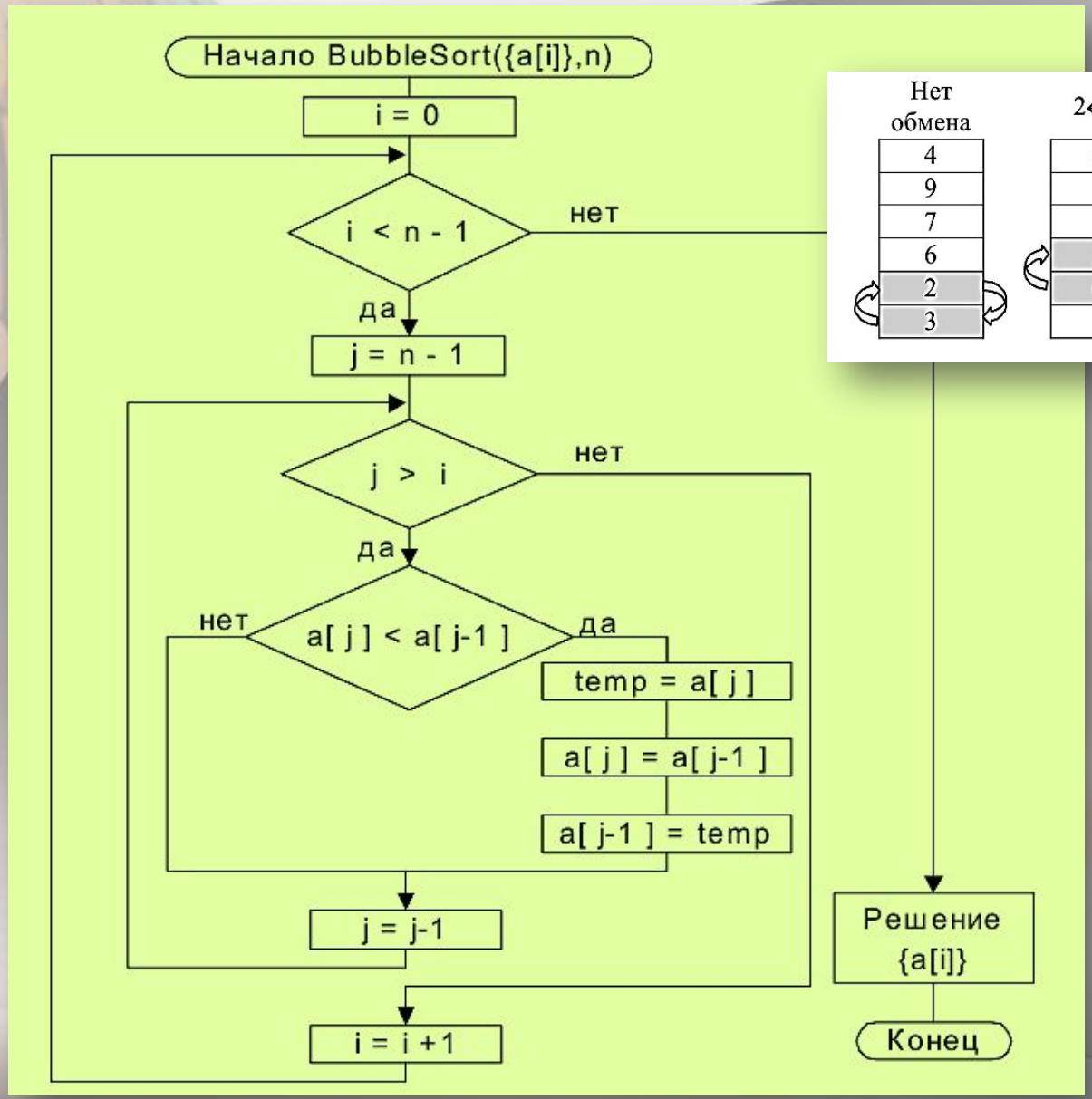


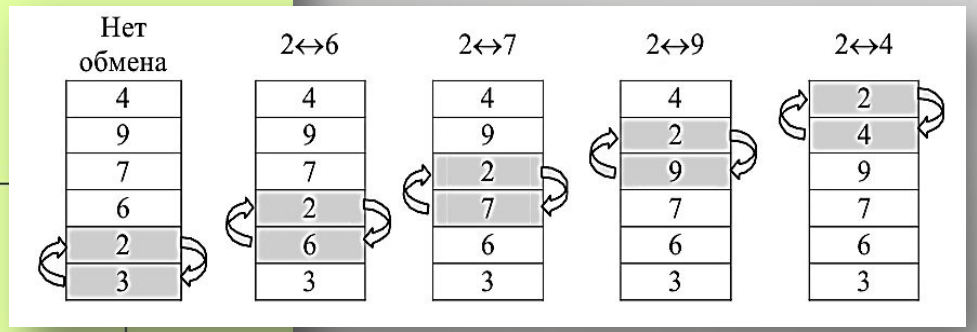
- На каждом шаге нужно организовать сравнение двух соседних элементов
- Так как сравнения начинают с последней пары элементов, то сначала сравниваются N-1-й и N-й элементы, затем N-2-й и N-1-й элементы и т.д., последними *на первом проходе* сравниваются 2 и 1 элементы
- Пару сравниваемых элементов можно задавать меньшим или большим номером из номеров, образующих пару элементов
- Если у сравниваемых элементов x[j] и x[j-1] обнаруживается «неправильный» порядок, то они стандартным способом меняются местами





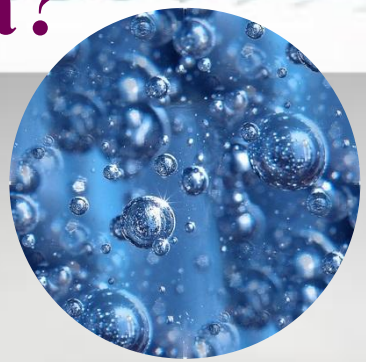
Метод пузырька, блок-схема



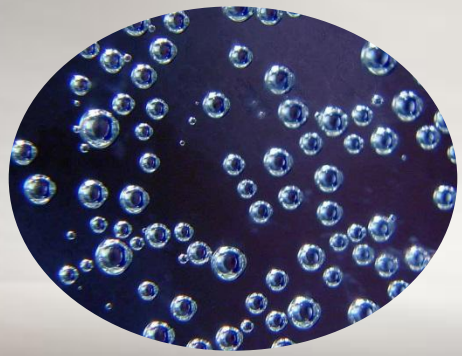


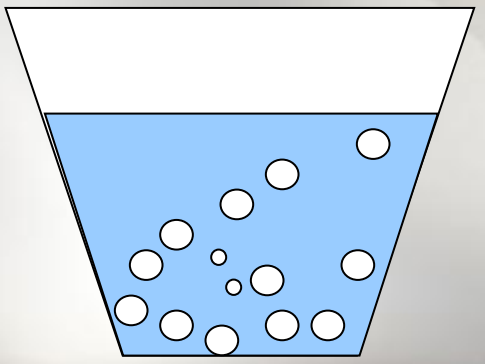
- *Расположим массив сверху вниз, от нулевого элемента – к последнему*
- *Шаг сортировки состоит в проходе снизу вверх по массиву*
- *По пути просматриваются пары соседних элементов*
- *Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами*

Метод пузырька? Другая схема?



5	2	1	3	9	0	4	6	8	7

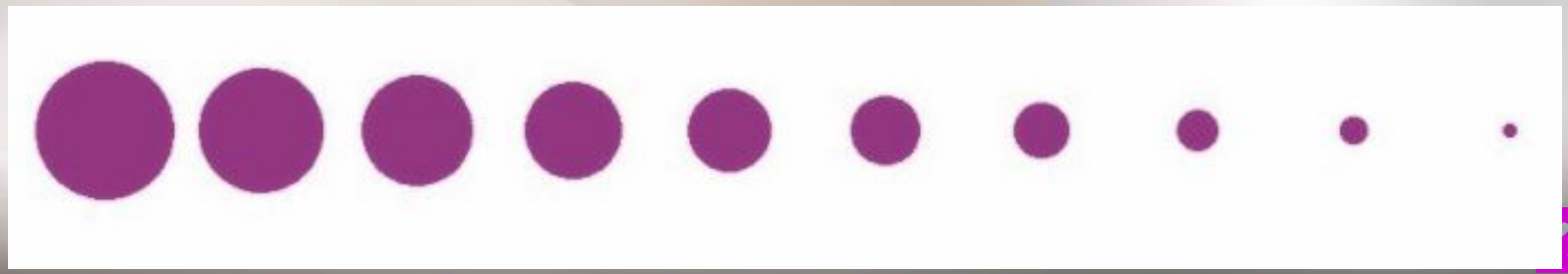




Сортировка выбором / выделением

Суть сортировки:

- Выбирается элемент с наименьшим значением и делается его обмен с первым элементом массива
- Затем находится элемент с наименьшим значением из оставшихся n-1 элементов и делается его обмен со вторым элементом и т.д. до обмена двух последних элементов



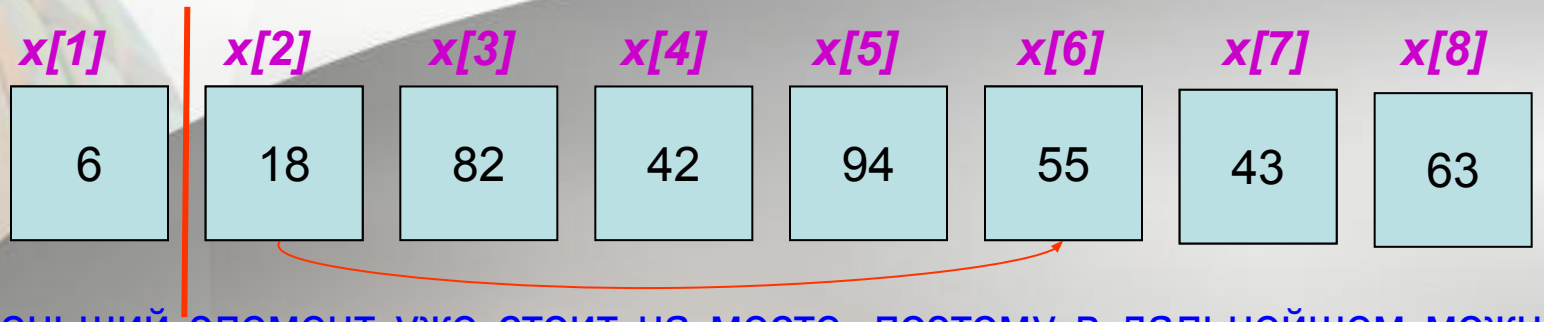
Сортировка выбором / выделением

На первом шаге ищется минимальный элемент *во всем* рассматриваемом массиве

x[1]	*x[2]*	*x[3]*	*x[4]*	*x[5]*	*x[6]*	*x[7]*	*x[8]*
6	55	82	42	94	18	43	63

В данном случае это x[7]=6. Чтобы массив был упорядоченным этот элемент должен стоять на первом месте. Поэтому, совершим обмен значениями между *найденным* и *начальным* элементом массива

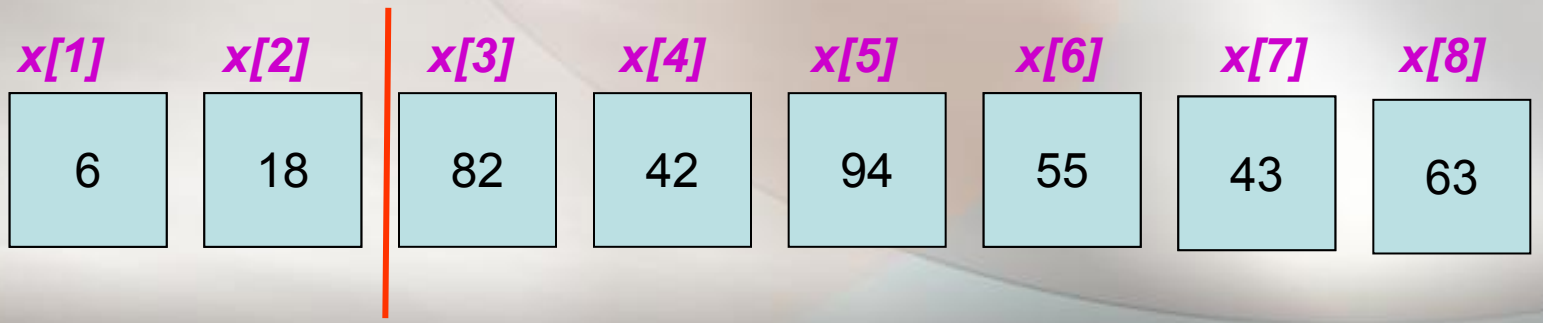
Сортировка выбором / выделением



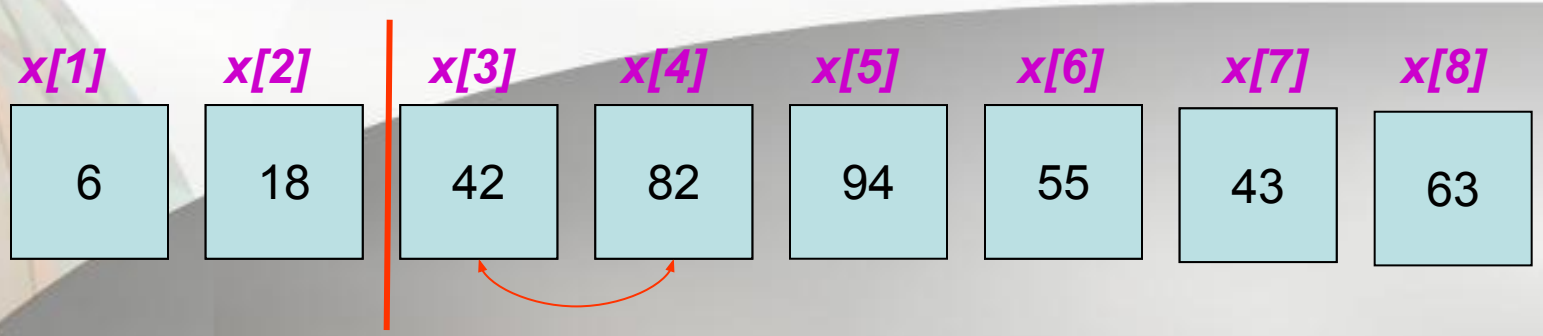
Наименьший элемент уже стоит на месте, поэтому в дальнейшем можно рассматривать уже не весь массив, а только его часть, начинающуюся со второго элемента

На втором шаге ищется минимальный элемент в части массива, начинающейся со *второго* элемента. В данном примере это x[6]=18. И менять шестой элемент нужно с начальным элементом рассматриваемого участка массива, то есть со вторым элементом массива.

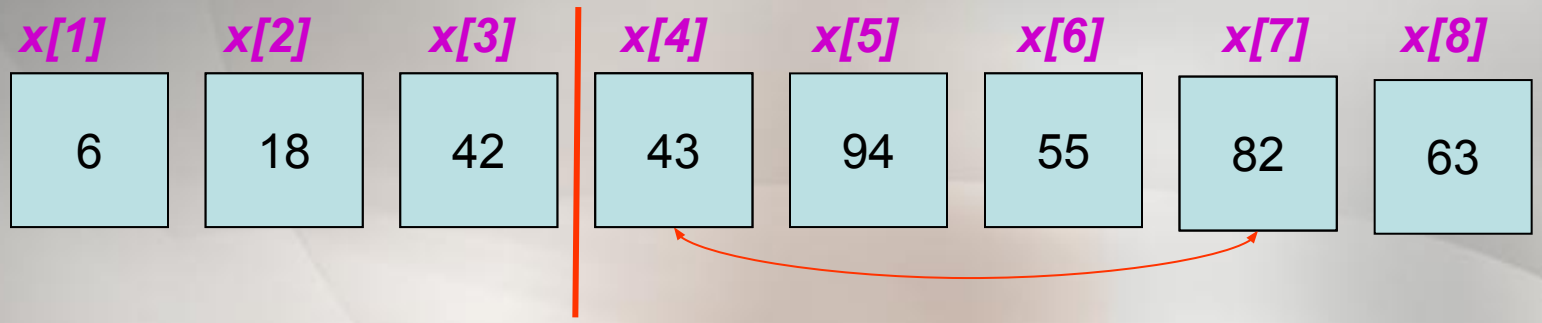
Теперь уже два элемента стоят на своих местах



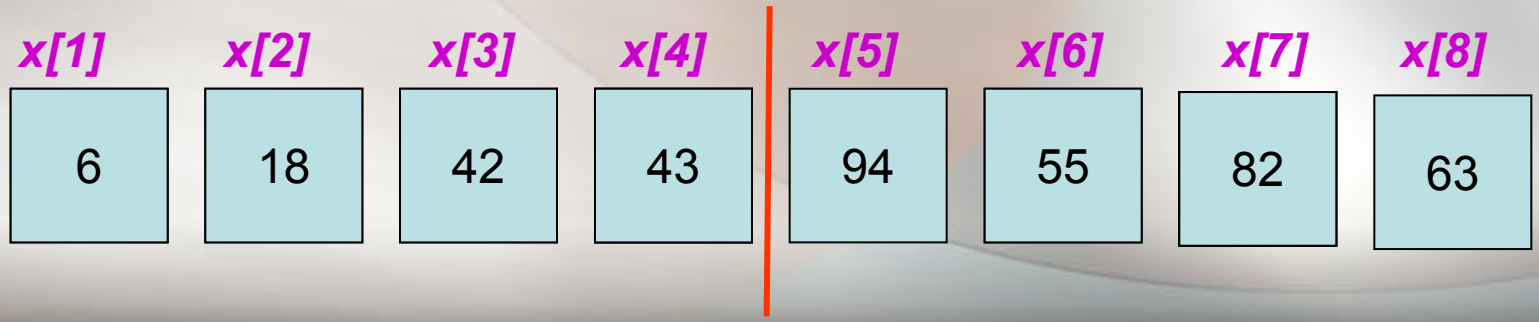
Сортировка выбором / выделением



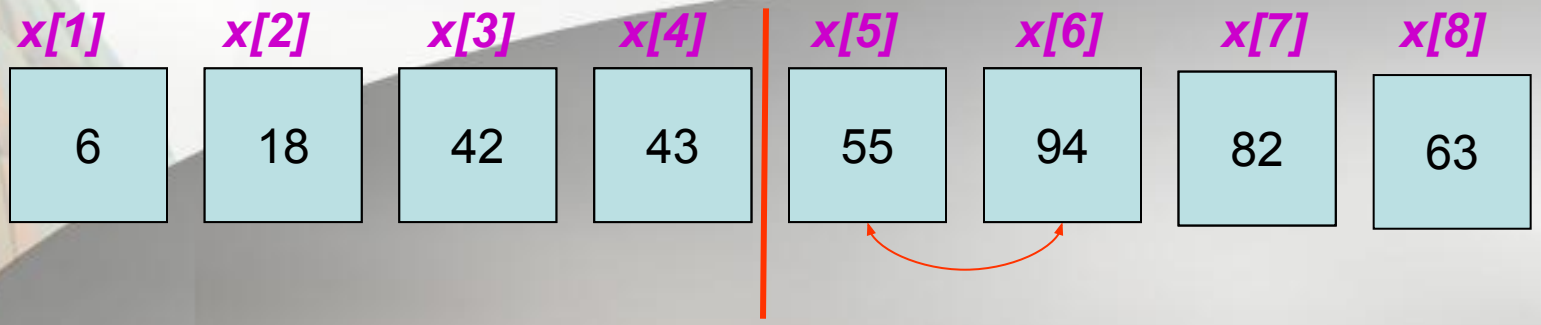
Третий шаг. Минимальный элемент x[4]=42



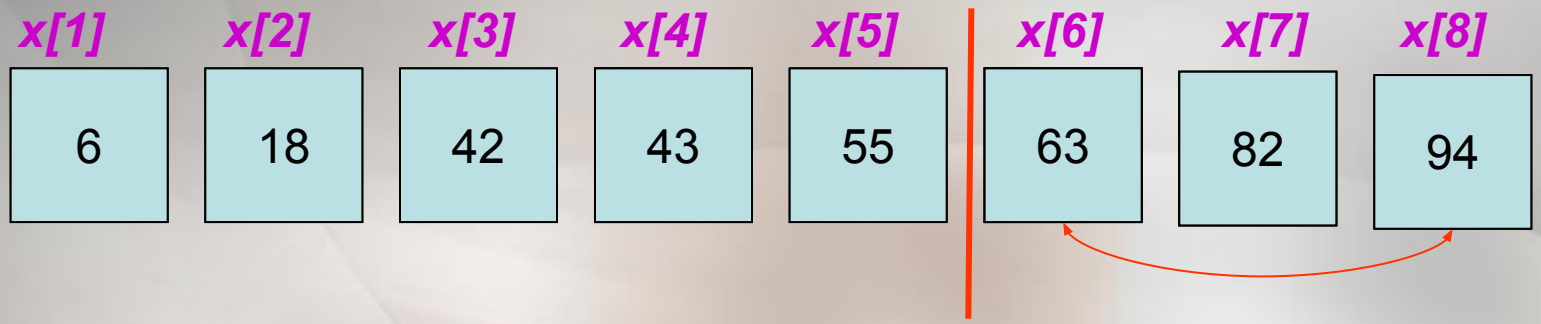
Четвертый шаг. Минимальный элемент x[7]=43



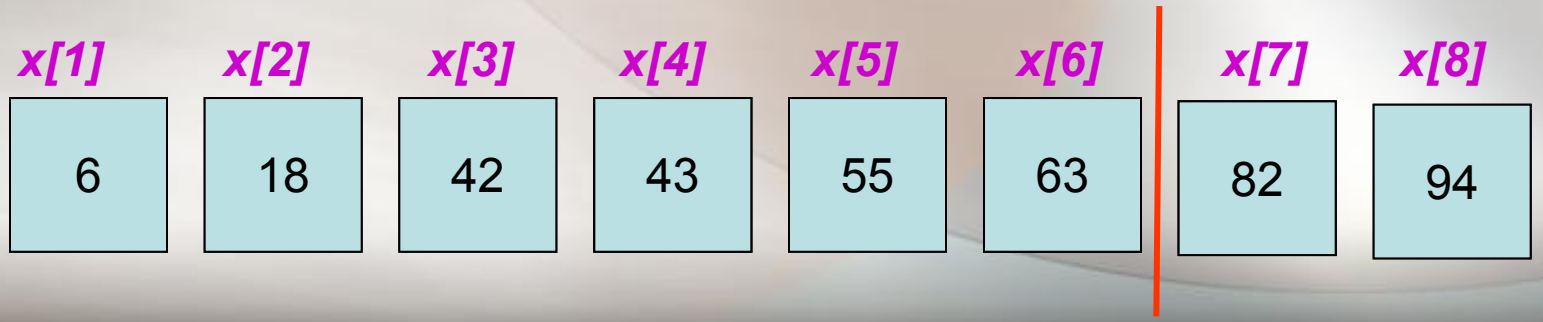
Сортировка выбором / выделением



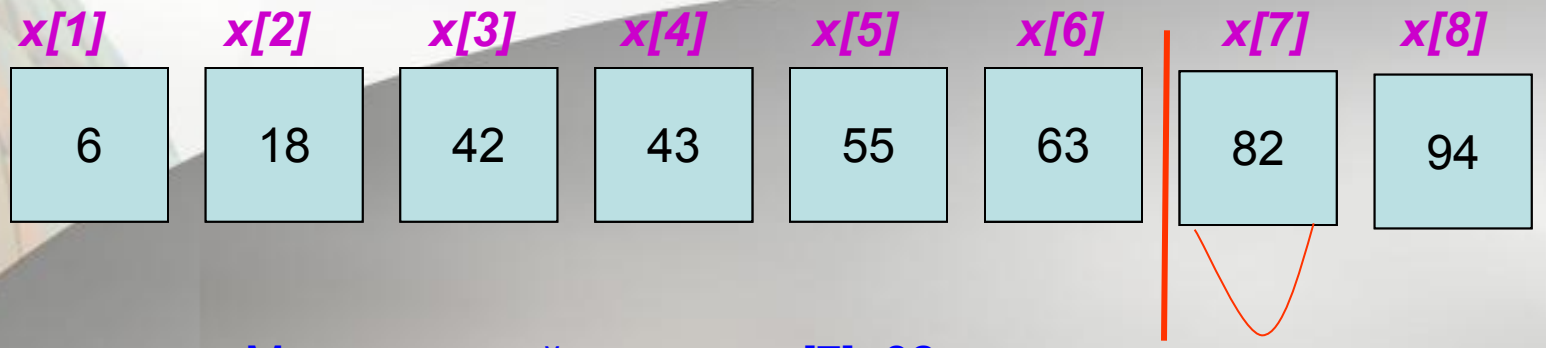
Пятый шаг. Минимальный элемент x[6]=55



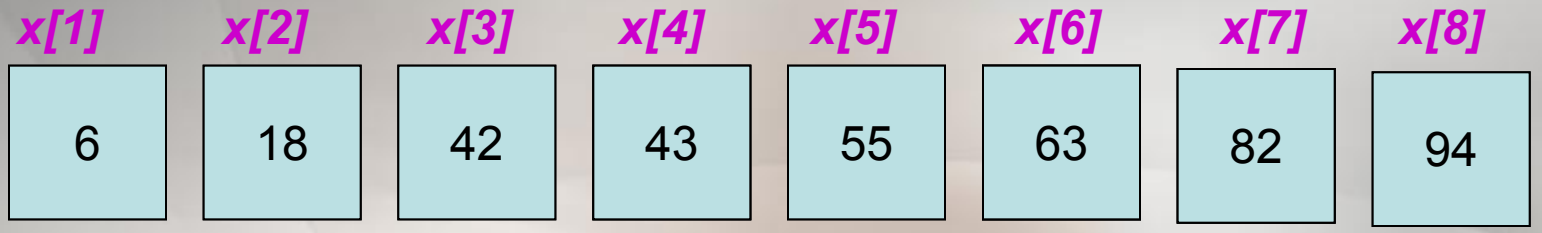
Шестой шаг. Минимальный элемент x[8]=63



Сортировка выбором / выделением



Седьмой шаг. Минимальный элемент x[7]=82



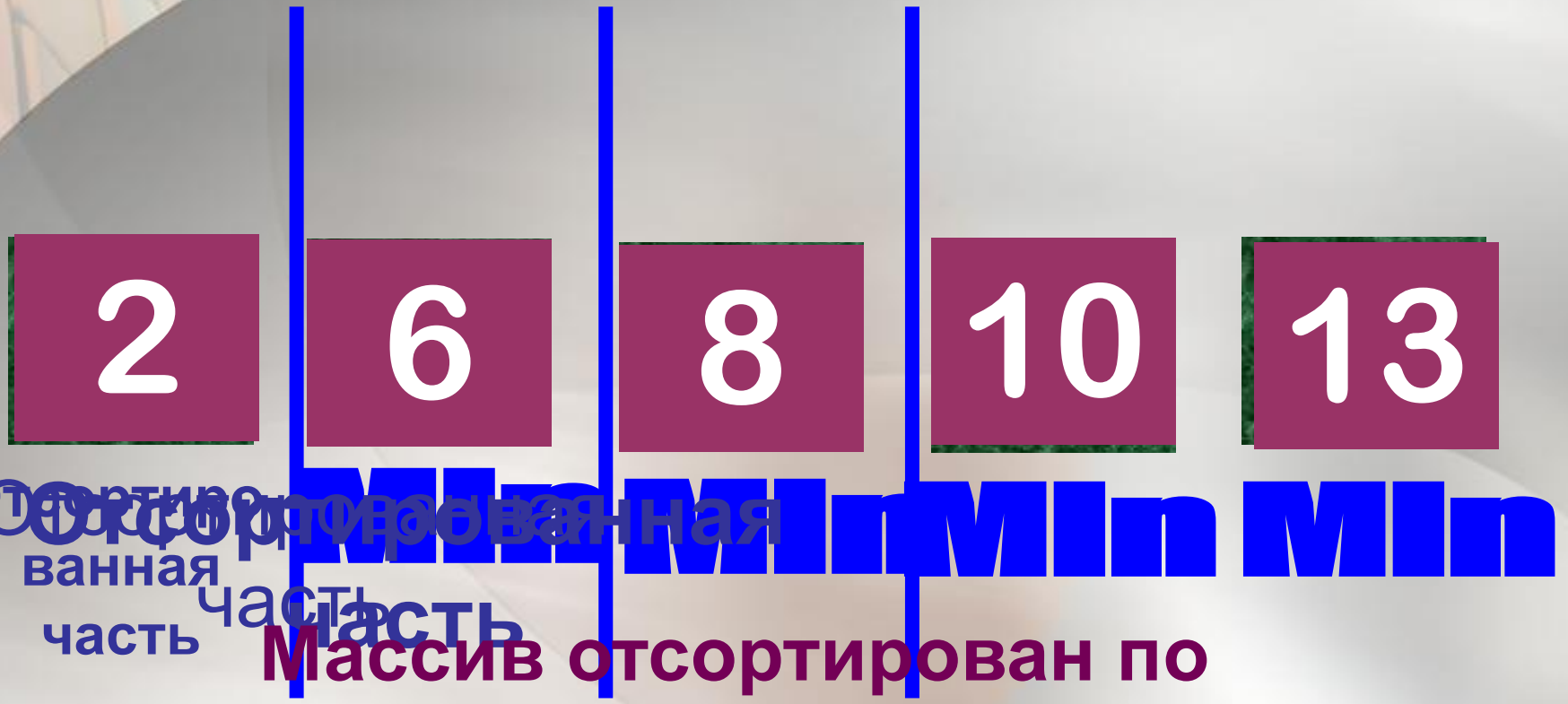
Массив упорядочен

Сущность выполняемых действий

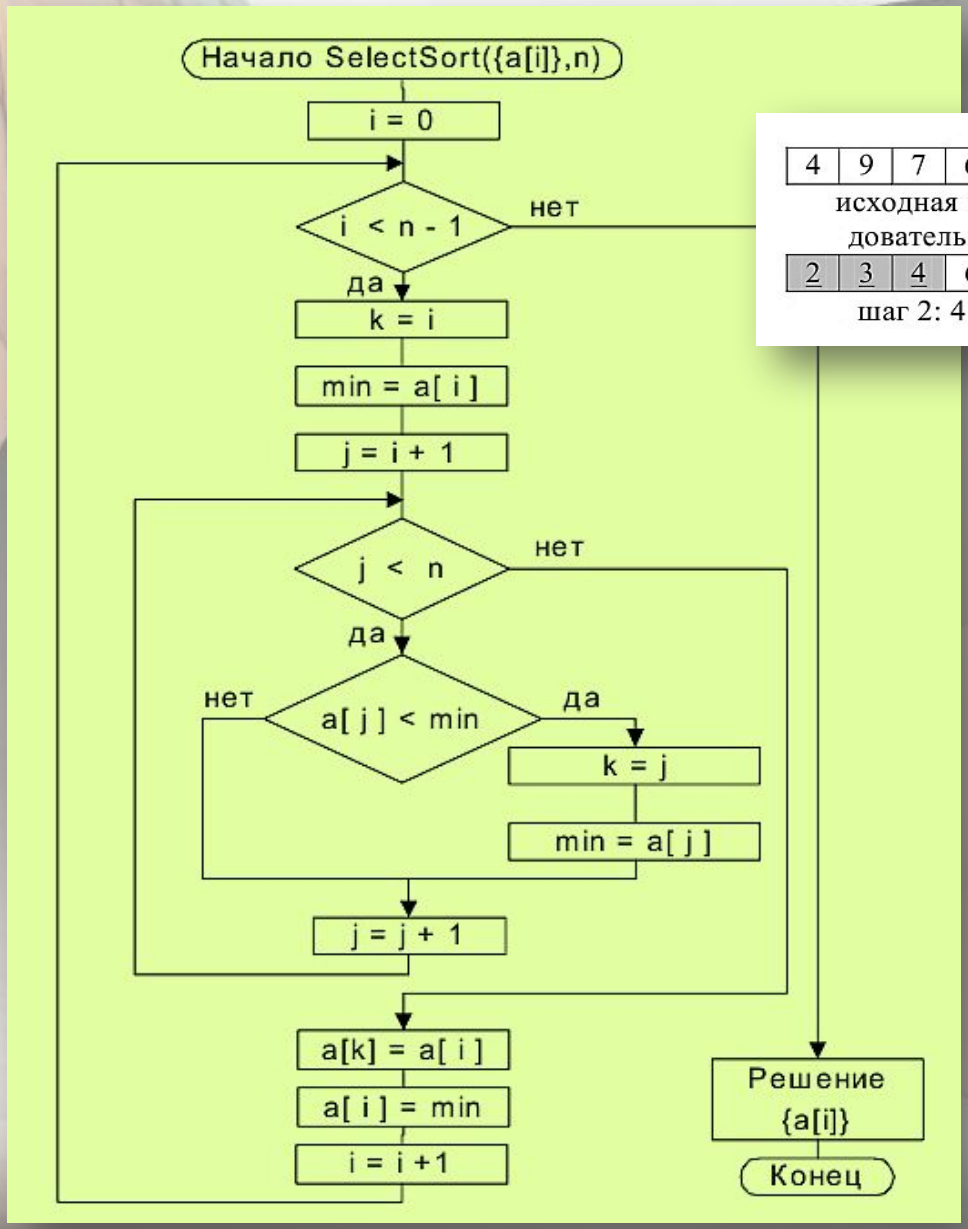
Сортируемый массив разбивается на два участка: уже «готовый», упорядоченный и ещё неупорядоченный. На каждом шаге сортировки *путем перебора неупорядоченного* участка выбирается один элемент и включается в конец уже упорядоченного участка

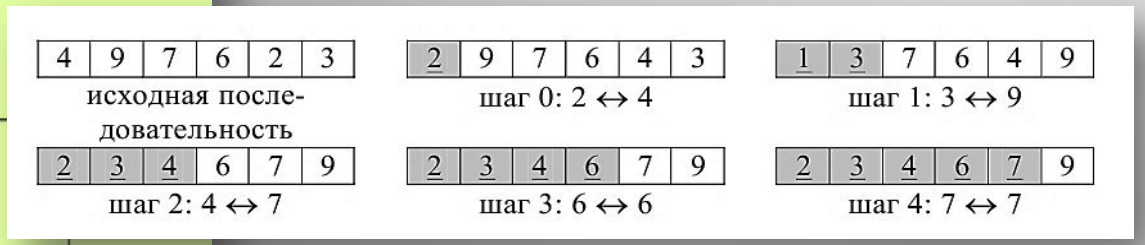
Сортировка выбором / выделением

по возрастанию

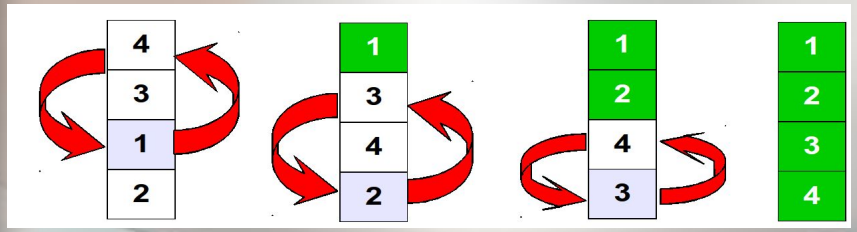


Сортировка выбором / выделением

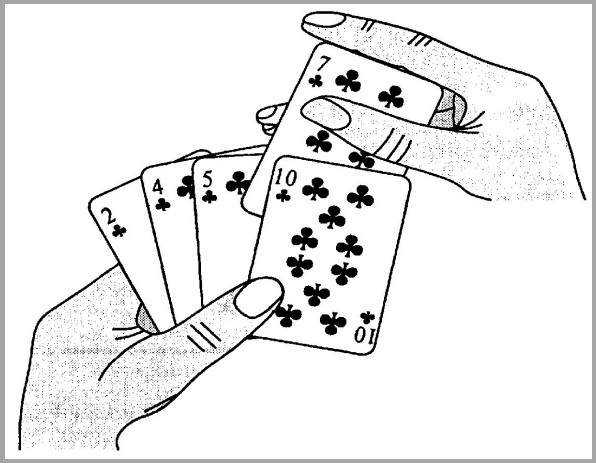




- *Строим готовую последовательность, начиная с левого конца массива*
- *Алгоритм состоит из n-1 последовательных шагов, начиная от нулевого и заканчивая (n-2)-м*
- *На i-м шаге выбираем наименьший из элементов a[i] … a[n-1] и меняем его местами с a[i]*



Сортировка прямыми вставками (прямым включением)



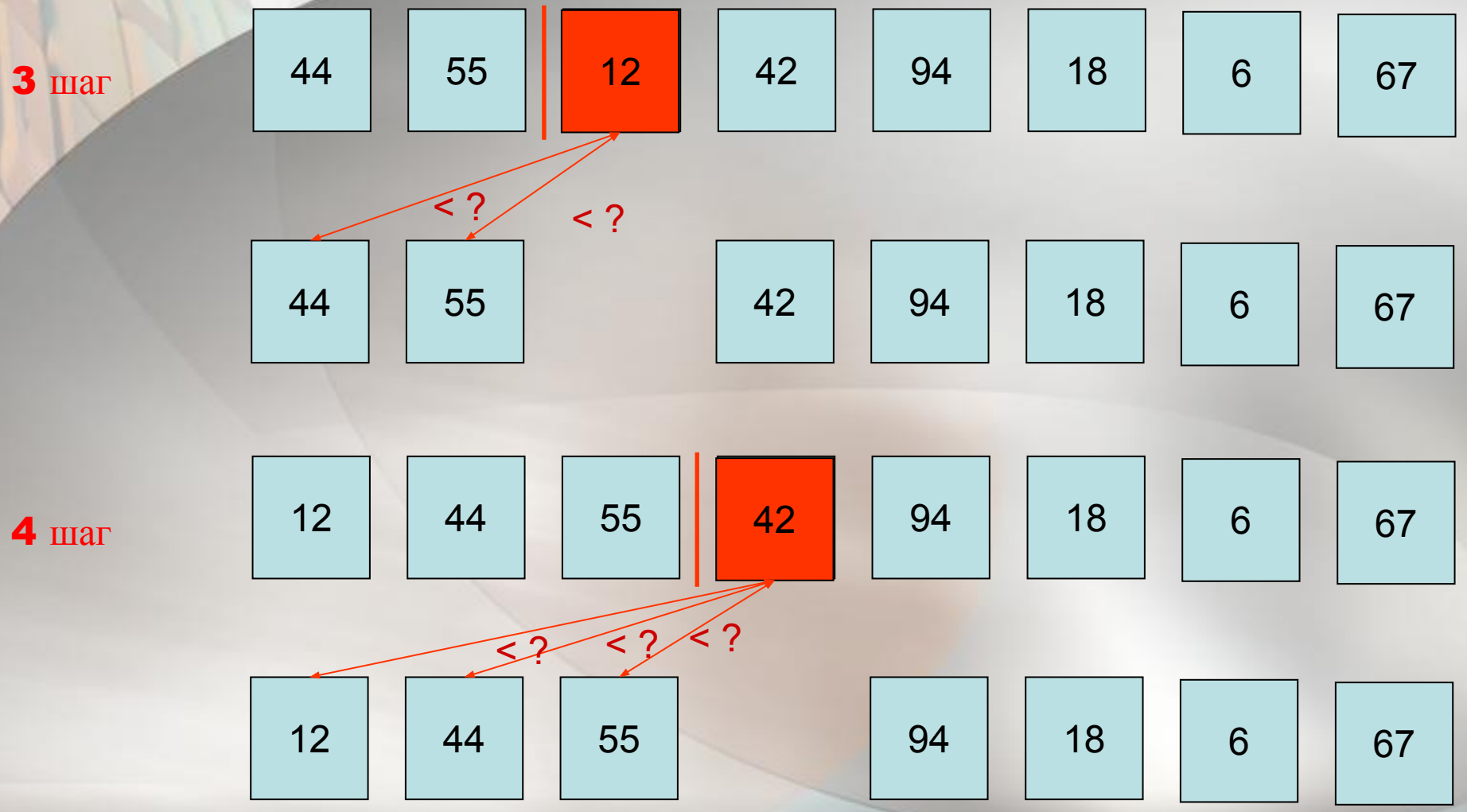
- Сортируемый массив разбивается на два участка: уже «готовый», упорядоченный и ещё неупорядоченный
- На каждом шаге берётся *первый* элемент из неупорядоченной части и *вставляется* в подходящее место в уже упорядоченной части, которое подбирается путём перебора её элементов

В чём принципиальное различие между методом выбора и методом включения?

- Способ определения места вставки - место ищется в *уже упорядоченной* части массива
- Её элементы, начиная с последнего, по очереди сравниваются с элементом, для которого ищется место
- Как только очередной элемент упорядоченной части оказывается меньше, чем новый элемент (сортировка по возрастанию), место вставки найдено.

Сортировка прямыми вставками (прямым включением)

Сортировка прямыми вставками (прямым включением)



Сортировка прямыми вставками (прямым включением)

Суть сортировки:

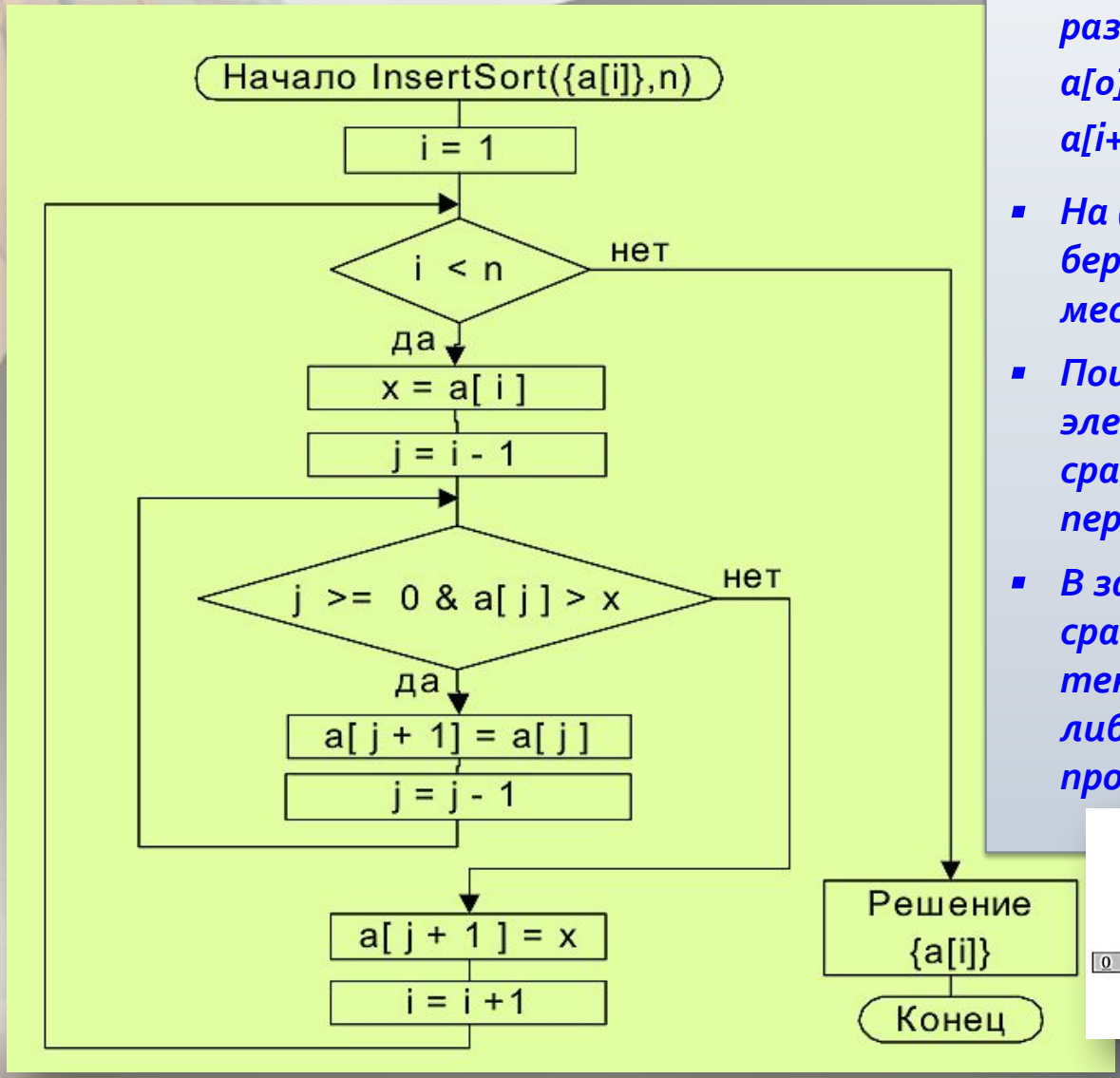
- Упорядочиваются два элемента массива
- Вставка третьего элемента в соответствующее место по отношению к первым двум элементам
- Этот процесс повторяется до тех пор, пока все элементы не будут упорядочены

Сортировка прямыми вставками (прямым включением)

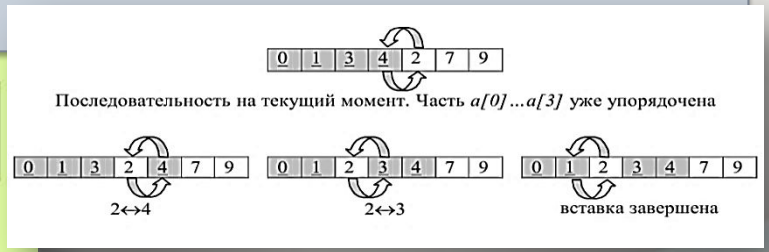
по возрастанию

Массив отсортирован по возрастанию

Сортировка прямыми вставками



- *На i-м шаге последовательность разделена на две части: готовую a[0]…a[i] и неупорядоченную a[i+1]…a[n-1]*
- *На (i+1)-м каждом шаге алгоритма берем a[i+1] и вставляем на нужное место в готовую часть массива*
- *Поиск подходящего места для элемента осуществляется путем сравнений с элементом, стоящим перед ним*
- *В зависимости от результата сравнения элемент либо остается на текущем месте (вставка завершена), либо они меняются местами и процесс повторяется*



Сортировка прямыми вставками (прямым включением)

5	2	1	3	9	0	4	6	8	7

5	2	1	3	9	0	4	6	8	7		

Анализ элементарных алгоритмов сортировок

Замечания к определению временной сложности

- мерой объема входных данных служит количество элементов ***n***, подлежащих сортировке
- время работы программы сортировки может зависеть не только от количества сортируемых элементов, но и от их исходной упорядоченности, два крайних случая :
  - сортируемый массив уже является упорядоченным как требуется - *лучший случай* исходных данных
  - массив упорядочен в обратной последовательности - *худший случай*
- основными операциями сортировки являются *сравнения* элементов и их *перестановка*

Анализ сортировки обменом

Количество сравнений - $(n^2-n)/2$

Общее количество операций – $T(n^2)$

Временная сложность алгоритма *BubbleSort* имеет порядок $O(n^2)$

Анализ сортировки обменом

- Для реализации алгоритма *дополнительная память не требуется*
- Временная сложность данного алгоритма практически не зависит от начальной упорядоченности массива, его *поведение можно считать не естественным*
- Обмен не затрагивает элементы с одинаковыми значениями ключа, ранее стоящие равные элементы "всплывут" к началу последовательности раньше последующих; их естественный порядок следования не изменится, следовательно, *сортировка обменом устойчива*

Анализ сортировки выбором

Количество сравнений - $(n^2-n)/2$

Общее количество операций – $T(n^2)$

Временная сложность алгоритма *SelectSort* имеет порядок $O(n^2)$

Анализ сортировки выбором

- Для реализации алгоритма *дополнительная память не требуется*
- Временная сложность данного алгоритма практически не зависит от начальной упорядоченности массива, его *поведение можно считать не естественным*
- Сортировка выбором осуществляет обмен между первым неупорядоченным элементом и первым найденным минимальным, что, в общем случае может нарушить естественный порядок следования элементов, следовательно, *сортировка выбором неустойчива*

2	a	2	b	1	a

исходная последовательность

1	a	2	b	2	a

шаг 0: 2↔1

Анализ сортировки вставками

Для массива 1 2 3 4 5 6 7 8 потребуется $n-1$ сравнение
Для массива 8 7 6 5 4 3 2 1 потребуется $(n^2-n)/2$ сравнений

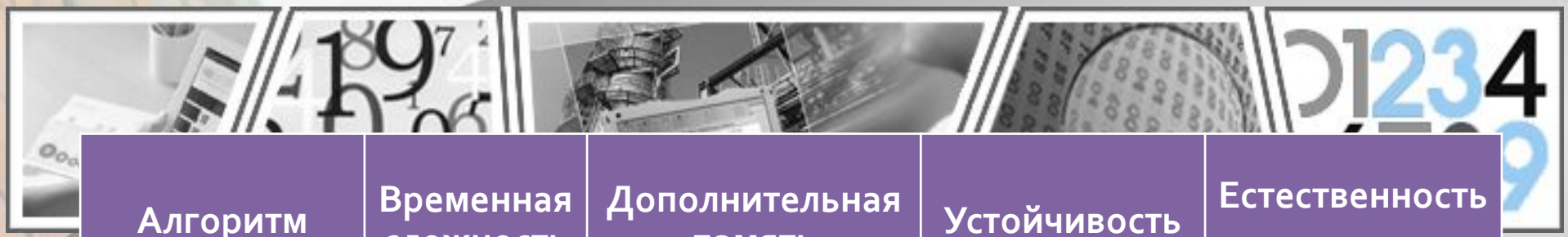
Вывод: Общее количество операций – $T(n^2)$

Временная сложность алгоритма *InsertSort* имеет верхний порядок роста $O(n^2)$ и нижний порядок роста $\Omega(n)$

Анализ сортировки вставками

- Для реализации алгоритма *дополнительная память не требуется*
- почти отсортированный массив будет досортирован очень быстро - *поведение можно считать естественным*
- элементы с равными ключами продвигаются на свои места в отсортированной последовательности в естественном порядке – *этот вид сортировки устойчив*

Анализ элементарных алгоритмов сортировок



Алгоритм	Временная сложность	Дополнительная память	Устойчивость	Естественность поведения
BubbleSort - метод обмена (пузырьковый)	$\Theta(n^2)$	не требуется	да	нет
SelectSort – метод выбора	$\Theta(n^2)$	не требуется	нет	нет
InsertSort - метод вставок	$\Omega(n), O(n^2)$	не требуется	да	да

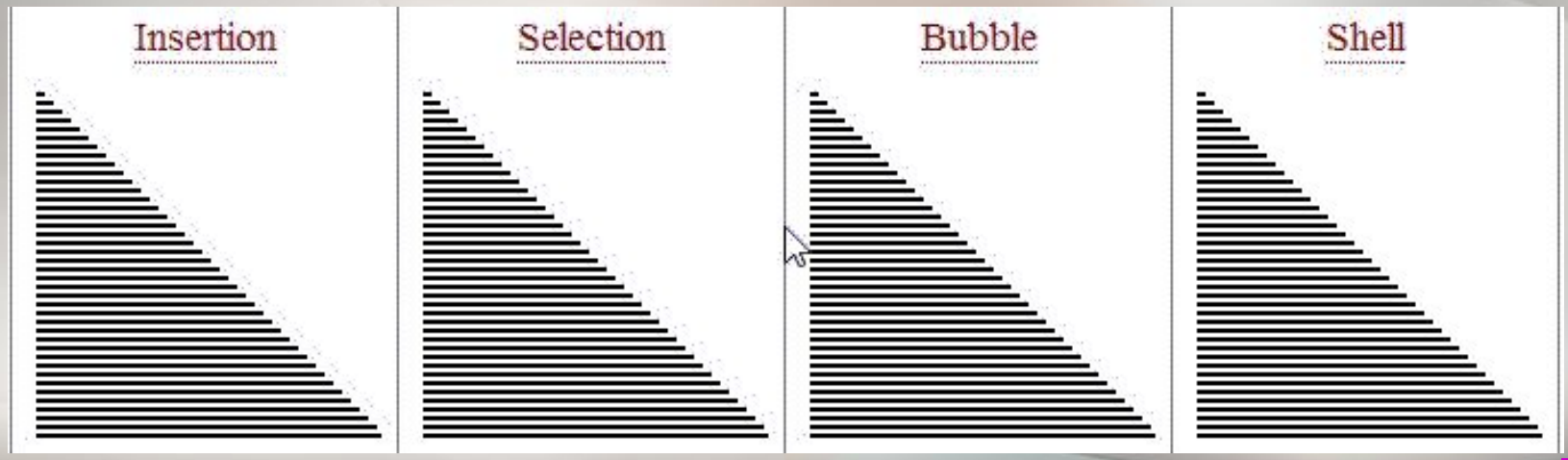
Сравнение методов простой сортировки

	Минимум	Максимум
Простые включения	C = n-1 M=2(n-1)	C=$(n^2-n)/2$ M=$(n^2+3n-4)/2$
Простой обмен	C=$(n^2-n)/2$ M=3(n-1)	C=$(n^2-n)/2$ M=$n^2/4+3(n-1)$
Простой выбор	C=$(n^2-n)/2$ M = 0	C=$(n^2-n)/2$ M=$(n^2-n)*1,5$

N – количество элементов,
M – количество пересылок,
C – количество сравнений

Выбор метода сортировки

- При сортировке небольших массивов (менее 100 элементов) лучше использовать метод «Всплывающего пузырька»
- Если известно, что список уже почти отсортирован, то подойдет любой метод



вставка *выбор* *обмен*

Сортировка вставкой

В совокупности устойчивость и естественность поведения алгоритма, делает метод хорошим выбором в соответствующих ситуациях

Не эффективный метод, так как включение элемента связано со сдвигом всех предшествующих элементов на одну позицию, а эта операция неэкономна

Сортировка обменом

Прост, и его можно улучшать

Очень медленен и малоэффективен.

На практике, даже с улучшениями, работает, слишком медленно, поэтому почти не применяется


Консультации
понедельник – 5 пара
пятница – 4 пара