JAVA

ПЛАТФОРМЕНО-НЕЗАВИСИМИ ПРОГРАМНИ ЕЗИЦИ

Лекция 4. Масиви

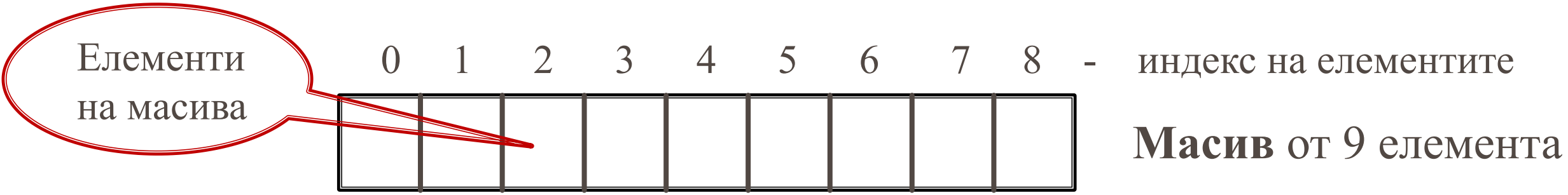
доц. д-р инж. Румен П. Миронов

1. Основни сведения

Масивът представлява съвкупност от променнливи от един и същ тип, които наричаме ***елементи*** на масива.

Особености:

- масивът съдържа фиксиран брой еднотипни елементи;
- дължината на масива се получава при създаването му;
- не се допуска промяна на дължината на масива след като е създаден.



Масив от 9 елемента

Елементите на масивите са номерирани последователно с числата 0, 1, 2, ... N-1. Тези номера на елементи се наричат **индекси**. Броят на елементите в даден масив се нарича **дължина на масива**.

Всички елементи на даден масив са от един и същи тип, независимо дали е **примитивен** или **референтен**.

2. Декларация и дефиниция на масиви

- Декларация на масивите:
  - едномерен масив: тип [ ] array_name;
  - двумерен масив: тип [ ][ ] array_name;
  - не се задават размери, само размерност.

```
int [ ] vector;              // one dimensional
int [ ][ ] matrix;           // two dimensional
double [ ][ ][ ] td;         // three dimensional
String [ ] str;              // string array
```

Числови масиви

Символен масив

При декларация, името на променливата, която е от тип масив, представлява референция (**reference**), която сочи към **null**, тъй като още не е заделена памет за елементите на масива.

В стека за изпълнение на програмата се заделя променлива с име myArray и в нея се поставя стойност null (липса на стойност).

2. Декларация и дефиниция на масиви

- Дефиниция (инстанциираване/създаване) на масив:

  - извършва се чрез оператор **new**

```
int [] a2 = new int [10];
int matrix [][] = new int [8][4];
```

```
int [] a1;      // Java convention
int a1[];       // C/C++ convention
```

- Инициализация на масив:
  - Инициализация по подразбиране (default initial value):

```
int [ ] c = new int [12];
```

Всички елементи на масива приемат стойност по подразбиране в зависимост от типа си - 0 за числени типове, **false** за типа **boolean**, **null** - за референтните типове.

  - Инициализация със списък от елементи:

```
int [ ] n = {10, 20, 30, 40, 50 };
```

- Списъкът се подава във фигурни скоби, разделен със запетаи;
- Не се използва ключовата дума **new**.

Достъп до елементите на масив

- достъпът до елементите на масивите може да е пряк, по индекс. Всеки елемент може да се достъпи с името на масива и съответния му индекс, поставен в квадратни скоби. Можем да осъществим достъп до даден елемент както за четене така и за писане т.е. да го третираме като обикновена променлива.

```
a2[5] = 100;
```

- обхождане с помощта на някоя от структурите за цикъл, като най-често използван е класическият for цикъл.

```
for ( int i = 0; i < a2.length ; i++ )
    a2[i] = i;
```

- обхождане с помощта на for-each цикъл

```
for ( int number : a2 ) {
    // number
}
```

- директен достъп до елементите на масива
- не може да се променят елементите
- няма достъп до индекса на елемента

Достъп до елементите на масив

Граници на масив

Масивите обикновено са нулево-базирани, т.е. номерацията на елементите започва от 0. Първият елемент има индекс 0, вторият 1 и т.н. Ако един масив има N елемента, то последният елемент се намира на индекс N-1.

Полето **length** връща броя на елементите на масива. В Java всеки масив знае своята дължина.

Излизане от границите на масив

Достъпът до елементите на масивите се проверява по време на изпълнение от виртуалната машина на Java и тя не допуска излизане извън границите и размерностите им. При всеки достъп до елемент на масива по се прави проверка, дали индексът е валиден или не. Ако не е се генерира изключение от тип java.lang.ArrayIndexOutOfBoundsException.

Многомерни масиви

- Двумерните масиви представляват масиви от масиви, и ги наричаме **двумерни**, защото имат две измерения или още **матрици** (терминът идва от математиката). Масиви с повече от едно измерение ще наричаме **многомерни**.
- На теория няма ограничения за броя на размерностите на тип на масив, но в практиката масиви с повече от две размерности са рядко използвани, затова се разглеждат подробно двумерните масиви.

- не е задължително дължината на редовете в двумерните масиви да е една и съща;

Декларация	Дефиниция
int [][] intMatrix; **float [][]** floatMatrix; String [][][] strCube;	**int** [][] intMatrix = **new int** [3][4]; **float** [][] floatMatrix = **new float** [8][2]; String [][][] stringCube = **new** String [5][5][5];

Многомерни масиви

Инициализацията на многомерни масиви е аналогична на инициализацията на едномерните. Стойностите на елементите могат да се изброяват непосредствено след декларацията:

```
int [][] matrix = {
        {1, 2, 3, 4},     // row 0 – 4 values
        {5, 6, 7, 8},     // row 1 – 4 values
};
```

Двумерните и многомерните масиви съхраняват стойностите си в динамичната памет като референция към област, съдържаща референции към други масиви.

Всяка променлива от тип масив (едномерен или многомерен) представлява референция към място в динамичната памет, където се съхраняват елементите на масива.

```
int [][]  newType = { {1, 2, 3}, {11, 12, 13, 14}, {21, 22} };
                    // масив с три реда и различен брой колони !!!
```

Достъп до елементите на многомерен масив

- пряк достъп до елементите на многомерен масив, чрез индекс:

```
matrix[2][3]
```

- обхождане с помощта на класически for цикъл:

```
for ( int i = 0; i < matrix.length ; i++ )
        for ( int j = 0; j <  matrix[i].length ; j++ )
                matrix[i][j]                    // i – row number
        }                                       // j – column number
}
```

- обхождане с помощта на for-each цикъл

```
for ( int row : matrix) {
        for ( int column : row ) {          // column
        }
}
```

Достъп до елементите на масив

Можем да извлечем броя на редовете на двумерен масив чрез **matrix.length**. Това на практика е дължината на едномерния масив, съдържащ референциите към своите елементи (които са също масиви). Извличането на дължината на **i**-ия ред става с **matrix[i].length**.

Деклариране чрез списък от аргументи с променлива дължина

- задава се като многоточие (.....) в списъка с аргументи;
- броят на аргументите е неизвестен;
- може да се задава само веднъж в списъка с аргументи на даден метод;
- трябва да се намира в края на списъка с аргументи;
- представлява масив от елементи от един и същи тип.

Достъп до елементите на масив

```
public class VarArgsTest
{                       // calculate average
      public static double average( double... numbers )
      {
            double total = 0.0;              // initialize total
            for ( double d : numbers )       // calculate total using the enhanced for statement
            {
               total += d;
            }
         return total / numbers.length;
      }
       public static void main( String [] args )
       {
            double d1 = 10.0;    double d2 = 20.0;    double d3 = 30.0;    double d4 = 40.0;
            System.out.printf("d1 = %.1f\nd2 = %.1f\nd3 = %.1f\nd4 = %.1f\n\n", d1, d2, d3, d4 );
            System.out.printf("Average of d1 and d2 is %.1f\n", average( d1, d2 ) );
            System.out.printf("Average of d1, d2 and d3 is %.1f\n", average( d1, d2, d3 ) );
            System.out.printf("Average of d1, d2, d3 and d4 is %.1f\n", average( d1, d2, d3, d4 ) );
       }
}
```

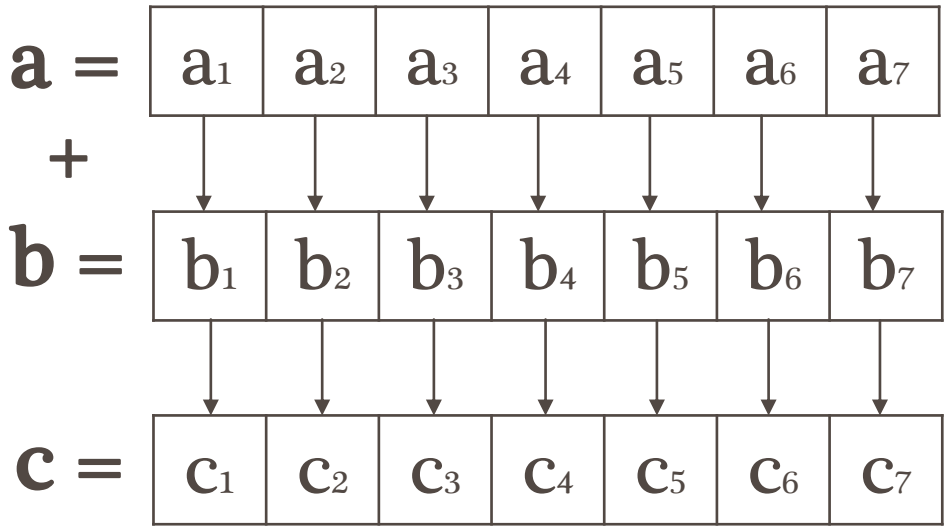
3. Основни операции за работа с масиви

- Базови математически операции с вектори:
  - събиране

$a_1 + b_1$	$a_2 + b_2$	$a_3 + b_3$	$a_4 + b_4$	$a_5 + b_5$	$a_6 + b_6$	$a_7 + b_7$

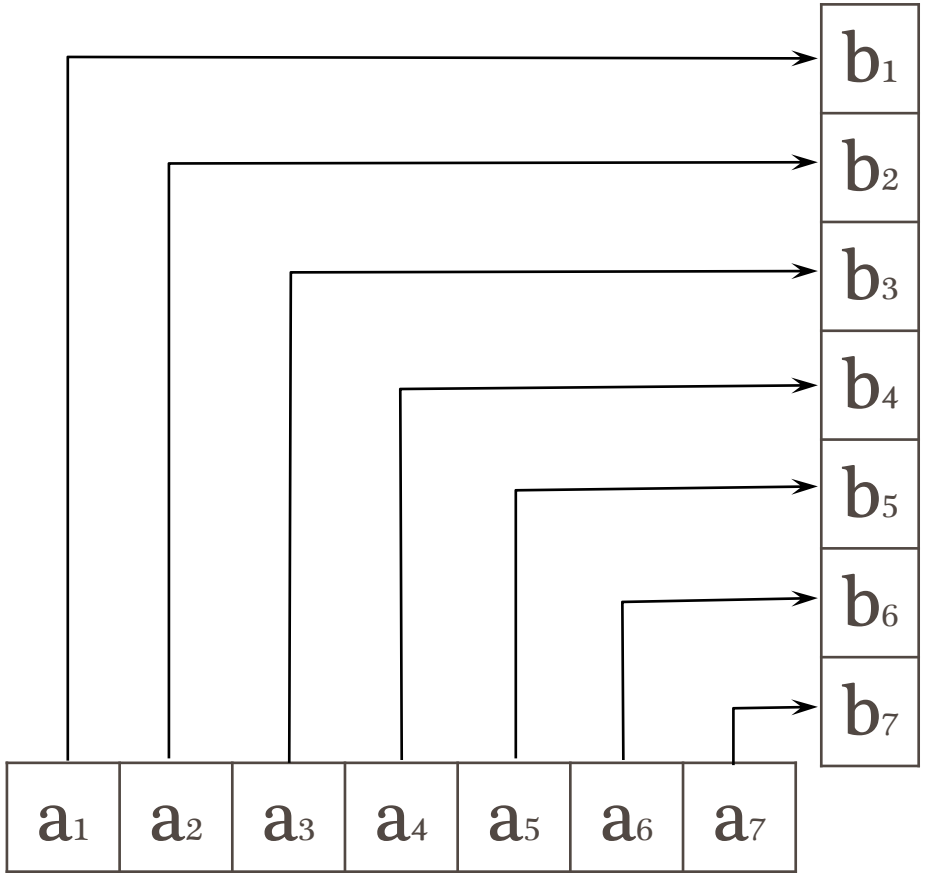
  - изваждане

$a_1 - b_1$	$a_2 - b_2$	$a_3 - b_3$	$a_4 - b_4$	$a_5 - b_5$	$a_6 - b_6$	$a_7 - b_7$



3. Основни операции за работа с масиви

- умножение - скаларно произведение **a*b**



a_1b_1

$a_1b_1+a_2b_2$

$a_1b_1+a_2b_2+a_3b_3$

$a_1b_1+a_2b_2+a_3b_3+a_4b_4$

$a_1b_1+a_2b_2+a_3b_3+a_4b_4+a_5b_5$

$a_1b_1+a_2b_2+a_3b_3+a_4b_4+a_5b_5+a_6b_6$

$a_1b_1+a_2b_2+a_3b_3+a_4b_4+a_5b_5+a_6b_6+a_7b_7$

3. Основни операции за работа с масиви

Матрично събиране: A+B

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

+

b_{11}	b_{12}	b_{13}	b_{14}
b_{21}	b_{22}	b_{23}	b_{24}
b_{31}	b_{32}	b_{33}	b_{34}
b_{41}	b_{42}	b_{43}	b_{44}

=

$a_{11}+b_{11}$	$a_{12}+b_{12}$	$a_{13}+b_{13}$	$a_{14}+b_{14}$
$a_{21}+b_{21}$	$a_{22}+b_{22}$	$a_{23}+b_{23}$	$a_{24}+b_{24}$
$a_{31}+b_{31}$	$a_{32}+b_{32}$	$a_{33}+b_{33}$	$a_{34}+b_{34}$
$a_{41}+b_{41}$	$a_{42}+b_{42}$	$a_{43}+b_{43}$	$a_{44}+b_{44}$

Матрично изваждане: A-B

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

-

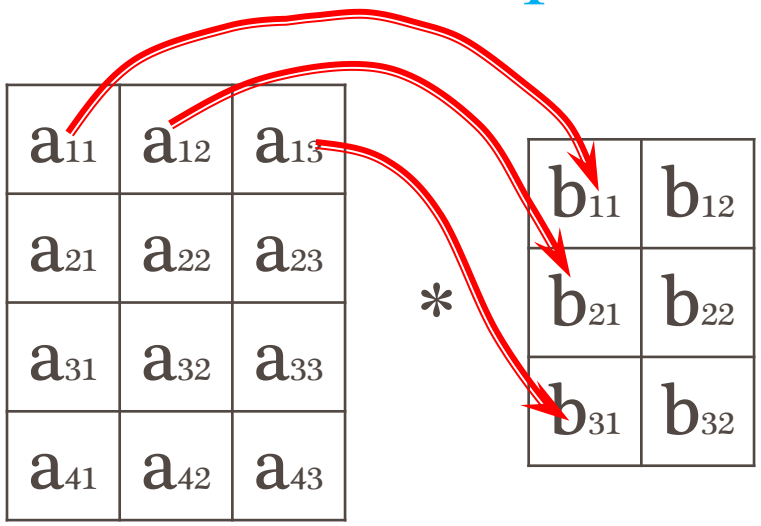
b_{11}	b_{12}	b_{13}	b_{14}
b_{21}	b_{22}	b_{23}	b_{24}
b_{31}	b_{32}	b_{33}	b_{34}
b_{41}	b_{42}	b_{43}	b_{44}

=

$a_{11}-b_{11}$	$a_{12}-b_{12}$	$a_{13}-b_{13}$	$a_{14}-b_{14}$
$a_{21}-b_{21}$	$a_{22}-b_{22}$	$a_{23}-b_{23}$	$a_{24}-b_{24}$
$a_{31}-b_{31}$	$a_{32}-b_{32}$	$a_{33}-b_{33}$	$a_{34}-b_{34}$
$a_{41}-b_{41}$	$a_{42}-b_{42}$	$a_{43}-b_{43}$	$a_{44}-b_{44}$

3. Основни операции за работа с масиви

Матрично произведение A*B



$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} \\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32} \\ a_{41}b_{11}+a_{42}b_{21}+a_{43}b_{31} & a_{41}b_{12}+a_{42}b_{22}+a_{43}b_{32} \end{pmatrix}$$

Повдигане на степен 2 $A^2 = A * A$ – операцията е дефинирана само за квадратни матрици

3. Основни операции за работа с масиви

```
 int [ ] vecA = new int [10];
 int [ ] vecB = new int [10];
 int [ ] vecC = new int [10];
for (int  i = 0; i < vecC.length; i++) {
        vecC[i] = vecA[i] + vecB[i];
}
for (int  i = 0; i < vecC.length; i++) {
        vecC[i] = vecA[i] - vecB[i];
}
 int vecSum =0;
for (int  i = 0; i < vecC.length; i++) {
        vecSum = vecSum+vecA[i]*vecB[i];
}
```

3. Основни операции за работа с масиви

- намиране на максимален/минимален елемент в масив;
- пренареждане на елементите в масив в зададен ред;
- промяна на стойностите на елементите в масив;
- аритметични операции между елементите в масив;
- логически операции между елементите в масив;
- извършване на математически операции между елементите на два или повече масиви;
- визуализация на елементите на даден масив.

❖ Едновременно разпечатване на елементите на масив:
System.out.println(Arrays.toString(a));

Алгоритми за търсене

Алгоритмите за търсене са метод за проверка за елемент или за извличане на елемент от колекция или структура от данни. Както подсказва името, алгоритмите за търсене използват итератор или по друг начин, за да проверяват индивидуално всеки елемент от колекция. Въз основа на типа на извършеното търсене тези алгоритми могат да бъдат широко класифицирани на последователно и интервално търсене.

Последователно търсене - тези техники за търсене включват започване от първия елемент и преминаване през целия списък, докато се намери необходимият елемент. Следователно това води до по-голяма сложност.

Интервално търсене - техниките за интервално търсене се прилагат, когато трябва да се търси в вече сортиран списък. Следователно, в зависимост от критериите за сортиране, се изисква да се търси само сегмент от целия списък, което прави интервалното търсене много по-ефективно от последователното търсене.

4. Сортиране и търсене

Примерни задачи за работа с едномерни и двумерни масиви

- *Записване на елементите в даден масив*
- *Отпечатване на елементите на даден масив*
- *Намиране на сумата на елементите в зададен масив*
- *Намиране на произведението на елементите в зададен масив*
- *Намиране на предварително зададен елемент в масив*
- *Намиране на максимален/миниален елемент в масив*
- *Подреждане на елементите в зададен масив в определен ред*

4. Сортиране и търсене

Алгоритмите за **сортиране** се използват основно за пренареждане на колекция от елементи. Сортирането обикновено се извършва в масиви с цел показване на елементите по определен начин, който се изисква. Сортирането от всякакъв вид се извършва с помощта на оператор за сравнение, който действа като праг за въпросните елементи. Операторът за сравнение определя окончателната структура на колекцията след извършване на сортирането.

Сортиране на място и не на място. Сортирането се извършва по два различни метода на място и не на място. Сортирането на място е, когато сортирането се извършва без използване на допълнителна или външна памет, различна от съответния масив. Два често срещани примера за тази техника са сортирането по избор и сортирането с вмъкване. Сортирането не на място, от друга страна, е, когато техниката изисква допълнителна памет (обикновено променлива за флаг), за да завърши пренареждането на съдържанието на колекция. Сортиране при сливане и сортиране с броене са два примера за сортиране не на място.

4. Сортиране и търсене

Сортиране по метода на мехурчето (Bubble sort)

Обхождайки последователно от началният елемент до крайният, се сравнява всеки елемент със следващия, като те се разменят ако не са подредени. По този начин на всяка стъпка се получава най-големият в края на правилното за него място. Повтаряйки този процедура толкова пъти колкото е размера на масива се постига правилно подреждане на целият масив.

Сортиране с пряка селекция

Намалява се броя на разменянията в сравнение с метода на мехурчето!

Подход- намира се най-големият елемент в масива и директно се поставя на последно място. След това следващият по големина и отново се поставя на неговото място и т.н. докато всички се подредят.

4. Сортиране и търсене

```
class BubbleSort {  // Java program for implementation of Bubble Sort

    void bubbleSort( int arr[] )
    {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n - i - 1; j++)
                if (arr[j] > arr[j + 1]) {      // swap arr[j+1] and arr[j]
                        int temp = arr[j];
                        arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                }
    }
..... ..... ..... .....
}
```

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

4. Сортиране и търсене

```
static void bubbleSort(int arr[], int n) {  // An optimized version of Bubble Sort
    int i, j, temp;   boolean swapped;
    for (i = 0; i < n - 1; i++)  {
        swapped = false;
        for (j = 0; j < n - i - 1; j++) {
            if ( arr[j] > arr[j + 1]) {  // swap arr[j] and arr[j+1]
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
               swapped = true;
            }
        }
       if (swapped == false)
            break;      // IF no two elements were swapped by inner loop, then break
    }
}
```

3. Сортиране и търсене

```
public static void bubblesort(int arr[], int n) {// function for sorting array recursively with bubble sort
   if (n == 0 || n == 1) {                    // base condition for recursion
     return;
    }
   for (int i = 0; i < n - 1; i++)  {         // if arr[i] greater than arr[i+1] then swap(arr[i], arr[i+1])
     if ( arr[i] > arr[i + 1])  {
       swap(arr, i, i + 1);
     }
   }
    bubblesort(arr, n - 1);
 }
public static final void swap (int [] arr, int i, int j) { // swap elementc of arr[] present at index i & j
    int temp = arr[i];      arr[i] = arr[j];
    arr[j] = temp;
  }
}
```

4. Сортиране и търсене

Сортиране с вмъкване

Постепенно сортиране на все по-голяма част от масива, като обхождайки несортираната част всеки един елемент се поставя в сортираната част с намиране на правилното за него място, на което сортираният масив остава подреден.

Сортиране чрез сливане (merge sort)

Два сортирани масива може да се слеят в един масив и то със линейна сложност. Тогава ако се раздели масива който ще сортираме на по-малки масиви и на всяка стъпка се сливат два по-малки масива в един голям, то за log(N) стъпки ще се слеят всички масиви до един масив, като всяка от стъпките е била линейна.

Сортиране със сливане (Забавно видео):

https://www.youtube.com/watch?v=XaqR3G_NVoo

4. Сортиране и търсене

Сортиране с вмъкване

```
void insertionSort( int [] arr ) {
 for (int i = 1; i < arr.length; i++) {
   int next = arr[i];
   int j = i-1;
  while ( j >= 0 && arr[j] > next) {
            arr[j+1] = arr[j];
            j --;
     }
   arr[j+1] = next;
  }
}
```

Сортиране с пряка селекция

```
for (int i = 0; i < arr.length; i++) {
        //обхожда като за началото взима всеки
        // един елемент последователно

    for (int k = 0; k < arr.length-i; k++) {
        // обхожда втория елемент за сравнение
                if (arr[i]>arr[i+k]) {
                        int a = arr[i];
                        arr[i] = arr[i+k];
                        arr[i+k] = a;
                }
        }
}
```

4. Сортиране и търсене

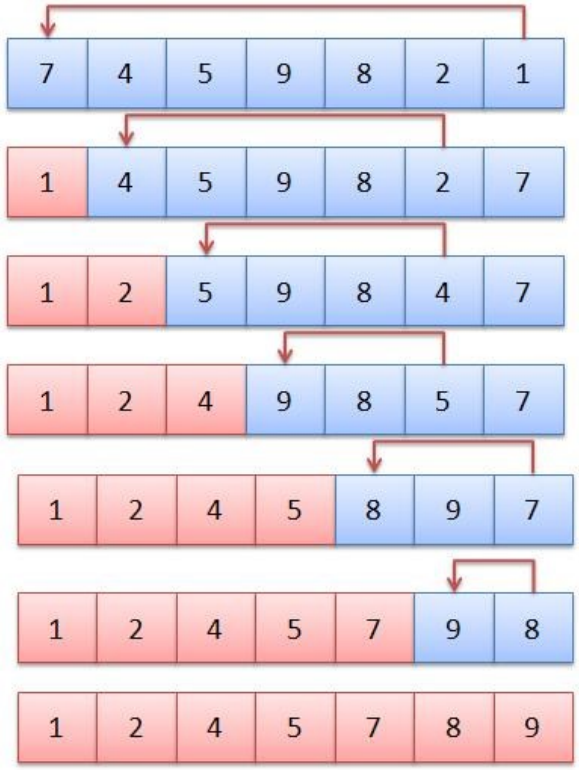
Сортиране с вмъкване

6 5 3 1 8 7 2 4

Бързо сортиране

6 5 3 1 8 7 2 4

Сортиране с пряка селекция



4. Сортиране и търсене

```
void mergesort(int arr[], int l, int r) {
    if (l < r) {                    // гранично условие на рекурсията
         int m = (l+r)/2;                // m – middle part
        mergesort(arr, l, m);          // l – left border, r – right border
        mergesort(arr , m+1, r);
        merge(arr, l, m, r);            // merge two arrays
     }
}
          //  Function to merge the subarrays of a[]
 void merge(int arr[], int beg, int middle, int end);
```

The important part of the merge sort is the **merge** function. This function performs the merging of two sorted sub-arrays that are **arr1[beg…mid]** and **arr2[mid+1…end]**, to build one sorted array **a[beg…end]**. So, the inputs of the **merge** function are **a[], beg, mid,** and **end**.

4. Сортиране и търсене

Бързо сортиране (Quick sort)

За едно произволно число от масива, с линейна сложност може да се прехвърлят всички по-малки числа от масива да са в ляво на числото, а всички по-големи в дясно от него. При бързото сортиране се избира число от масива, прехвърлят се по-малите отляво, по-големите от дясно и след това се изпълнява същата процедура за лявата и дясната половина. Получената сложност в средният случай е: O(n*log(n)).

Сортиране с броене (Counting sort)

Понеже има ограничен брой различни стойности в масива, то може да се преброи по колко пъти се среща всяка една от тези стойности (с едно обхождане на масива) и след това със второ обхождане да се наредят стойностите в техният ред.

Стабилност на сортирането - ако има два елемента които са равни, то във финалният те се срещат във същият ред като в първоначалният масив.

4. Сортиране и търсене

```
void quicksort(int arr[], int left, int right) {
    if (left < right) {
        int pi = partition(arr, left, right);  // calculate pivot index
        quicksort(arr, left, pi-1);
        quicksort(arr, pi+1, right);
    }
}
```

❖ В най-ранните версии на бързото сортиране често най-левият елемент е бил избиран за главен. Това не работи при вече сортирани масиви, които са доста често срещан случай. Проблемът бил лесно решен, като се избирал или произволен индекс за главен елемент, или средният елемент на дяла, или (особено за по-дълги редици) медианата на първия, средния и последния елемент на дяла (както препоръчвал Робърт Седжуик).

```
int partition(int arr[], int low, int high){
  int pivot = arr[high];
  int i = low; // index of smaller element
  for (int j=low; j<high; j++){
        if (arr[j] <= pivot){
              swap(arr, i, j)
            i++;
        }
    }
   swap(arr, i, high)
 return i;
}
```

Въпроси ?