

Java

Массивы



Массивы

Массив – это группа однотипных элементов, имеющих общее имя и расположенных в памяти рядом.

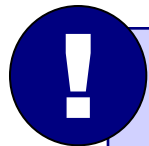
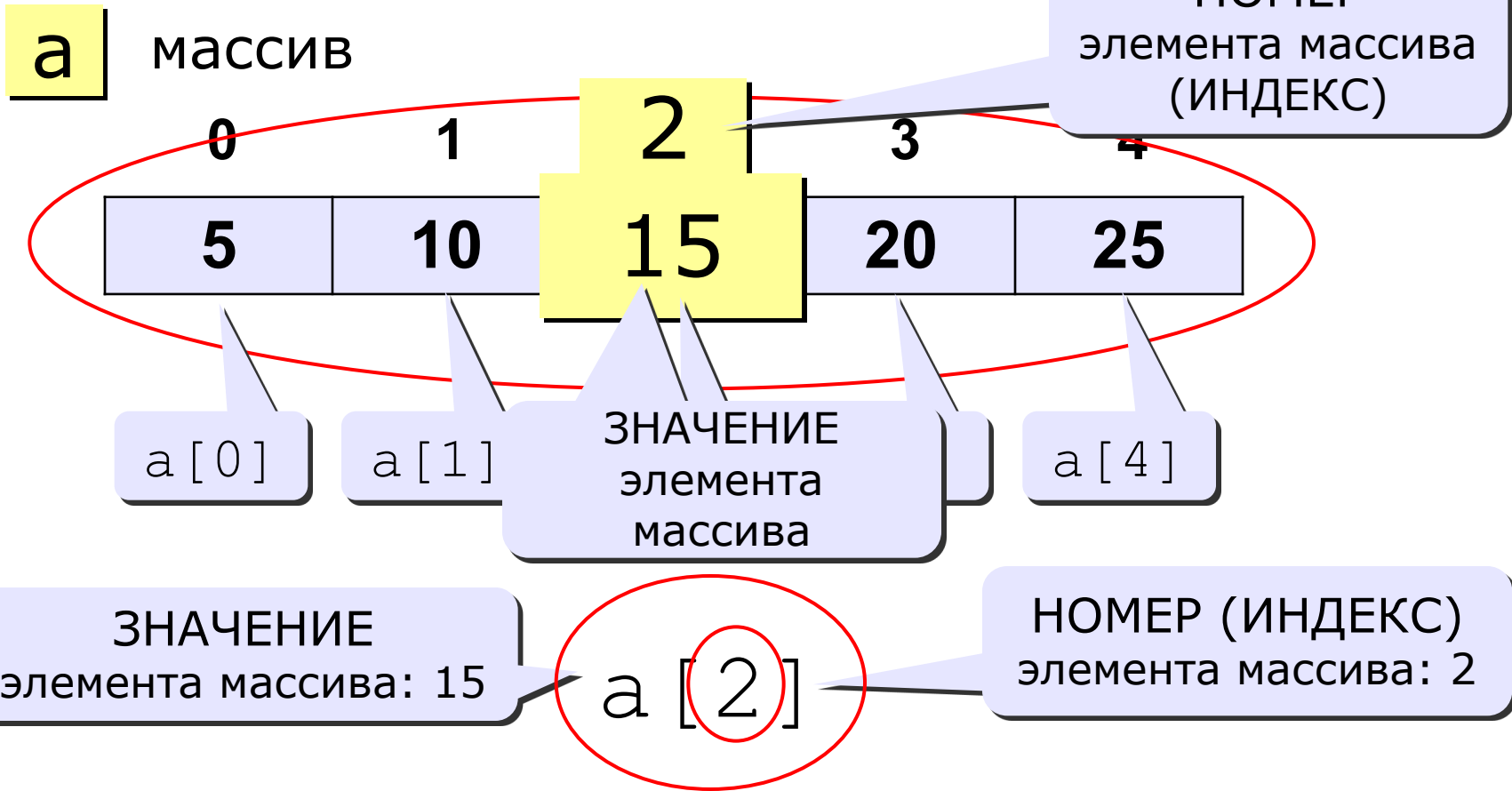
Особенности:

- все элементы имеют один тип
- весь массив имеет одно имя
- все элементы расположены в памяти друг за другом

Примеры:

- список учеников в классе
- школы в городе
- данные о температуре воздуха за год

Массивы



Нумерация элементов массива в Java начинается с **НУЛЯ!**

Объявление массивов

```
тип [] имяМассива;
```

Где **тип** — это тип элементов массива, а **ИМЯ** — уникальный идентификатор, начинающийся с буквы.

Таким образом можно объявить массив любого типа:

```
int[] myFirstArray;  
long[] anArrayOfLongs;  
double[] anArrayOfDoubles;  
boolean[] anArrayOfBooleans;  
char[] anArrayOfChars;  
String[] anArrayOfStrings;
```

Определение массива

```
имяМассива = new тип [количество элементов];
```

для объявленного имениМассива, зарезервируем память при помощи ключевого слова **new**.

Примеры:

```
myFirstArray = new int[15];  
int n = 5;  
anArrayOfDoubles = new double[n];
```

Объявлять имя массива и резервировать для него память также можно на одной строке.

```
int[] myArray = new int[10];
```

Объявление массивов

Еще примеры:

```
int[] cats = new int[6];  
cats[3] = 5;  
cats[5] = 7;
```

С присвоением начальных значений:

```
int[] arr = {0, 1, 2, 3, 4};  
double[] arrDouble;  
arrDouble = {3.14, 2.71, 0, -2.5, 99.123};
```



Все численные типы инициализируются нулями; `boolean` – **false**, остальные типы *null*

Заполнение массива

```
int[] myFirstArray = new int[15];  
for(int i = 0; i < 15; i++){  
    myFirstArray[i] = i;  
}
```

Заполнение массива числами, вводимыми с клавиатуры.

```
Scanner in = new Scanner(System.in);  
int n = in.nextInt();  
int[] arr = new int[n];  
for (int i = 0; i < n; i++){  
    arr[i] = in.nextInt();  
}
```

Вывод элементов массива

```
for (int i = 0; i < n; i++) {  
    System.out.print(arr[i] + " ");  
}
```

Как получить длину массива в Java?

```
int arrLength = arr.length;
```

Как получить последний элемент массива?

```
int lastElem = arr[arr.length - 1];
```

Как заполнить массив случайными числами?

```
for(int i = 0; i < arr.length; i++){  
    arr[i] = (int) round( random() * 10);  
    System.out.print(arr[i] + " ");  
}
```


Массивы Часть II

Обработка массивов

Максимальный элемент

Дополнение: как найти номер максимального элемента?

```
        // пока A[0] – максимальный
iMax = 0;
for (int i=1; i < n; i++ ) // проверяем остальные
    if ( a[i] > a[iMax] ) { // нашли новый
        // запомнить a[i]
        iMax = i;          // запомнить i
    }
```



Как упростить?

По номеру элемента `iMax` всегда можно найти его значение `a[iMax]`. Поэтому везде меняем `max` на `a[iMax]` и убираем переменную `max`.

Максимальный элемент

```
int max = Integer.MIN_VALUE;  
for (int i = 0; i < n; i++) {  
    arr[i] = in.nextInt();  
    if (arr[i] > max) {  
        max = arr[i];  
    }  
}  
System.out.println(max);
```

Дополнение: `min = Integer.MAX_VALUE;`

Удаление элемента

дан массив A:

3 5 6 8 12 15 17 18 20 25

Элемент который нужно

1. **k = 3** удалить

2. 3 5 6 12 15 17 18 20 25 25

```
int k = in.nextInt();
for (int i = k; i < n-1 ; i++) {
    arr[i] = arr[i+1];
}
n--;
```

Вставка элемента

дан массив A:

3 5 6 8 12 15 17 18 20 25

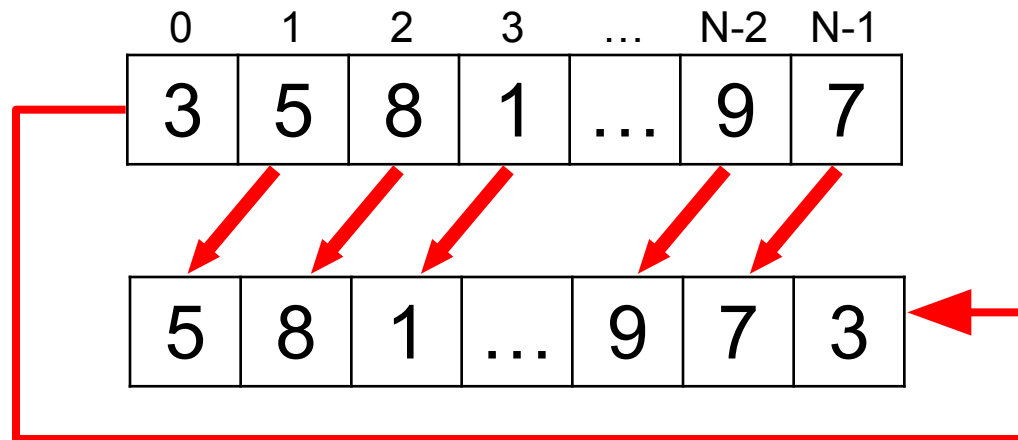
Элемент на место которого

1. **k = 3** нужно вставить новый

2. 3 5 6 x 8 12 15 17 18 20 25

```
int k = in.nextInt();
for (int i = n; i > k ; i--) {
    arr[i] = arr[i-1];
}
n++;
```

Циклический сдвиг I способ



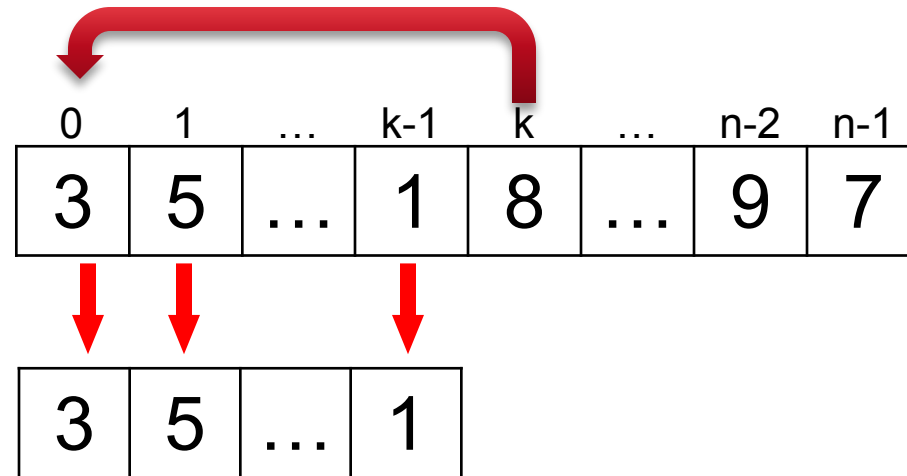
Алгоритм:

1. определить сколько раз необходимо произвести одноэлементный сдвиг $k \% n$;
2. k раз применить одноэлементный сдвиг

Одноэлементный сдвиг :

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

Циклический сдвиг II способ



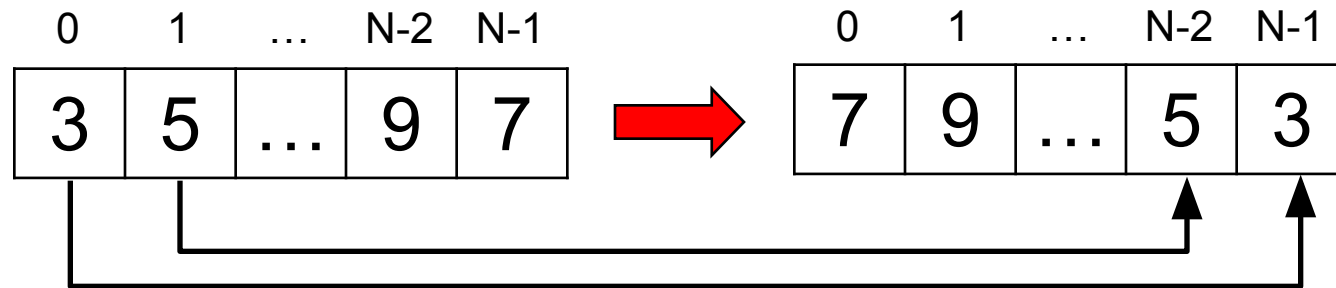
Алгоритм:

1. Скопировать первые **k-1** элементов массива во временный массив
2. Сдвинуть оставшиеся **n-k** элементов влево на **k** позиций
3. Скопировать данные из временного массива обратно в основной массив на последние **k** позиций

```
System.arraycopy(from, fromIndex, to, toIndex, count);
```

Реверс массива

Задача: переставить элементы массива в обратном порядке (выполнить инверсию).



Алгоритм:

сумма индексов $N-1$

поменять местами $a[0]$ и $a[n-1]$, $a[1]$ и $a[n-2]$, ...

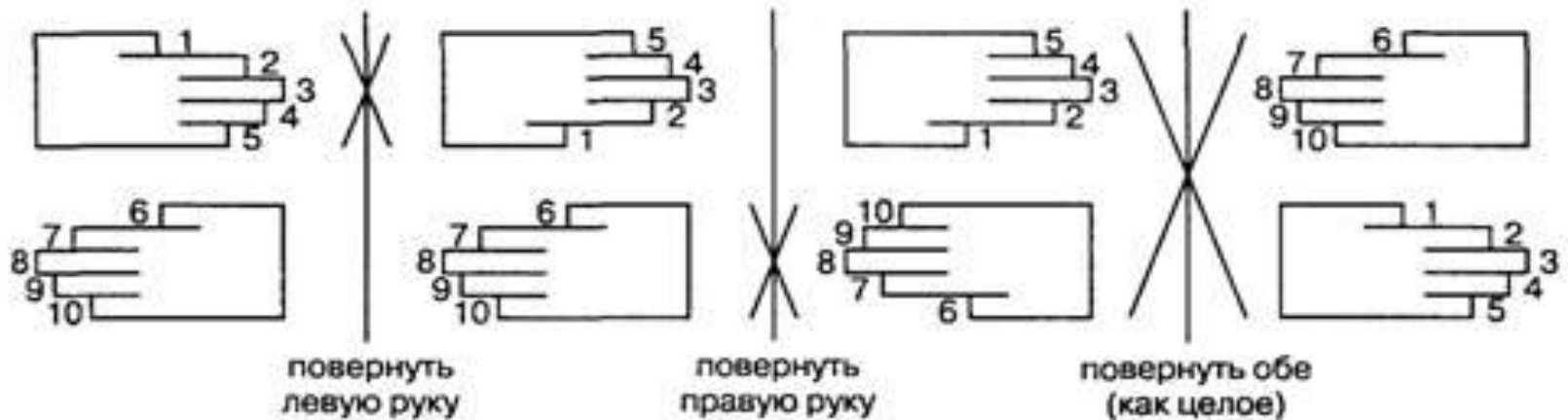
Псевдокод:

```
for ( i = 0; i < n / 2; i++ )  
    //  $a[i]$  ↔  $a[n-1-i]$ 
```


Циклический сдвиг III способ

Алгоритм:

1. отобразить элементы массива $(0, k-1)$
2. отобразить элементы массива $(k, n-1)$
3. отобразить элементы массива $(0, n-1)$



Циклический сдвиг отображениями



```
left = 0; right = k - 1;
count = (right - left + 1) / 2;
for(int i = 0; i < count; i++) {
    temp = arr[left + i];
    arr[left + i] = arr[right - i];
    arr[right - i] = temp;
}
left = k; right = n - 1;
count = (right - left + 1) / 2;
***
left = 0; right = n - 1;
count = (right - left + 1) / 2;
***
```

```
public static void main(String[] args) throws
IOException {
    Scanner sc = new Scanner(new
File("input.txt"));
    int[] a = new int[100000];
    int n = 0;
    while (sc.hasNextInt()) {
        a[n] = sc.nextInt();
        n++;
    }
    sc.close();
    PrintWriter output = new PrintWriter(new
File("output.txt"));
    for (int i = 0; i < n; i++) {
        output.print(a[i] + " ");
    }
    output.close();
}
```

Массивы Часть III

Поиск в массиве

Линейный поиск

indexX – номер
нужного

```
indexX = -1; // пока не нашли элемента в массиве
for ( i = 0; i < n; i ++ ) // цикл по всем элементам
    if ( a[i] == X ) // если нашли, то ...
        indexX = i; // ... запомнили номер
if ( indexX < 0 ) System.out.print ("Не нашли...")
else System.out.print (indexX );
```



Что можно улучшить?

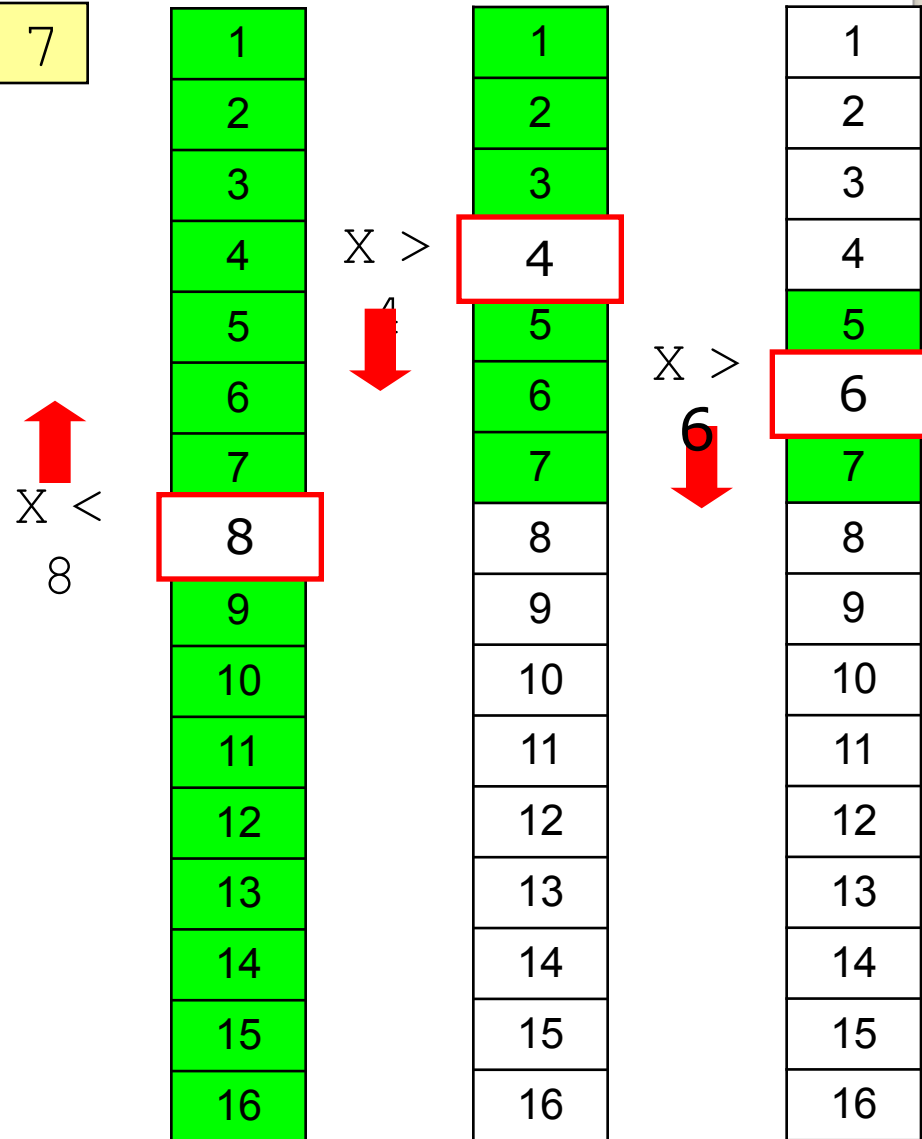
Улучшение: после того, как нашли X, выходим из цикла.

```
indexX = -1;
for ( i = 0; i < n; i ++ )
    if ( a[i] == X ) {
        indexX = i;
        break // выход из цикла
    }
```

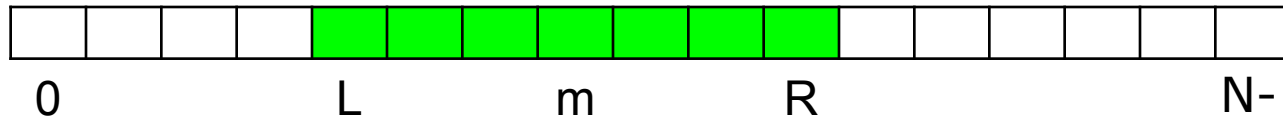
Двоичный поиск

$x = 7$

1. Выбрать средний элемент $a[\text{middle}]$ и сравнить с x .
2. Если $x = a[\text{middle}]$, нашли (выход).
3. Если $x < a[\text{middle}]$, искать дальше в первой половине.
4. Если $x > a[\text{middle}]$, искать дальше во второй половине.



ДВОИЧНЫЙ ПОИСК



```
iX = -1;
left = 0; right = n-1; //ищем от A[0] до A[N-1]
while ( left<=right ) {
    middle = (right + left) / 2;
    if (x == a[middle]) {
        iX = middle ;
        break;
    }
    if (x < a[middle]) right = middle - 1;
    else left = middle + 1;
}

if (iX < 0) System.out.print("Не нашли...")
else System.out.print (iX);
```

номер среднего элемента

если нашли ...

ВЫЙТИ ИЗ ЦИКЛА

сдвигаем границы

Слияние двух упорядоченных массивов

0	1	2	3	4	5	6
1	3	3	5	7	56	70

0	1	2	3	4
2	4	6	8	95

0	1	2	3	4	5	6	7	8	9	10	11
1	2	3	3	4	5	6	7	8	56	70	95


```
Int I = 0;
Int J = 0;
Int k = 0;
while (i <= N-1 && j <= N-1 ) {
    if (arr1[i] < arr2[j])
        { arr3[k] = arr1[i]; i++ ;}
    else
        { arr3[k] = arr2[j]; j + + ; }
    k++
}

while (i <= N-1 )
{
arr3[k] = arr1[i];
i ++;
k ++ ;
}
while (j <= N-1)
{
arr3[k] = arr1[j];
j++;
k++ ;
}
```

Массивы

Часть IV

Квадратичные сортировки массивов

Сортировка

Сортировка – это расстановка элементов массива в заданном порядке (по возрастанию, убыванию, последней цифре, сумме делителей, ...).

Задача: переставить элементы массива в порядке возрастания.

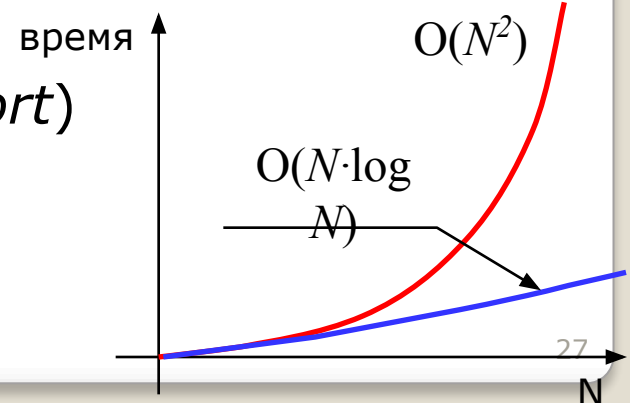
Алгоритмы:

- простые и понятные, но неэффективные для больших массивов
 - метод пузырька
 - метод выбора
- сложные, но эффективные
 - «быстрая сортировка» (*Quick Sort*)
 - сортировка «кучей» (*Heap Sort*)
 - сортировка слиянием
 - пирамидальная сортировка

сложность

$$O(N^2)$$

сложность $O(N \cdot \log N)$



Программа (1-ый проход)

0	5
1	2
...	...
N-2	6
N-1	3

сравниваются пары

$a[0]$ и $a[1]$,

$a[1]$ и $a[2]$

...

$a[n-2]$ и $a[n-1]$

$a[j]$ и $a[j+1]$

```
for( j = 0; j < n-1 ; j++ )
    if ( a[j] > a[j+1] ) {
        c = a[j];
        a[j] = a[j+1];
        a[j+1] = c;
    }
```

Программа (следующие проходы)

2-ой
проход
0



$a[n-1]$ уже на своем
месте!

1

...

N-2

N-1

($i+1$)-ый
проход

```
for ( j = 0; j < n-2 ; j++ )  
    if ( a[j] > a[j+1] ) {  
        temp = a[j];  
        a[j] = a[j+1];  
        a[j+1] = temp;  
    }
```

```
for (int j = 0; j < n - i - 1; j++)  
    ...
```

Программа сортировки "пузырьком"

```
public static void main(String[] args) {  
    int n = in.nextInt();  
    // описать, заполнить массив  
    // вывести исходный массив  
    for (int i = 0; i < n - 1; i++) {  
        for (int j = 0; j < n - i - 1; j++)  
        {  
            if (a[j] > a[j + 1])  
            {  
                temp = a[j];  
                a[j] = a[j + 1];  
                a[j + 1] = temp;  
            }  
        }  
    }  
    // вывести полученный массив  
}
```

Меняем
a[j] и a[j+1]

Программа сортировки "пузырьком"

```
int n = in.nextInt();
    // описать, заполнить массив
boolean flag;
int i = 0;
do{
    flag = false;
    for (int j = 0; j < n - i - 1; j++) {
        if (mass[j] > mass[j + 1]) {
            flag = true;
            temp = mass[j];
            mass[j] = mass[j + 1];
            mass[j + 1] = temp;
        }
    }
    i++;
} while (flag );
```

Сортировка "выбором"

```
int iMax;
for (int i = n - 1; i >= 0; i--) {
    iMax = i;
    for (int j = i; j >= 0; j--) {
        if (mass[j] > mass[iMax]) {
            iMax = j;
        }
    }
    if (iMax != i) {
        temp = mass[iMax];
        mass[iMax] = mass[i];
        mass[i] = temp;
    }
}
```


Сортировка вставкой

Алгоритм:

1. На **k**-ом шаге считаем, что часть массива, содержащая элементы **[0, k-1]** уже упорядочена, то есть $a[0] \leq a[1] \leq \dots \leq a[k-1]$
2. Берем **k**-ый элемент и подбираем для него место в отсортированном массиве такое, чтобы после его вставки упорядоченность не нарушилась. То есть необходимо найти **j**, которое удовлетворяло бы условиям: **$0 \leq j \leq k-1$** , $a[j] \leq a[k] \leq a[j+1]$
3. Вставляем элемент **a[k]** на найденное место.

Сортировка вставкой

Алгоритм:

1. Просматриваем элементы массива (упорядоченного), двигаясь от конца к началу массива (то есть от $k-1$ до 0)
2. Просматриваем пока не будет выполнено одно из условий:
 - a) найдем $a[j] < x$ (будем вставлять между $a[j-1]$ и $a[j]$)
 - b) достигнут левый конец упорядоченной части массива (тогда необходимо x вставить на нулевое место)
3. Пока условие 2 не выполнено будем смещать просматриваемые элементы на 1 позицию вправо, в результате чего в отсортированной части будет освобождено место под x .

