

Almost all really new ideas  
have a certain aspect of  
foolishness when they are  
first produced.  
*Alfred North Whitehead*

# Estruturas de Dados

## Ficha 2 - Pesquisa binária e variações

12 16 23 41 **63** **74** **85**  
start end

Engenharia Informática

3º ano, 1º semestre

1 – Construa um método que efetua uma **pesquisa binária** de forma **recursiva**, devolvendo **true** ou **false** conforme o valor seja encontrado ou não.

12	16	23	41	63	74	85
				start		end

2 – Construa um método que efectua uma pesquisa binária de forma **iterativa**, devolvendo **true** ou **false** conforme o valor seja encontrado ou não.

**Chave: 65**

**12 16 23 41 63 74 85**  
start end



**12 16 23 41 63 74 85**  
start end



**12 16 23 41 63 74 85**  
start end



**12 16 23 41 63 74 85**  
end start

3 - Construa um método que efetua uma pesquisa binária. Este método deve devolver a **posição** em que o valor procurado se encontra, ou então -1 caso este não esteja no array indicado. Teste o método, conforme o indicado nos exercícios anteriores.

4 - Construa um método que efetua uma pesquisa binária. Este método deve devolver a **posição** em que o valor procurado se encontra, ou então um valor negativo (-X) caso este não esteja no array indicado. O valor de  $|X+1|$  deve indicar uma posição em que o valor procurado poderia ser inserido para preservar a ordem. Teste o método, conforme o indicado nos exercícios anteriores.

**Exemplo para Array:** {3, 7, 12, 15}

3	7	12	15
0	1	2	3

Chave=15 resultado= 3

Chave=3 resultado= 0

Chave=1 resultado= -1 (Deveria ser inserido na posição 0)

Chave=4 resultado= -2 (o valor deveria ser inserido na posição 1)

Chave=10 resultado= -3 (o valor deveria ser inserido na posição 2)

3	7	<del>12</del> 10	12	15
0	1	2	2 3	3 4

5 - Construa um método que recebe um array ordenado de inteiros, todos diferentes, e um valor, e devolve a **percentagem** de valores do array que são menores do que o valor indicado. O algoritmo deve ser de complexidade logarítmica.

Exemplo:

**Array:** {3,7,12,15}

Valor =15      resultado= 0.75

Valor =14      resultado= 0.75

Valor =3      resultado= 0.0

Valor =1      resultado= 0.0

Valor =100      resultado= 1.0

3	7	12	15
0	1	2	3

6 – Construa um método que recebe um array ordenado de inteiros, todos diferentes, e dois valores que definem um **intervalo**. O método deve indicar quantos valores do array se encontram dentro do intervalo especificado.

O método deve ter complexidade logarítmica.

Exemplo:

**Array:** {3,7,12,15}

Valores =(0,15)    resultado= 4

Valores=(3,7)    resultado= 2

Valores =(4,14)    resultado= 2

Valores =(4,5)    resultado= 0

Valores =(0,100)    resultado= 4

3	7	12	15
0	1	2	3

*Multiplicar por uma constante não altera a ordem do algoritmo. Portanto, ao chamar duas vezes a pesquisa binária o algoritmo continua a ser de complexidade logaritmica.*

```
public static int contaIntervalo(int []tab,int min,int max){
    int j, k, count = 0;

    //    for(j=0;j<tab.length && tab[j]<=max;j++) // Solução linear
    //        if(tab[j]>=min && tab[j]<=max) count++;
    //    return count;

    j = binCmp(tab,min); // Solução logarítmica
    k = binCmp(tab,max); //  $O(2.\log(n))$  continua a ser logaritmico
    if(j<0) j = -j-1;
    if(k<0) k = -k-1; else k++;

    return k-j;
}
```

3	7	12	15
0	1	2	3



7 – Construa um método que recebe um *array* ordenado de inteiros, e um valor. O método deve indicar se esse valor se encontra repetido no *array*.

Exemplo:

**Array:** {3,3,7,12,12,15}

Valor =15      resultado= false

Valor =14      resultado= false

Valor =12      resultado= true

Valor =3      resultado= true

3	3	7	12	12	15
0	1	2	3	4	5

# Ficha 1 – Exercício 7

```
public static boolean testaRepetidos(int []tab,int chave){
```

```
    int j;
```

```
    j = pesquisaBinaria(tab,chave);
```

```
    if(j<0) return false; // Número não existe
```

```
    if(j==0) // Limite inferior da tabela, só se pode comparar o seguinte
```

```
        if(tab[1] != chave)
```

```
            return false;
```

```
        else return true;
```

```
    if(j==tab.length-1) // Limite superior da tabela, só se pode comparar o anterior
```

```
        if(tab[j-1] != chave)
```

```
            return false;
```

```
        else return true;
```

```
    if( (tab[j]==tab[j-1]) || (tab[j]==tab[j+1])) return true; // Ver vizinhos
```

```
    return false;
```

```
}
```

<b>3</b>	<b>5</b>	<b>7</b>	<b>7</b>	<b>15</b>
0	1	2	3	4

8 – Construa um método que recebe por parâmetro um array ordenado de inteiros, não repetidos, bem como um inteiro Z, e devolve o maior elemento do array menor do que Z (ou Z se esse elemento não existir).

Exemplo:

**Array:** {3,7,12,15}

Valor =15      resultado= 12

Valor =14      resultado= 12

Valor =3      resultado= 3

Valor =1      resultado= 1

Valor =100      resultado= 15

## Ficha 1 – Exercício 8

8 – Construa um método que recebe por parâmetro um array ordenado de inteiros, não repetidos, bem como um inteiro Z, e devolve o maior elemento do array menor do que Z (ou Z se esse elemento não existir).

Exemplo:

**Array:** {3,7,12,15}

Valor =15    resultado= 12

Valor =14    resultado= 12

Valor =3    resultado= 3

Valor =1    resultado= 1

Valor =100    resultado= 15

<b>3</b>	<b>7</b>	<b>12</b>	<b>15</b>
0	1	2	3

```
public static int procuraMenor(int []tab,int chave){
    int j, pos = binCmp(tab,chave);

    if( pos==-1 || pos==0) return chave;

    if(pos<0) pos = -pos-1;

    return tab[pos-1];
}
```

9 – Considere um método que recebe um array de inteiros no qual os números estão dispostos da seguinte forma: todos os números negativos se encontram em posições maiores do que os números positivos, e todos os números positivos e negativos se encontram ordenados entre si.

*Exemplo:* {3,6,8, -10,-3,-2,-1}

Construa um método que procura um número no array.

9 – *Exemplo*: {3,6,8, -10,-3,-1}

Se a chave e o `tab[meio]` têm o mesmo sinal, estamos no lado certo da tabela e é o algoritmo normal de pesquisa binária.

Se têm sinais contrários, então estamos no "lado errado" da tabela...

6					
3	6	8	-10	-3	-2
0	1	2	3	4	5

9 – Exemplo: {3,6,8, -10,-3,-1}

```
public static int pesquisaBinariaNeg(int []tab,int chave){
    int linf=0,lsup = tab.length-1,meio = tab.length/2;

    do{
        if(tab[meio]==chave) return meio;
        if( chave * tab[meio] >=0 ) // meio e chave têm mesmo sinal
            if(tab[meio]<chave) linf=meio+1;
            else lsup=meio-1;
        else //meio e chave têm sinais contrários - inverte as contas
            if(tab[meio]<chave) lsup=meio-1;
            else linf=meio+1;
        meio = (linf+lsup)/2;
    }while(linf<lsup);
    if(tab[meio]==chave) return meio;
    return -1;
}
```

6

3	6	8	-10	-3	-2
0	1	2	3	4	5

10 – Construa um método que recebe um array ordenado de inteiros, não repetidos, e que devolve o índice da primeira posição na qual o valor guardado é superior ao índice (ou -1 se esta posição não existir).

Exemplos:

**Array:** {3,7,12,15} Resultado=0

**Array:** {-3,1,7,12,15} Resultado=2

**Array:** {-15,-14,1,2,3,4} Resultado= -1



10 – Construa um método que recebe um array ordenado de inteiros, não repetidos, e que devolve o índice da primeira posição na qual o valor guardado é superior ao índice (ou -1 se esta posição não existir).

Exemplos:

**Array:** {3,7,12,15} **Resultado=0**

**Array:** {-3,1,7,12,15} **Resultado=2**

**Array:** {-15,-14,1,2,3,4} **Resultado= -1**

```
public static int maiorQueIndice(int []tab){
    int j;

    for(j=0; j<tab.length; j++) // Solução complexidade linear
        if(tab[j]>j) return j;
    return -1;
}
```

# Ficha 1 – Exercício 10

```
public static int maiorQueIndice(int []tab){
    int j, linf=0,lsup = tab.length-1,meio = tab.length/2;

    //    for(j=0;j<tab.length;j++) // Solução de complexidade linear
    //        if(tab[j]>j) return j;
    //    return -1;

    do{ // Solução de complexidade logaritimica
        System.out.println("Meio: "+meio+" linf "+linf+" lsup "+lsup);
        if(tab[meio]>meio)
            lsup = meio;
        else
            linf = meio+1;
        meio = (linf+lsup)/2;
    }while(linf<lsup);

    if(tab[meio]>meio) return meio;
    return -1;
}
```

