Работа с одномерными массивами
Методы сортировки

Чернецов Андрей Михайлович,

К.т.н. доцент каф. ПМ

Список литературы по курсу

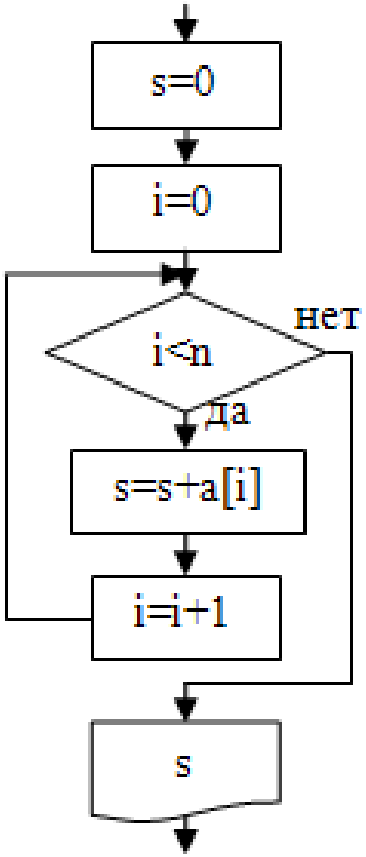
- 1. Князев А.В. Основы языка C++. Учебное пособие. М.: Издательство МЭИ, 2013 – 80 с. ISBN 978-5-7046-1425-8.
- 2. Князев А.В. Работа со сложными структурами данных на языке C++. Учебное пособие. М.: Издательство МЭИ, 2015 – 48 с. ISBN 978-5-7046-1658-0
- 3. Программирование. Сборник задач. Учебное пособие. Санкт-Петербург: Лань, 2019 – 140 с. ISBN 978-5-8114-3857-0

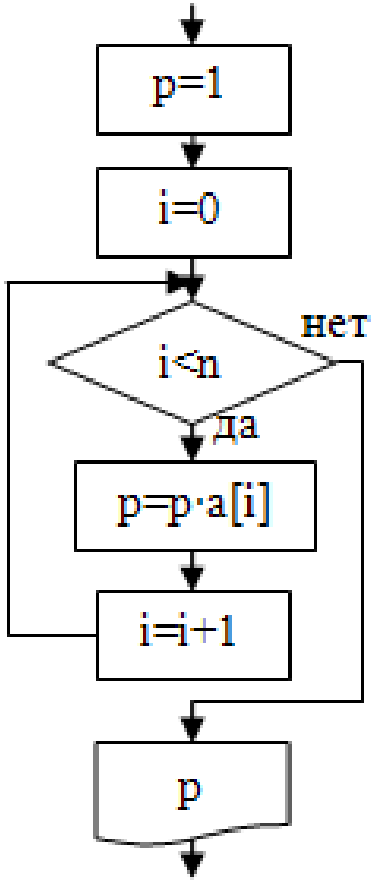
URL: https://e.lanbook.com/book/121485

ТИПОВЫЕ АЛГОРИТМЫ

ОДНОМЕРНЫЕ МАССИВЫ

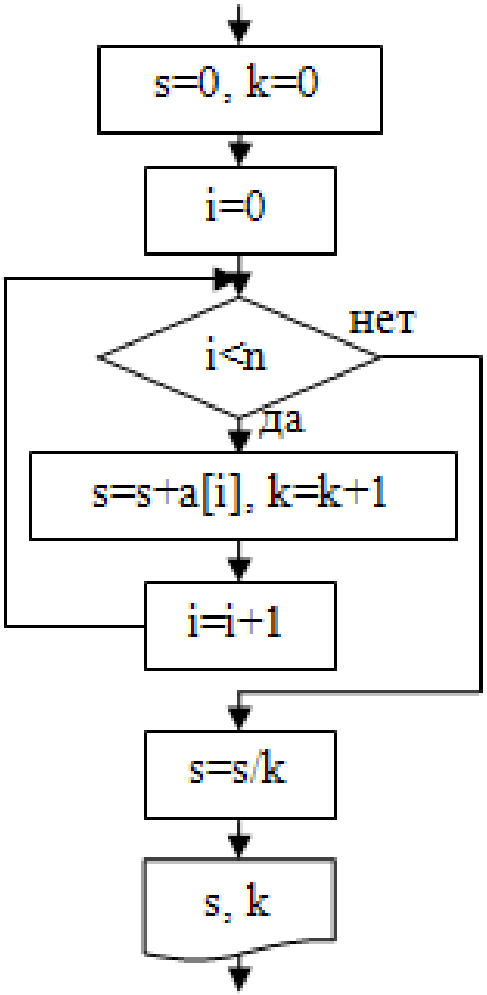
Найти сумму (произведение) элементов массива






```
s=0;
 for(i=0; i<n; i=i+1){
   s=s+a[i]; }

cout<<"s="<<s<<endl;

p=1;
 for(i=0; i<n; i=i+1){
   p=p*a[i]; }
cout<<«P="<<p<<endl;
```

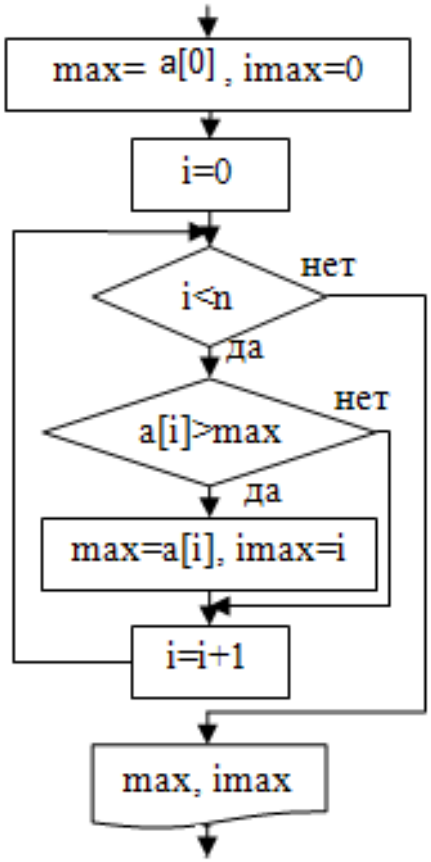
Нахождение среднего арифметического и количества элементов




```
s=0; k=0;
 for(i=0; i<n; i=i+1){
    s=s+a[i];
    k=k+1;
   }

   s=s/k;

 cout<<"s="<<s<<endl;

  cout<<"k="<<k<<endl;
```

K=0 ?

Нахождение максимального элемента массива и его номера



max=0

```
max=a[0];
imax=0;
for(i=0; i<n; i=i+1) {
   if(a[i]>max){
      max=a[i]; imax=i;
   }
 }
cout<<"max="<<max<<" imax="<<imax<<endl;
```

Задача сортировки

Сортировка - это процесс упорядочения некоторого множества элементов, на котором определены отношения порядка >, <. Когда говорят о сортировке, подразумевают упорядочение множества элементов по возрастанию или убыванию.

Алгоритмы сортировки - одна из самых хорошо исследованных областей информатики. Тем не менее не исключены открытия и в этой области, потому что наверняка существуют еще какие-то пока неизвестные методы сортировки, основанные на новых принципах и идеях.

Алгоритмы сортировки имеют большое практическое применение.
Их можно встретить почти везде, где речь идет об обработке и хранении больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные упорядочены.

Методы сортировки

Традиционно различают *внутреннюю* сортировку, в которой предполагается, что данные находятся в оперативной памяти, и важно оптимизировать число действий программы (для методов, основанных на сравнении, число сравнений, обменов элементов и пр.), и *внешнюю*, в которой данные хранятся на внешнем устройстве с медленным доступом (жесткий диск) и прежде всего надо снизить число обращений к этому устройству.

Далее мы будем говорить о ***внутренней*** сортировке.

Задание: Упорядочить массив «на месте» (без использования дополнительного массива) в порядке и направлении перемещения, указанными в условии.

Условие: Упорядочить в порядке убывания (максимум в начало) элементы одномерного массива.

2. Уточненная постановка задачи.

Задан вещественный одномерный массив ***a, состоящий из n элементов.***

Упорядочить в порядке убывания (максимум в начало) элементы заданного массива ***a.***

Пусть n=6.

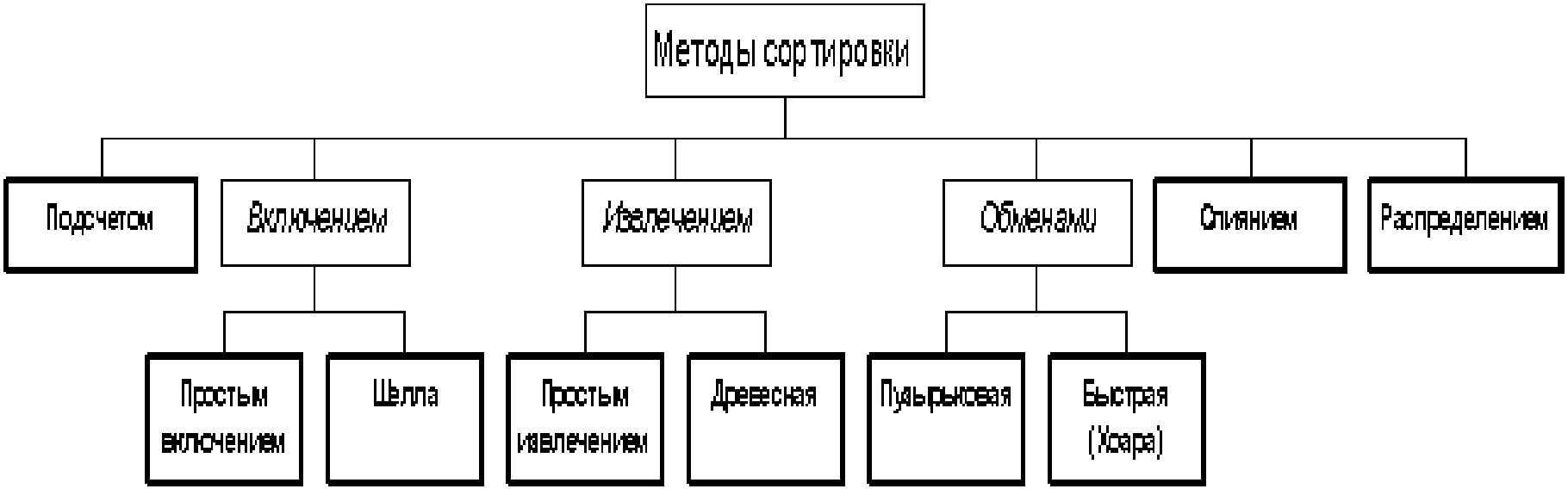
a

4	β	5	9	5	2
1	2	3	4	5	6

В результате должен получиться массив

a

9	5	5	4	3	2
1	2	3	4	5	6


Методы сортировки
Подсчетом
Включением
Извлечением
Обменами
Слиянием
Распределением
Простым включением
Шелла
Простым извлечением
Древесная
Пузырьковая
Быстрая (Хоара)

Входные данные

Имя	**Описание (смысл), диапазон, точность**	**Тип**	**Структура**
a	заданный массив, |ai|<100, точн. 0.1	вещ	одномерный массив (10)
n	число элементов массива a, 0 < ***n*** <= 10	цел	простая переменная

Выходные данные

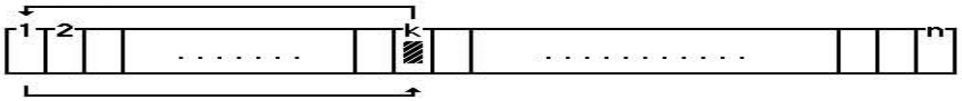
a	упорядоченный массив, |ai|<100, точн. 0.1	вещ	простая переменная

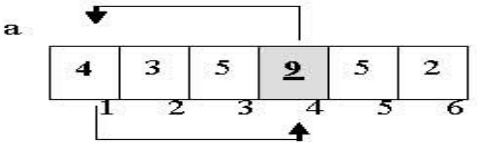
Промежуточные данные

i	индекс текущего элемента, . . .*	*	*
amax	значение максимального элемента	*	*
k	номер максимального элемента.	*	*
z	шаг упорядочения, 1 <= **z** <= 9	*	*

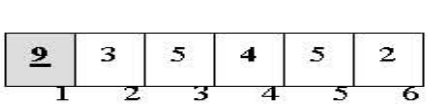
Метод сортировки – метод выбора (максимум в начало):

1 шаг (*z*=1). Ищем в массиве, начиная с *первого* элемента, значение максимального элемента ***amax*** и его номер ***k***, затем меняем значениями первый и ***k***-й элементы.

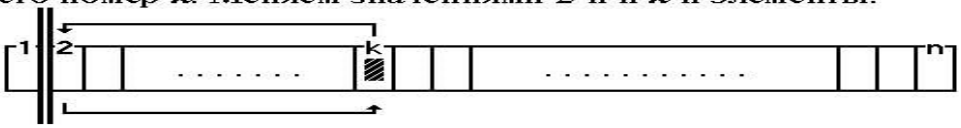


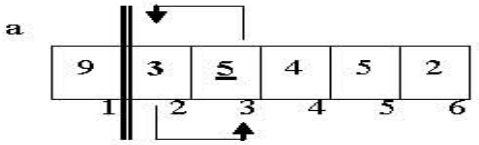


Первый элемент поставлен на место.

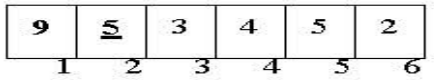


2-й шаг (z=2). Ищем в массиве, начиная со *второго* элемента, значение максимального элемент ***amax*** и его номер ***k***. Меняем значениями 2-й и ***k***-й элементы.

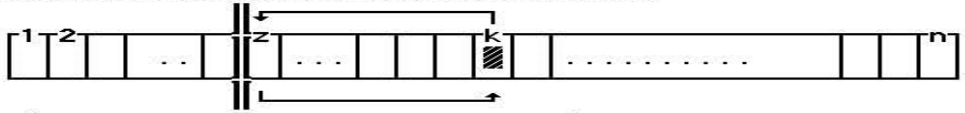




Теперь и *второй* элемент поставлен на место.

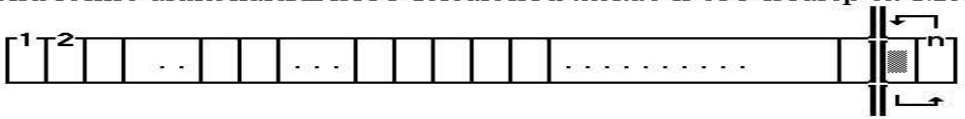


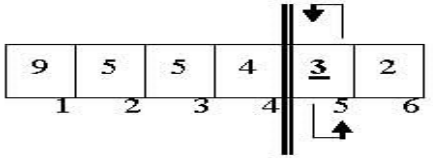
***z*-ый шаг.** Часть массива с первого по (*z*-1)-й элемент уже упорядочена.
Ищем в массиве, начиная с *z*-го элемента, значение максимального элемента ***amax*** и его номер ***k***. Меняем значениями *z*-й и ***k***-й элементы.



z-й элемент тоже поставлен на своё место.

Последний (*z* = ***n***-1) шаг. Часть массива с первого по (***n***-2)-й элемент уже упорядочена.
Ищем в массиве, начиная с **(n–1)**-го элемента (их осталось всего два: последний и предпоследний), значение максимального элемента ***amax*** и его номер ***k***. Меняем значениями (***n***–**1**)-й и ***k***-й элементы.





Теперь и последние два элемента тоже стоят в правильном порядке.
Массив упорядочен.

Замечание 1. Обратите внимание, что в данном массиве элементы фактически оставались на своих местах уже с третьего шага, и далее все элементы менялись местами сами с собой. Внесем небольшую модификацию в алгоритм: будем проверять индексы *z* и ***k*** меняющихся значениями элементов.

Сортировка методом пузырька

Сортировка «пузырьком» (обменом) (максимум в начало) – метод, при котором все соседние элементы массива попарно сравниваются друг с другом, начиная с конца, и меняются местами в том случае, если предшествующий элемент меньше последующего.

В результате этого максимальный элемент постепенно смещается влево и за первый же проход по массиву занимает свое крайнее левое место в массиве, после чего он исключается из дальнейшей обработки. Затем процесс повторяется, и свое место занимает второй по величине элемент, который также исключается из дальнейшего рассмотрения. Для полной сортировки надо выполнить (n-1) проходов по массиву.

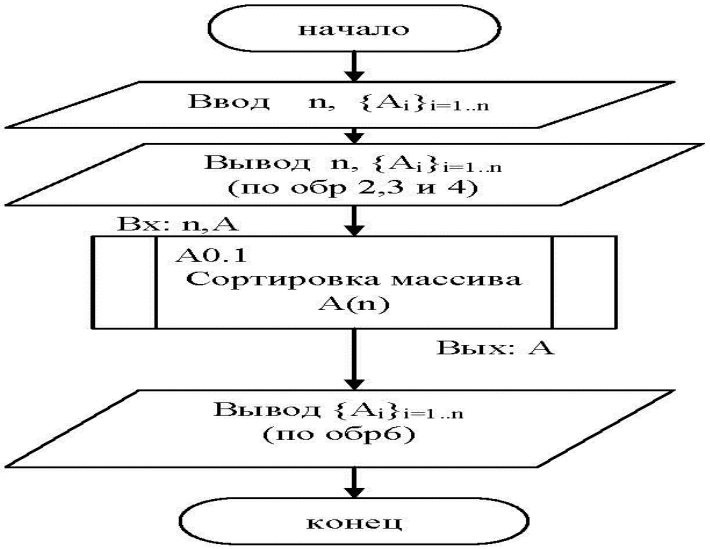
Сортировка пузырьком – один из самых простых, но медленных алгоритмов, имеющий много модификаций. Например, можно изменить его, добавив флажок, показывающий, были ли на данном проходе неупорядоченные пары элементов. Если таких пар не нашлось, закончить сортировку досрочно, а не за (n–1) шаг (проход).

Если последовательность сортируемых чисел расположить вертикально (первый элемент – вверху) и проследить за перемещениями элементов, то можно увидеть, что большие элементы, подобно пузырькам воздуха в воде, «всплывают» на соответствующую позицию.

Поэтому упорядочение таким образом и называют еще сортировкой методом «пузырька», или пузырьковой сортировкой.

Алгоритм.

A0 Основной алгоритм
(отделяем ввод-вывод от обработки)



Раскрываем абстракцию A0.1
(упорядочение методом выбора)

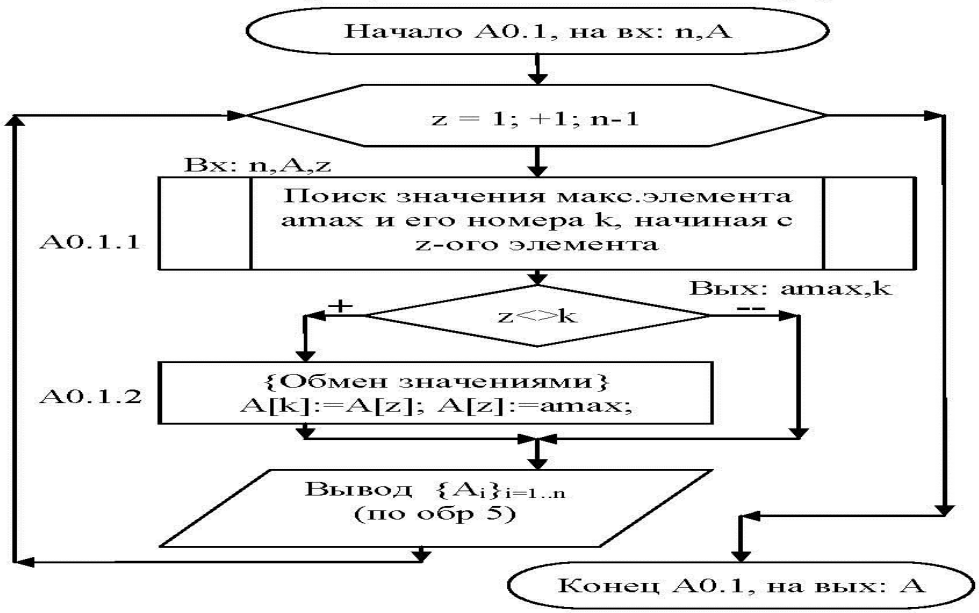
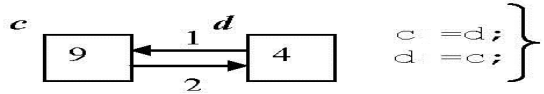


Рис. Блок-схемы алгоритма для задачи *Extremum*

Подзадача A0.1.1 – модификация задачи ***Extremum***: начальное значение индекса элемента, с которого начинается поиск, заменяется с 1 на ***z***.

Подзадача A0.1.2 – обмен значениями ***z***-го и ***k***-го элементов (в ***amax*** лежит ***A*[*k*]**).

Пусть в общем случае необходимо обменять значениями переменные ***c*** и ***d***.

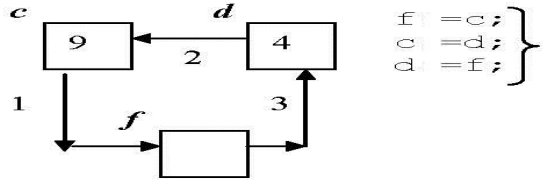


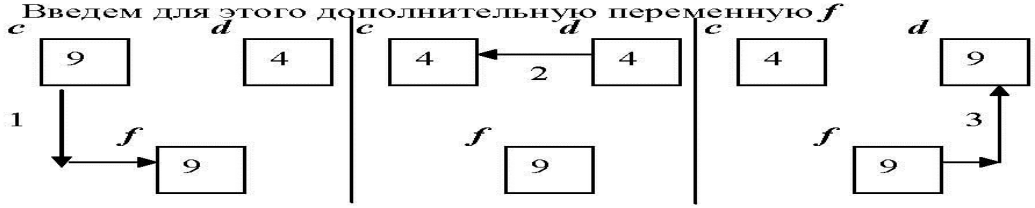
Способ 1).

Неверно, т.к. после первого же присваивания теряется *начальное значение переменной c*:

Надо его сохранить…

Введем для этого дополнительную переменную ***f***





В рассмотренном алгоритме сортировки роль переменной ***c*** играет ***A*[*z*]**, роль переменной ***d*** играет ***A*[*k*]**.
Другие способы обмена: 2) ***c*** = ***c***+***d***; ***d*** = ***c***-***d***; ***c*** = ***c***-***d***;

Добавим в таблицу данных две переменные:

Цел *z – номер прохода. За один проход сравниваем пары соседних элементов (Ai, Ai+1)i=(n-1)..z , т.е. первая рассматриваемая пара (An-1, An), последняя – (Az, Az+1).*

истина, если массив упорядочен (неупорядоченных пар нет), лог *y* =

ложь, в противном случае (есть хотя бы одна пара неупорядоченных соседних элементов).

Вначале каждого прохода следует положить *y=истина (массив упорядочен), но если встретим хотя бы одну неупорядоченную пару элементов, поменяем значение переменной y на ложь.*

Если в конце прохода значение переменной *y осталось истинным, значит, массив упорядочен.*

Начинаем с первого прохода (z:=1)

Повторяем в цикле ДО следующие действия:

- Положим, что массив упорядочен (y:=**истина**)
- В цикле с *убывающим* параметром i= (n–1); -1; z
  - Сравниваем пары соседних элементов:
  - Если пара не упорядочена (A_i<A_{i+1}), то
    - Меняем значение y на **ложь**, и
    - выполняем обмен значениями A_i и A_{i+1}.
- Увеличиваем номер прохода (z:=z+1), готовясь к следующему заходу

ДО тех пор, пока после очередного прохода y не останется истиной, либо не будет сделан (n–1) проход.

Замечание. Если у вас в задании *максимум* надо заставить всплывать не в начало, а *в конец* массива, то просмотр элементов надо начинать с начала массива, а скапливаться уже упорядоченные элементы будут в конце. При поиске *минимума* знак сравнения «<» смениться на «>».

Начинаем с первого прохода (z:=1)

Повторяем в цикле ДО следующие действия:

- Положим, что массив упорядочен (y:=**истина**)
- В цикле с *возрастающим* параметром i=1; +1; (n-z)
  - Сравниваем пары соседних элементов:
  - Если пара не упорядочена (A_i>A_{i+1}), то
    - Меняем значение у на **ложь**, и
    - выполняем обмен значениями A_i и A_{i+1}.
- Увеличиваем номер прохода (z:=z+1), готовясь к следующему проходу

До тех пор, пока после очередного прохода y не останется истиной, либо не будет сделан (n–1) проход.

Пример

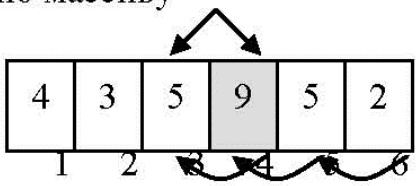
Пусть n=6,

Упорядоченный массив:

Просматриваем массив *с конца до начала*, меняя значениями элементы неупорядоченных пар ***соседних*** элементов. ***За первый проход максимальный элемент всплывет в самое начало***. Но не только он: если бы последние два элемента были не упорядочены, то 5 бы тоже продвинулась на шаг:

1-й проход по массиву

z=1



5 < 9, меняем местами и продолжаем обход