



# 10. Алгоритми

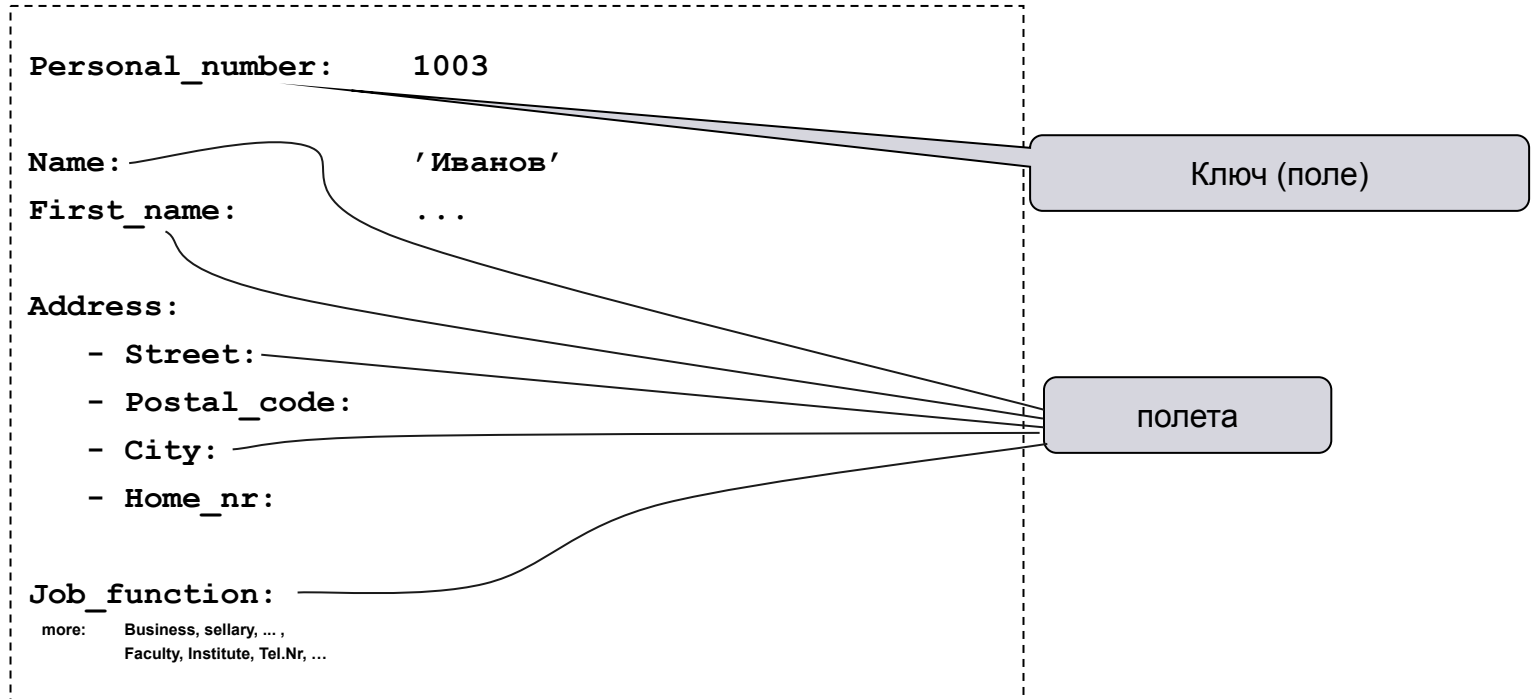
Програмиране на Java



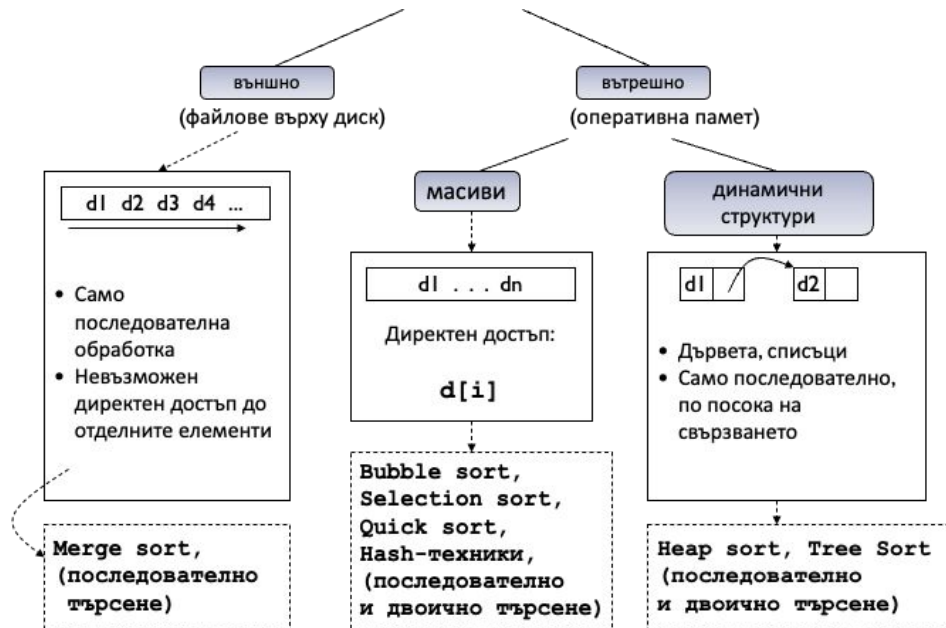
## Задача: търсене и сортиране

- Основна функционалност на много софтуерни системи е работа с данни
  - Съществуват специализирани системи за управление на бази от данни (СУБД)
- Основните функции при работа с данни са:
  - **Търсене**
  - **Сортиране** (опростява последващо търсене)
- **Ефективността** е от съществено значение
  - Определя се от комплексността на използвания алгоритъм и оказва влияние при:
    - Обемни структури данни (прим. жителите на София са над 2 милиона)
    - Комплексни структури от данни

# Комплексни структури



# Къде и как се съхраняват записите?





## Класове комплексност на алгоритмите

логаритмични: (power1, bin search)	$O(n) = k * \log_2 n$
линейни: (power, lin search)	$O(n) = k * n$
$n \log_2 n$ : (Quick sort, Merge sort, Heap sort)	$O(n) = k * n \log_2 n$
квадратичен: ( Selection sort, Bubble sort)	$O(n) = k * n^2$
полиномен:	$O(n) = k * n^m \quad (m > 1)$
експоненциален: (Hanoi)	$O(n) = k * 2^n$

Кратка нотация:  $O(f(n))$  за  $O(n) = k * f(n)$

напр. power1 е  $O(\log_2 n)$ , Selection sort е  $O(n^2)$



# Линейно търсене в масив

Несортиран масив


100	6	33	77	39	20	20	206	200
-----	---	----	----	----	----	----	-----	-----



Търсенето е последователно - претърсват се всички елементи за търсената стойност (напр. 39)

**Среден разход за търсене:**  $O(n) = \frac{1}{2} n$

Т.е. последователното търсене има линейна комплексност (проблем при голям обем данни - напр. 2 млн. жители)



## Пример: линейно търсене

Несортиран масив

```
public class LinearSearch {  
    public static void main(String[] args) {  
        int[] a = {10, 3, 8, 12, 34, 5, 1, 48};  
  
        int value = 12;  
        int pos = find(value, a);  
  
        if (pos >= 0) {  
            System.out.printf("Стойността %d е  
намерена на позиция %d", value, pos);  
        } else {  
            System.err.printf("Стойността %d не  
е намерена", value);  
        }  
    }  
  
    static int find(int value, int[] array) {  
        for(int i = 0; i < array.length; i++) {  
            if (array[i] == value) {  
                return i;  
            }  
        }  
  
        return -1;  
    }  
}
```



## Алтернативна имплементация

```
public class LinearSearch {  
    ...  
  
    static int find(int value, int[] array) {  
        for (int i = 0; array[i] != value && i < array.length; i++);  
        if (i < array.length) {  
            return i;  
        }  
        return -1;  
    }  
}
```

Има две грешки.  
Кои са те?





# Алтернативна имплементация

```
public class LinearSearch {  
    ...  
    static int find(int value, int[] array) {  
        int i;  
        for (i = 0; i < array.length && array[i] != value; i++);  
        if (i < array.length) {  
            return i;  
        }  
        return -1;  
    }  
}
```

Работеща  
имплементация

Удачен ли е видът  
цикъл тук?



## Алтернативна имплементация

```
public class LinearSearch {  
    ...  
    static int find(int value, int[] array) {  
        int i = 0;  
        while (i < array.length && array[i] != value) {  
            i++;  
        }  
        if (i < array.length) {  
            return i;  
        }  
        return -1;  
    }  
}
```

По-адекватен  
оператор за цикъл



## Двоично търсене

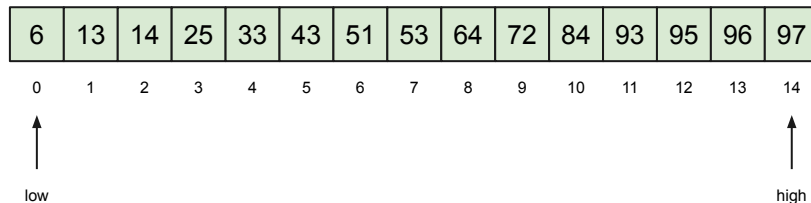
2	5	7	10	20	55	77	78	80	100	101
---	---	---	----	----	----	----	----	----	-----	-----

- Метод на разделянето
- Работи само при сортиран масив
- Алгоритъм: търсен елемент  $x = 80$ 
  - Сравняване на  $x$  със средния елемент на масива  $a[m]$  (в случая  $m = 5$ ), като при:
    - $x == a[m]$   $\Rightarrow$  елементът е намерен
    - $x > a[m]$   $\Rightarrow$  търсене в десния подмасив
    - $x < a[m]$   $\Rightarrow$  търсене в левия подмасив
  - Ако подмасивът е празен  $\Rightarrow$  елементът не е намерен

## Двоично търсене: алгоритъм

- Даден е key и сортиран масив  $a[]$
- Да се намери индекса  $i$ , за който  $a[i] == \text{key}$

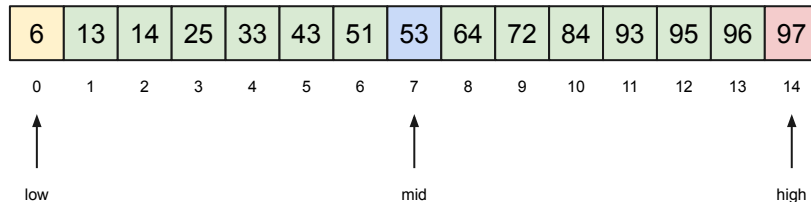
Пример:  $\text{key} = 33$



## Двоично търсене: алгоритъм

- Даден е `key` и сортиран масив `a[]`
- Да се намери индекса `i`, за който `a[i] == key`

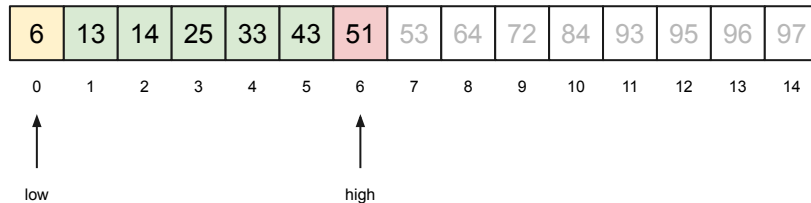
Пример: `key = 33`



## Двоично търсене: алгоритъм

- Даден е key и сортиран масив  $a[]$
- Да се намери индекса  $i$ , за който  $a[i] == \text{key}$

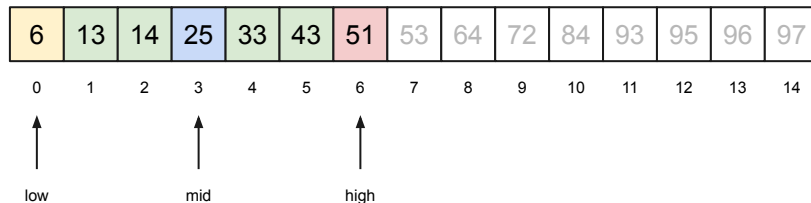
Пример:  $\text{key} = 33$



# Двоично търсене: алгоритъм

- Даден е key и сортиран масив  $a[]$
- Да се намери индекса  $i$ , за който  $a[i] == key$

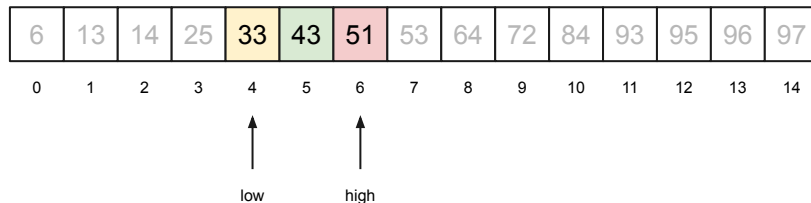
Пример: key = 33



## Двоично търсене: алгоритъм

- Даден е key и сортиран масив  $a[]$
- Да се намери индекса  $i$ , за който  $a[i] == \text{key}$

Пример:  $\text{key} = 33$

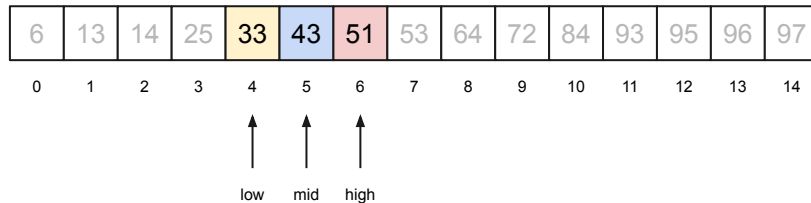




# Двоично търсене: алгоритъм

- Даден е key и сортиран масив  $a[]$
- Да се намери индекса  $i$ , за който  $a[i] == \text{key}$

Пример:  $\text{key} = 33$

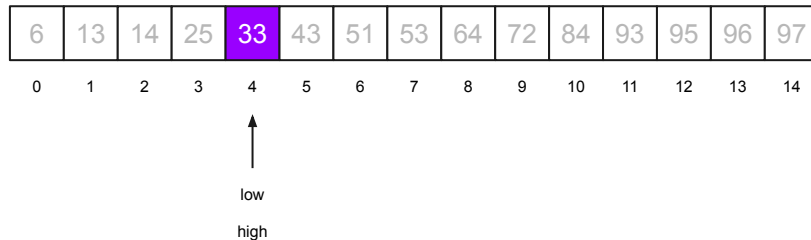




## Двоично търсене: алгоритъм

- Даден е key и сортиран масив  $a[]$
- Да се намери индекса  $i$ , за който  $a[i] == \text{key}$

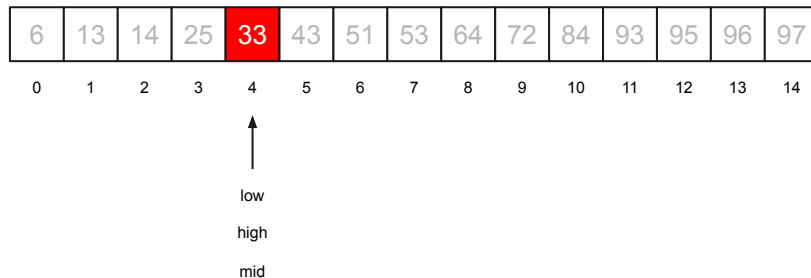
Пример:  $\text{key} = 33$



## Двоично търсене: алгоритъм

- Даден е `key` и сортиран масив `a[]`
- Да се намери индекса `i`, за който `a[i] == key`

Пример: `key = 33`





## Двоично търсене: комплексност

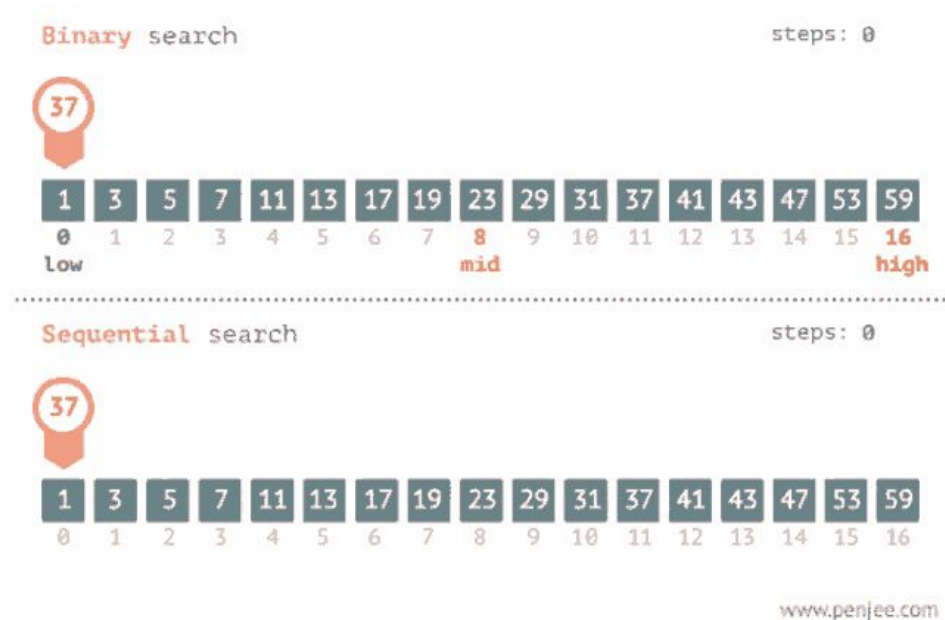
- Брой на разделянията - в най-лошият случай делим масива докато остане само един елемент
- Максимално  $\log_2 n$  стъпки




## Двоично vs линейно търсене

Брой	100	1024	1 Mio
Линейно (средно)	50	512	500.000
Двоично (максимално)	7	10	20

## Двоично vs линейно търсене





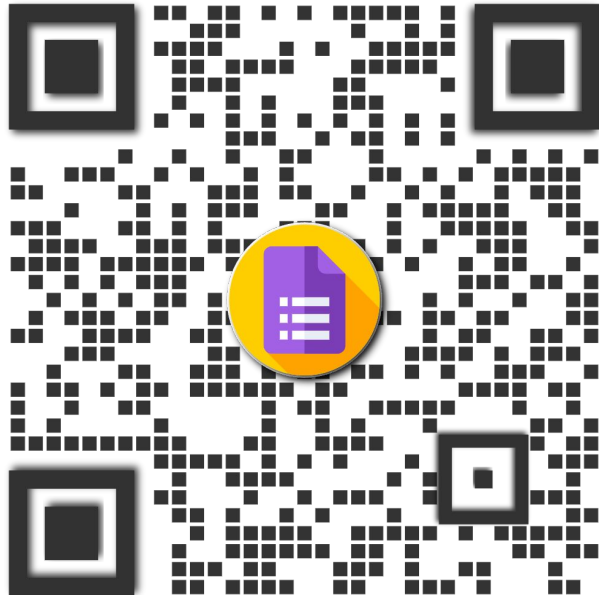
# Двоично търсене: имплементация

```
public class BinarySearch {  
    public static void main(String[] args) {  
        int[] a = {1, 3, 5, 8, 10, 12, 34, 48};  
  
        ...  
    }  
  
    static int binarySearch(int value, int[] array) {  
        int left = 0;  
        int right = array.length - 1;  
        int middle;  
  
        while (left <= right) {  
            middle = (left + right) / 2;  
  
            if (array[middle] == value) {  
                return middle;  
            }  
  
            if (array[middle] < value) {  
                left = middle + 1;  
            } else {  
                right = middle - 1;  
            }  
        }  
  
        return -1;  
    }  
}
```

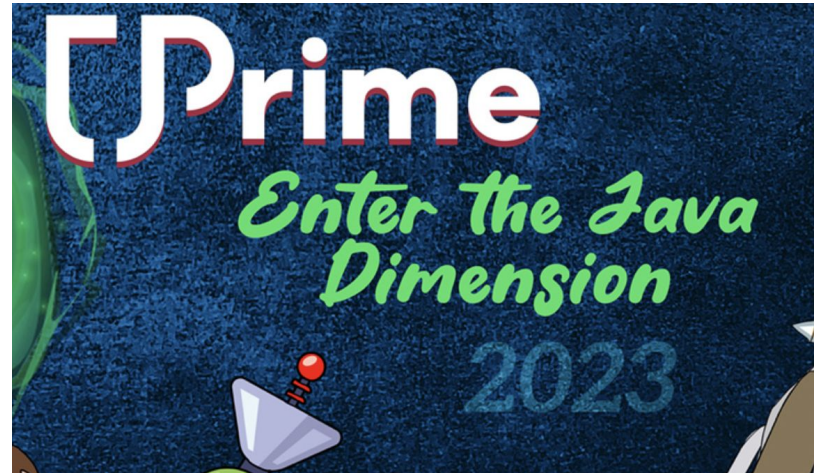


JP

# JPrime



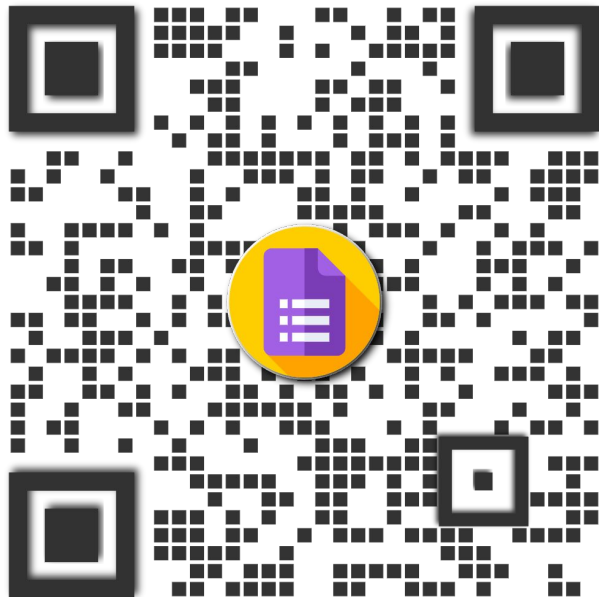
<https://t.ly/Vo3h>







## Регистриране на присъствие



**<https://t.ly/FxDA>**

Отговор: object