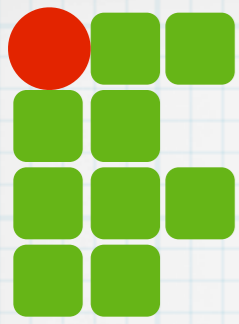


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Algoritmos

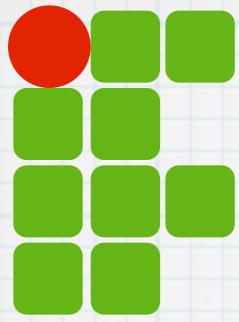
Divisão e conquista

Copyright © 2014 IFRN

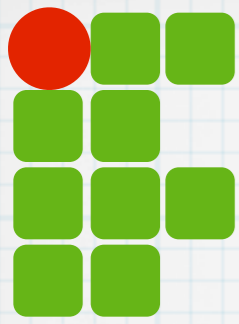


Agenda

- * Divisão e conquista
- * Exemplos
- * Exercícios



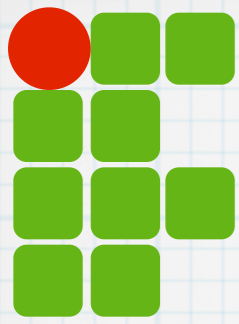
Introdução



Introdução

* Divisão

- * resolver recursivamente problemas menores (até caso base)



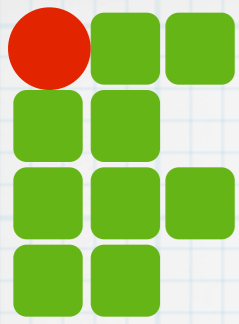
Introdução

* Divisão

- * resolver recursivamente problemas menores (até caso base)

* Conquista

- * solução do problema original é formada com as soluções dos subproblemas



Introdução

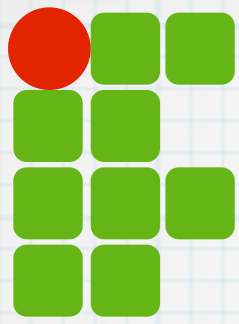
* Divisão

- * resolver recursivamente problemas menores (até caso base)

* Conquista

- * solução do problema original é formada com as soluções dos subproblemas

- * Há divisão quando o algoritmo tem pelo menos 2 chamadas recursivas no corpo



Introdução

* Divisão

- * resolver recursivamente problemas menores (até caso base)

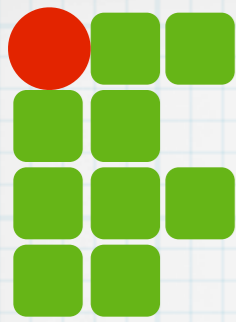
* Conquista

- * solução do problema original é formada com as soluções dos subproblemas

* Há divisão quando o algoritmo tem pelo menos 2 chamadas recursivas no corpo

* Subproblemas devem ser disjuntos

- * Senão, resolver de forma bottom-up com programação dinâmica



Introdução

* Divisão

- * resolver recursivamente problemas menores (até caso base)

* Conquista

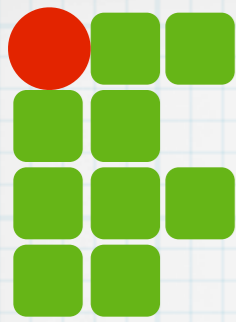
- * solução do problema original é formada com as soluções dos subproblemas

* Há divisão quando o algoritmo tem pelo menos 2 chamadas recursivas no corpo

* Subproblemas devem ser disjuntos

- * Senão, resolver de forma bottom-up com programação dinâmica

* Divisão em subproblemas de dimensão semelhante é importante para se obter uma boa eficiência temporal



Introdução

* Divisão

- * resolver recursivamente problemas menores (até caso base)

* Conquista

- * solução do problema original é formada com as soluções dos subproblemas

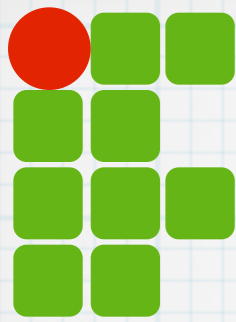
* Há divisão quando o algoritmo tem pelo menos 2 chamadas recursivas no corpo

* Subproblemas devem ser disjuntos

- * Senão, resolver de forma bottom-up com programação dinâmica

* Divisão em subproblemas de dimensão semelhante é importante para se obter uma boa eficiência temporal

* Algoritmos adequados para processamento paralelo



Exponencial

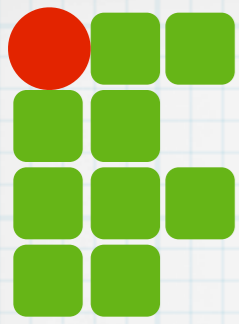
✓

$$\exp(a, n) \begin{cases} 1 & \text{Se } n = 0 \\ a & \text{Se } n = 1 \\ a^{\frac{n}{2}} \times a^{\frac{n}{2}} & \text{Se } n \text{ é par} \\ a \times a^{\frac{n-1}{2}} \times a^{\frac{n-1}{2}} & \text{Se } n \text{ é ímpar} \end{cases}$$

```
double exp(double a, int n) {
    if (n == 0)
        return 1;
    if (n == 1)
        return a;
    double p = 1;
    if (n % 2 == 1) {
        p = a;
        n = n - 1;
    }
    double r = exp(a, n/2);
    return p * r * r;
}
```

✓

```
double exp(double a, int n) {
    if (n == 0)
        return 1;
    if (n == 1)
        return x;
    double p = exp(x, n/2);
    if (n % 2 == 0)
        return p * p;
    else
        return a * p * p;
}
```



Exponencial

- * Divisão em subproblemas iguais, junção em tempo $O(1)$
- * Número de multiplicações reduzido
 - * $\text{Log}n \rightarrow O(\text{log}n)$
- * ... mas $S(n) = O(\log n)$ (Espaço)

$$2^{10} = 2^5 * 2^5$$



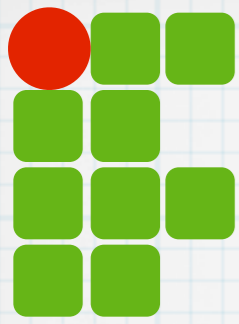
$$2 * 2^2 * 2^2$$



$$2^1 * 2^1$$

$$2^{10} = 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2$$

Quantas multiplicações, aproximadamente, são realizadas para calcular x^{100} ?

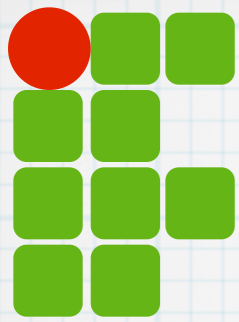


O problema da busca

- * Considere um array ordenados de n elementos
- * Qual a complexidade de uma função que retorne o índice de um elemento x no array (-1 se não estiver no array)?

1	3	7	19	21	22	90	121	131	132	201	241	261	312	432	526	566	712	812	901	902
---	---	---	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

```
int busca(int *a, int size, int x) {  
    int indice = -1;  
    int i;  
    for (i=0 ; i < size ; i++){  
        if (a[i]==x){  
            indice = i;  
            break;  
        }  
    }  
    return indice;  
}
```



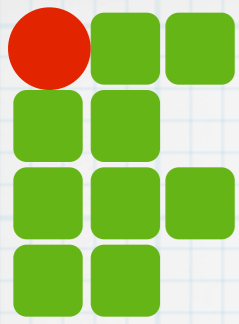
O problema da busca

- * Considere um array ordenados de n elementos
- * Qual a complexidade de uma função que retorne o índice de um elemento x no array (-1 se não estiver no array)?

1	3	7	19	21	22	90	121	131	132	201	241	261	312	432	526	566	712	812	901	902
---	---	---	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

```
int busca(int *a, int size, int x) {  
    int indice = -1;  
    int i;  
    for (i=0 ; i < size ; i++){  
        if (a[i]==x){  
            indice = i;  
            break;  
        }  
    }  
    return indice;  
}
```

Algoritmo
executa em
tempo $O(n)$, no
pior caso



O problema da busca

* Divisão e conquista

- * Array ordenado

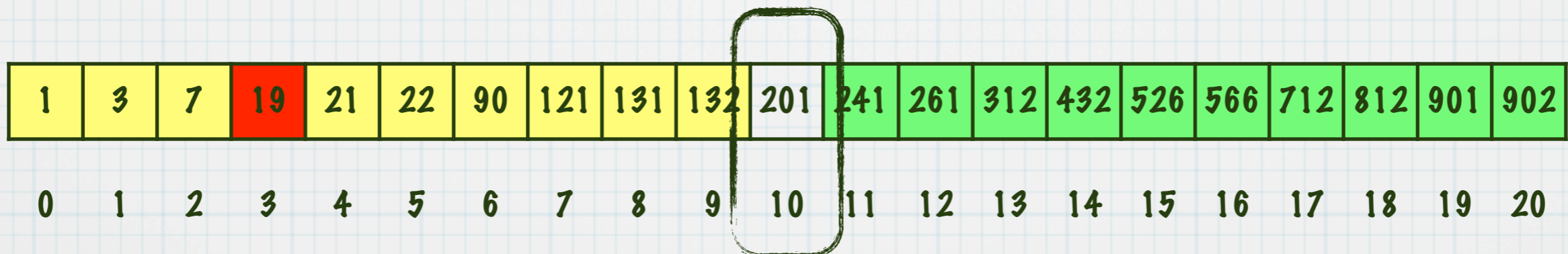
- * Verifica se elemento está no meio do array

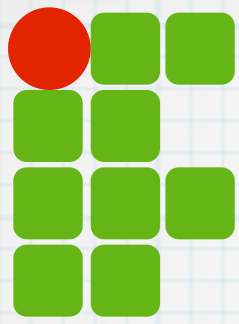
- * Como o array está ordenado:

- * Se for menor que o do meio, está a esquerda

- * Se for maior está a direita

* Busca do elemento 19

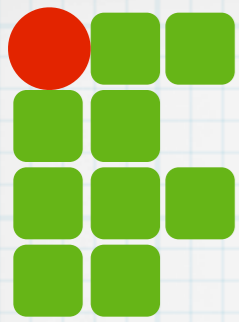




O problema da busca

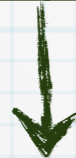
1	3	7	19	21	22	90	121	131	132	201	241	261	312	432	526	566	712	812	901	902
---	---	---	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

19 é menor do
que 201



O problema da busca

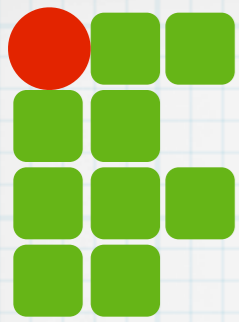
1	3	7	19	21	22	90	121	131	132	201	241	261	312	432	526	566	712	812	901	902
---	---	---	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



19 é menor do
que 201

1	3	7	19	21	22	90	121	131	132
---	---	---	----	----	----	----	-----	-----	-----

19 é menor
do que 20



O problema da busca

1	3	7	19	21	22	90	121	131	132	201	241	261	312	432	526	566	712	812	901	902
---	---	---	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

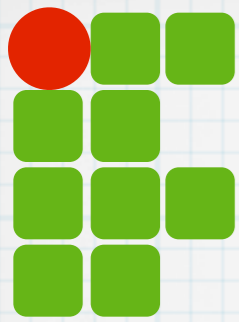
19 é menor do que 201

1	3	7	19	21	22	90	121	131	132
---	---	---	----	----	----	----	-----	-----	-----

19 é menor do que 20

1	3	7	19
---	---	---	----

19 é maior do que 3



O problema da busca

1	3	7	19	21	22	90	121	131	132	201	241	261	312	432	526	566	712	812	901	902
---	---	---	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

19 é menor do que 201

1	3	7	19	21	22	90	121	131	132
---	---	---	----	----	----	----	-----	-----	-----

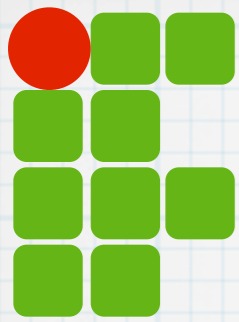
19 é menor do que 20

1	3	7	19
---	---	---	----

19 é maior do que 3

7	19
---	----

19 é maior do que 7



O problema da busca

1	3	7	19	21	22	90	121	131	132	201	241	261	312	432	526	566	712	812	901	902
---	---	---	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

19 é menor do que 201

1	3	7	19	21	22	90	121	131	132
---	---	---	----	----	----	----	-----	-----	-----

19 é menor do que 20

1	3	7	19
---	---	---	----

19 é maior do que 3

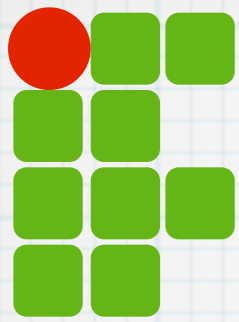
7	19
---	----

19 é maior do que 7

19

A cada passo o array é dividido em 2

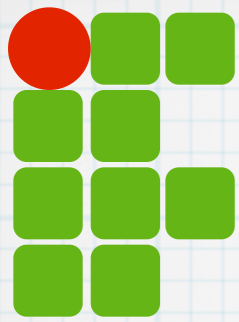
No pior caso, o tempo é $O(\log n)$



O problema da busca

```
int buscaRec(int *a, int x, int inicio, int fim) {
    int meio = 0;
    if (inicio > fim) {
        return 0;
    } else {
        meio = (inicio + fim) / 2;
    }
    if (x == a[meio]) {
        return meio;
    } else {
        if (x < a[meio]) {
            return buscaRec(a, x, inicio, meio - 1);
        } else {
            return buscaRec(a, x, meio + 1, fim);
        }
    }
}

int busca(int *a, int size, int x) {
    return buscaRec(a, x, 0, size - 1);
}
```

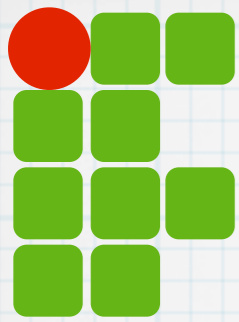


O problema da busca

Final da
busca

```
int buscaRec(int *a, int x, int inicio, int fim) {
    int meio = 0;
    if (inicio > fim) {
        return 0;
    } else {
        meio = (inicio + fim) / 2;
    }
    if (x == a[meio]) {
        return meio;
    } else {
        if (x < a[meio]) {
            return buscaRec(a, x, inicio, meio - 1);
        } else {
            return buscaRec(a, x, meio + 1, fim);
        }
    }
}

int busca(int *a, int size, int x) {
    return buscaRec(a, x, 0, size - 1);
}
```



O problema da busca

Final da
busca

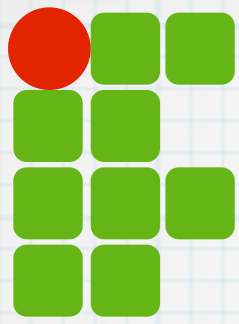
```
int buscaRec(int *a, int x, int inicio, int fim) {  
    int meio = 0;  
    if (inicio > fim) {  
        return 0;  
    } else {  
        meio = (inicio + fim) / 2;  
    }
```

```
    if (x == a[meio]) {  
        return meio;  
    } else {
```

```
        if (x < a[meio]) {  
            return buscaRec(a, x, inicio, meio - 1);  
        } else {  
            return buscaRec(a, x, meio + 1, fim);  
        }  
    }
```

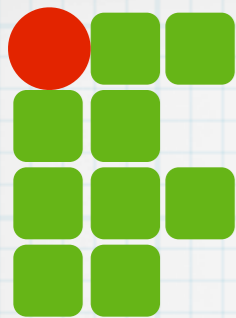
Busca
recursiva

```
int busca(int *a, int size, int x) {  
    return buscaRec(a, x, 0, size - 1);  
}
```



Conclusão

- * Melhorar desempenho temporal
- * Algoritmos passíveis de se conseguir subdividir
- * Recursividade



Dúvidas?