

>> Divisão e Conquista

-Observamos que um algoritmo pode invocar outro algoritmo dentro dele mesmo, o que pode afetar fortemente no cálculo de passos. Para calcular isso, utilizaremos a técnica de divisão e conquista, subdividida em três fases:

1. Dividir: Trabalhar sobre a instância do problema para gerar sub-instâncias menores.

2. Conquistar: Resolver o problema para as sub-instâncias construídas. Normalmente é um trabalho recursivo.

3. Combinar: Trabalhar sobre as soluções obtidas para as sub-instâncias de forma a construir a solução procurada.

Problema: Contar o número de identidades em um vetor numérico. Uma identidade é um índice i tal que $V[i] = i$.

```
contador ← 0
para i ← 1,2,...|V| faça
| se  $V[i] = i$  então
| | contador ← contador + 1
retorne contador
```

*O algoritmo está certo, mas não acontece divisão e conquista.

Algoritmo: IdentidadesDC(V, p, q)
Entrada: Vetor numérico V , índices p e q
Saída: Número de identidades em $V[p...q]$

```
se  $p > q$  então:                                +1
| retorne 0
se  $p = q$  então:                                  +1
| retorne  $V[p] = p$ 
meio ← piso( $(p + q) / 2$ )                          +2
 $c1 \leftarrow \text{IdentidadesDC}(V, p, \text{meio}) + Fp(\text{piso}(n / 2))$ 
 $c2 \leftarrow \text{IdentidadesDC}(V, \text{meio}+1, q) + Fp(\text{teto}(n / 2))$ 
retorne  $c1 + c2$                                     +2
```

Total de passos
 $6 + Fp(\text{piso}(n / 2)) + Fp(\text{teto}(n / 2))$

* Fp : Representação do valor dessa recursão que vamos calcular.

*Agora sim. Podemos calcular os passos através de divisão e conquista.

Complexidade de Algoritmos Recursivos

Calculamos $F(n)$ = divisão + conquista + combinação --Valor total da iteração

$$Fp(n) = \begin{cases} 6 + Fp(\text{ piso}(n/2) + Fp(\text{ teto}(n/2)) & , n > 1 \\ 2 & , n = 0 \\ 3 & , n = 1 \end{cases}$$

$$\begin{aligned} Fp(n) &= 6 + Fp(\text{ piso}(n/2) + Fp(\text{ teto}(n/2)) \\ &= 6 + Fp(n/2) + Fp(n/2) \quad \text{--supondo que } n/2 \text{ é potencia de 2 pra facilitar o} \\ &\quad \text{calculado, afinal, buscamos apenas um valor aproximado} \\ &= 6 + 2Fp(n/2) \\ &= 6 + 2(6 + 2Fp(n/2^2)) \quad \text{-- +1 recursão} \\ &= 6 + 2 * 6 + 2^2(6 + 2Fp(n/2^3)) \quad \text{-- +1 recursão} \\ &= 6 + 2 * 6 + 2^2 * 6 + 2^3(6 + 2Fp(n/2^4)) \quad \text{-- +1 recursão} \\ &= (\text{ Somatório } (i = 0 \text{ a } k-1) \text{ de } 2^i * 6) + 2^k * Fp(n/2^k) \\ &* n/2^k = 1 \text{ (vai chegar em 1)} \rightarrow n = 2^k \rightarrow k = \lg n \end{aligned}$$

$$\begin{aligned} Fp(n) &= (\text{ Somatório } (i = 0 \text{ a } \lg n-1) \text{ de } 2^i * 6) + 2^{\lg n} * 3 \quad \text{--o 3 veio do numero de} \\ &\quad \text{passos do ultimo caso.} \\ &= 6 * (\text{Somatorio}(i=0 \text{ a } \lg n-1) \text{ de } 2^i) + n * 3 \quad \text{--lembrando que } 2^{\lg n} = 2^{\log_2 2} = 1 \\ &= 6 * 2^{\lg n} + 3n \\ &= 6n + 3n \\ &= \Theta(n) \end{aligned}$$

$MS(V, p, q)$

```
Se p < q entao:          --+1
| meio ← piso((p + q) / 2)  --+1
| MS(V, p, meio)           --Fp(?)
| MS(V, meio+1, q)         --Fp(?)
| Entrelaçamento(V, p, m, q) --Fp(n) ≤ c*n
```

Usando o método de árvore de recursão para recorrências nesse algoritmo MS:

nível	$Fp(n)$	
0	$Fp(n/2) + Fp(n/2)$	$+ c*n$
1	$Fp(n/4) + Fp(n/4) + Fp(n/4) + Fp(n/4)$	$+ c*n/2 + c*n/2$
2	$8 * Fp(n/8)$	$+ c*n$
3	$16 * Fp(n/16)$	$+ c*n$
4	.	
5	.	
6	.	
k	$2^k * F(1)$	-- nao tem $c*n$, ou seja, ele vai acontecer $k-1$ vezes
	----> $n/2^k = 1 \rightarrow k = \lg n$	

$$\begin{aligned} Fp(n) &= (k - 1)c * n + 2^k * F(1) \\ Fp(n) &= (\lg n - 1)c * n + n * 1 \\ Fp(n) &= c * n * \lg n - c * n + n \\ &= c * n * \lg n - n * (1 - c) = \Theta(n * \lg n) \end{aligned}$$

Exemplo: $2 * T(n / 3) + n$

Usando o método de para recorrências:

Nível	#Termo	Recorrência	Não-recorrência
1	2^1	$T(n/3)+T(n/3)$	$+n*(2/3)^0$
2	2^2	$2*T(n/9)+2*T(n/9)$	$+n*(2/3)^1$
3	2^3	$8*T(n/27)$	$+n*(2/3)^2$
4	2^4	$16*T(n/81)$	$+n*(2/3)^3$
.			
.			
.			
k	2^k	$2^k*T(1)$	$+n*(2/3)^{k-1}$

$$\begin{aligned}
 T(n) &= 2^k * c + \text{Somatório}(i = 0, k-1) n * (2/3)^i \\
 T(n) &= c * 2^{(\log_3 n)} + n * \text{Somatório}(i = 0, \log_3 n - 1) n * (2/3)^i \\
 &\leq c * 2^{(\log_3 n)} + n * \text{Somatório}(i = 0, \infty) n * (2/3)^i \\
 &\leq c * 2^{(\log_3 n)} + n * 3 \\
 &\leq c * (3^{\log_3 n})^{(\log_3 2)} + n * 3 \\
 &\leq c * (3^{\log_3 n})^{(\log_2 3)} + n * 3 \\
 &\leq c * n^{(\log_3 2)} + n * 3 \\
 &\leq \Theta(n)
 \end{aligned}$$

Suponhamos o seguinte vetor para QuickSort

| | | | | |
 '-----> neste ultimo ficando o pivô, vimos que seria $\Theta(n^2)$

Supondo um outro dessa forma

| | |99%| | |1%| -- Aqui, temos que 1% do vetor estara a direita, 99% a esquerda
 '-----> Pivô

Temos então:

$$\begin{aligned}
 Fp(n) &= \{ Fp((99/100)*n) + Fp(n/100) + c*n \\
 &\quad \{ 1, \text{ caso base } (n=1)
 \end{aligned}$$

Analisando com arvore de recorrência:

Nível	Recorrência	
0	$Fp(n)$	+ 0
1	$Fp(99/100) + Fp(n/100)$	+ $c*n$
2	$Fp((99/100)^2*n) + Fp((99/100^2)*n) + Fp(99/100^2*n) + Fp(n/100^2)$	+ $2c*n$
.	.	.
.	.	.
.	.	.
k	$2^k * Fp(1)$	+ $k * c * n$

neste caso, como uma parte da recorrência terminara depois, e estamos buscando saber o valor de O, vamos calcular o O como se a arvore terminasse perfeita onde toda recorrência iria ate o nível da recorrência mais baixa.

tendo que: $1 = n / (100/99)^k \rightarrow n = (100/99)^k \rightarrow k = \log_{(100/99)}(n)$ $j = (100/99)$

$$\begin{aligned}
 \text{calculando: } 2^k * Fp(1) + k * c * n &= 2^{(\log_j n)} + (\log_j^n * c * n) \\
 &= ((100/99)^{\log_j^2})^{\log_j^n} + \log_j^n * c * n \\
 &= n^{\log_j^n} + c * n * \log_j^n \\
 &= O(n * \log n)
 \end{aligned}$$

TEOREMA MESTRE

Dada uma recorrência $T(n)$ da forma:

$$T(n) = a * T(n / b) + \Theta(n^c) \quad , \quad n \geq n_0$$
$$k \rightarrow \text{Caso base (Uma constante)} \quad , \quad n < n_0$$

Com a, b, c, n_0, k constantes, $b \neq 0$, temos:

- (1) $T(n) = \Theta(n^{\log_b a})$, se $\log_b a > c$
- (2) $T(n) = \Theta(n^c)$, se $\log_b a < c$
- (3) $T(n) = \Theta(n^c * \log n)$, se $\log_b a = c$

Exemplo:

$$Bp(n) = \begin{cases} \Theta(n^0) + 1 * Bp(n / 2) \\ k \end{cases}$$

$$a = 1, b = 2, c = 0$$

$$\log_2^1 = 0$$

$$Bp(n) = \Theta(\lg n)$$

O que ter em mente para gerar um algoritmo de Divisão e Conquista?

1. É possível responder o problema em termo das respostas para sub-instâncias?
 2. É "fácil" gerar sub-instâncias?
 3. É "fácil" combinar as respostas das sub-instâncias na resposta buscada?
-

POTENCIAÇÃO COM EXPOENTE NATURAL

Calcular x^n , onde x pertence aos reais e n aos naturais.
(Supondo que x e n não são nulos simultaneamente)

$$x^n = \begin{cases} x^{(n/2)} * x^{(n/2)} & , \text{ se } n \text{ é par} \\ x^{((n-1)/2)} * x^{((n-1)/2)} * x & , \text{ se } n \text{ é ímpar} \\ x^1 = x \\ x^0 = 1 \end{cases}$$

Algoritmo: PotNatural(x, n)

Entrada: x pertence aos reais e n aos naturais

Saída: valor de x^n

Requisito: $x \neq 0 \Leftrightarrow n \neq 0$

se $n = 0$ então:

| retorne 1

se n é par então:

| pot \leftarrow PotNatural($x, n/2$)

| retorne pot * pot

senão:

| pot \leftarrow PotNatural($x, (n - 1) / 2$)

| retorne pot * pot * x

$$Pp(n) = \begin{cases} Pp(n / 2) + \Theta(1) & \rightarrow c = 0, b = 2, a = 1 \rightarrow \log_2^1 = 0 \rightarrow Pp(n) = \Theta(n * \lg n) \\ k \end{cases}$$