# Сортировки одномерных массивов.

Под сортировкой понимается упорядочивание элементов массива по возрастанию (или по убыванию) без создания копии массива (т.е. «на месте»).

## 1. Сортировка массива методом простого обмена (метод «пузырька»).

Метод основан на том, что в процессе исполнения алгоритма более «легкие» элементы массива постепенно «всплывают». Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов. Проходы по массиву повторяются  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).

Алгоритм сортировки массива по возрастанию методом простого обмена:

1. Начнем просмотр с первой пары элементов (a[1] и a[2]). Если первый элемент этой пары больше второго, то меняем их местами, иначе оставляем без изменения. Затем берем вторую пару элементов ( a[2] и a[3] ), если второй больше третьего, то также меняем их, далее сравниваем третий и четвертый, и если третий больше четвертого, меняем их местами, и т.д. Последними сравниваем (n-1)-ый и n-ый  элементы. При первом обходе массива будут просмотрены все пары элементов массива a[i] и a[i+1] для i от 1 до (n-1). В результате максимальный элемент массива переместится в конец массива.

2. Поскольку самый большой элемент находится на своем месте, рассмотрим часть массива без него, то есть с первого до (n-1) — го элемента. Повторим предыдущие  действия для этой части массива, в результате чего второй по величине элемент массива переместится на последнее место рассматриваемой части массива, то есть на ( n-1 ) — е место во всем массиве.

3. Эти действия продолжают до тех пор, пока количество элементов в текущей части массива не уменьшится до двух. В этом случае

необходимо выполнить последнее сравнение и упорядочить последние два элемента.

Нетрудно заметить, что для преобразования массива, состоящего из n элементов, необходимо просмотреть его n–1 раз, каждый раз уменьшая диапазон просмотра на один элемент.

## Пример работы алгоритма.

Возьмём массив с числами «5 1 4 2 8» и отсортируем значения по возрастанию, используя сортировку пузырьком. Выделены те элементы, которые сравниваются на данном этапе.

Первый проход:

(**5 1** 4 2 8) (**1 5** 4 2 8), Здесь алгоритм сравнивает два первых элемента и меняет их местами.
(1 **5 4** 2 8) (1 **4 5** 2 8), Меняет местами
(1 4 **5 2** 8) (1 4 **2 5** 8), Меняет местами
(1 4 2 **5 8**) (1 4 2 **5 8**), Теперь, ввиду того, что элементы стоят на своих местах, алгоритм не меняет их местами.

Второй проход:

(**1 4** 2 5 8) (**1 4** 2 5 8)
(1 **4 2** 5 8) (1 **2 4** 5 8), Меняет местами
(1 2 **4 5** 8) (1 2 **4 5** 8)

Теперь массив полностью отсортирован, но алгоритму это неизвестно. Поэтому ему необходимо сделать полный проход и определить, что перестановок элементов не было.

Третий проход:

(**1 2** 4 5 8) (**1 2** 4 5 8)
(1 **2 4** 5 8) (1 **2 4** 5 8)

Теперь массив отсортирован и алгоритм может быть завершён.

**Текст программы упорядочивания массива (сортировка по возрастанию).**

```
 {Внешний цикл отвечает за повторение внутреннего цикла столько раз, сколько у
нас элементов массива минус 1}
for var i:=1 to n-1 do begin
  {Внутренний цикл уже перебирает элементы и сравнивает между собой.}
  for var j:=1 to n-i do begin
    {Если элемент, больше следующего, то меняем местами.}
    if m[j]>m[j+1] then
      swap(m[j],m[j+1]);
  end;
end;
```

## 2. Сортировка выбором.

Сортировка выбором – возможно, самый простой в реализации алгоритм сортировки. Как и в большинстве других подобных алгоритмов, в его основе лежит операция сравнения. Сравнивая каждый элемент с каждым, и в случае необходимости производя обмен, метод приводит последовательность к необходимому упорядоченному виду.

Идея алгоритма очень проста. Пусть имеется массив a размером n, тогда сортировка выбором сводится к следующему:

1. берем первый элемент последовательности a[i], здесь i – номер элемента, для первого i равен 1;
2. находим минимальный (максимальный) элемент последовательности и запоминаем его номер в переменную;
3. если номер первого элемента и номер найденного элемента не совпадают, тогда два этих элемента обмениваются значениями, иначе никаких манипуляций не происходит;
4. увеличиваем i на 1 и продолжаем сортировку оставшейся части массива, а именно с элемента с номером 2 по n, так как элемент a[1] уже занимает свою позицию;

С каждым последующим шагом размер подмассива, с которым работает алгоритм, уменьшается на 1, но на способ сортировки это не влияет, он одинаков для каждого шага.

Для сортировки массива по возрастанию:

1. Найти максимальный элемент в массиве.
2. Поместить его на последнее место.
3. Элемент, находившийся в конце массива переместить на место, где прежде находился max.

4. Уменьшить просматриваемую область массива на единицу.
5. Снова найти максимальный элемент в оставшейся области.
6. Поместить его в конец просматриваемой области массива.
7. и т.д.

## Пример работы алгоритма.

| Массив | Описание шага |
|---|---|
| *Первый проход (текущий массив начинается с первого элемента)* | |
| **5** 4 **1** 2 3 | Находим первый минимальный элемент — **1** |
| **1** 4 **5** 2 3 | Меняем минимальный и первый элементы местами |
| *Второй проход (текущий массив начинается со следующего элемента)* | |
| 1 **4** 5 **2** 3 | Находим следующий минимальный элемент — **2** |
| 1 **2** 5 **4** 3 | Меняем минимальный и второй элементы местами |
| *Третий проход (текущий массив начинается со следующего элемента)* | |
| 1 2 **5** 4 **3** | Находим следующий минимальный элемент — **3** |
| 1 2 **3** 4 **5** | Меняем минимальный и третий элементы местами |
| *Четвертый проход (текущий массив начинается со следующего элемента)* | |
| 1 2 3 **4** 5 | Находим следующий минимальный элемент — **4**. Меняем его местами с самим собой. |
| 1 2 3 4 5 | Массив отсортирован |

## Текст программы упорядочивания массива (сортировка по возрастанию).

```
{Внешний цикл отвечает за повторение внутреннего цикла столько раз, сколько у нас элементов массива минус 1}
for var i := 1 to n - 1 do begin
    var min := i;
{находим минимальный элемент последовательности и запоминаем его номер в переменную}
    for var j := i + 1 to n do
        if a[min] > a[j] then
            min := j;
{если номер i-го элемента и номер найденного элемента не совпадают, тогда два этих элемента обмениваются значениями, иначе никаких манипуляций не происходит}
      if min<>i then
        swap(a[i],a[min])
end;
```

## 3. Метод вставок.

Задача заключается в следующем: есть часть массива, которая уже отсортирована, и требуется вставить остальные элементы массива в отсортированную часть, сохранив при этом упорядоченность. Для этого на каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированной части массива, до тех пор, пока весь набор входных данных не будет отсортирован. Метод выбора очередного элемента из исходного массива произволен, однако обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве.

Алгоритм:

-при каждом просмотре элемент вставляется на нужную позицию среди уже упорядоченных элементов

-после вставки i-го элемента первые i элементов будут упорядочены –

Для сортировки требуется два вложенных цикла:

-внешний цикл проходит по всем элементам массива от второго до последнего

-внутренний цикл выполняется, пока текущий элемент не встанет на нужное место в массиве.

### Текст программы упорядочивания массива (сортировка по возрастанию).

```
for var i := 2 to N do
  begin
    var buf := x[i];
    var j := i - 1;
    while (j >= 1) and (x[j] > buf) do
    begin
      x[j + 1] := x[j];
      j := j - 1;
    end;
    x[j + 1] := buf;
        end;
```

**Пример работы алгоритма.**

| До | После | Описание шага |
|---|---|---|
| *Первый проход (проталкиваем второй элемент — **2**)* | | |
| **5 2** 4 3 1 | **2 5** 4 3 1 | Алгоритм сравнивает второй элемент с первым и меняет их местами. |
| *Второй проход (проталкиваем третий элемент — **4**)* | | |
| 2 **5 4** 3 1 | 2 **4 5** 3 1 | Сравнивает третий со вторым и меняет местами |
| **2 4** 5 3 1 | **2 4** 5 3 1 | Второй и первый отсортированы, swap не требуется |
| *Третий проход (проталкиваем четвертый — **3**)* | | |
| 2 4 **5 3** 1 | 2 4 **3 5** 1 | Меняет четвертый и третий местами |
| 2 **4 3** 5 1 | 2 **3 4** 5 1 | Меняет третий и второй местами |
| **2 3** 4 5 1 | **2 3** 4 5 1 | Второй и первый отсортированы, swap не требуется |
| *Четвертый проход (проталкиваем пятый элемент — **1**)* | | |
| 2 3 4 **5 1** | 2 3 4 **1 5** | Меняет пятый и четвертый местами |
| 2 3 **4 1** 5 | 2 3 **1 4** 5 | Меняет четвертый и третий местами |
| 2 **3 1** 4 5 | 2 **1 3** 4 5 | Меняет третий и второй местами |
| **2 1** 3 4 5 | **1 2** 3 4 5 | Меняет второй и первый местами. Массив отсортирован. |

# Задание на пару:

Отсортировать одномерные массивы по возрастанию, убыванию, невозрастанию и неубыванию с помощью указанных методов.

# Домашнее задание:

Изменить решения в трёх рассмотренных методах так, чтобы осуществлялась сортировка **по возрастанию**:

1) четных элементов массива;
2) отрицательных элементов массива;
3) элементов, записанных на нечетных местах.