

Técnicas de Desenvolvimento de Algoritmos (parte 2)

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados 2 (AE43CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

- Divisão e Conquista
 - Exemplo 1: encontrar o maior valor
 - Exemplo 2: potenciação
 - Exemplo 3: problema da mochila

Divisão e Conquista

- Ideia básica
 - **Divisão**: dividir o problema a ser resolvido em subproblemas menores e independentes
 - **Conquista**: encontrar soluções para as partes recursivamente
 - **Combinação**: combinar as soluções obtidas em uma solução global

Divisao_e_Conquista(x)

```
1: if  $x$  é pequeno ou simples then  
2:   return resolver( $x$ )  
3: else  
4:   decompor  $x$  em conjuntos menores  $x_0, x_1, \dots, x_n$   
5:   for ( $i \leftarrow 0$  até  $n$ ) do  
6:      $y_i \leftarrow \text{Divisao\_e\_Conquista}(x_i)$   
7:   end for  
8:   combinar  $y_i$ 's  
9: end if  
10: return  $y$ 
```

Divisão e Conquista

Exemplo 1: encontrar o maior valor

- O problema consiste em encontrar o maior elemento de um array $A[1..n]$

Solução Ingênua

```
1:  $max \leftarrow A[1]$ 
2: for  $i \leftarrow 2$  até  $n$  do
3:   if  $A[i] > max$  then
4:      $max \leftarrow A[i]$ 
5:   end if
6: end for
7: return  $max$ 
```

- Complexidade de tempo e de espaço: $\Theta(n)$

Divisão e Conquista

Exemplo 1: encontrar o maior valor

- Solução por divisão e conquista

Maxim(A, x, y)

```
1: if  $y - x \leq 1$  then  
2:   return  $\max(A[x], A[y])$   
3: else  
4:    $m \leftarrow (x + y)/2$   
5:    $v1 \leftarrow \text{Maxim}(A[x..m])$   
6:    $v2 \leftarrow \text{Maxim}(A[m + 1..y])$   
7: end if  
8: return  $\max(v1, v2)$ 
```

Divisão e Conquista

Exemplo 1: encontrar o maior valor

- A complexidade de tempo $T(n)$ do algoritmo é uma fórmula de recorrência
 - $T(n) = c$, para $n \leq 2$, onde c é uma constante
 - $T(n) = 2T(n/2) + c$ para $n > 2$
- A complexidade de espaço $T(n)$ do algoritmo também é uma fórmula de recorrência
 - $T(n) = n + c$, para $n \leq 2$ (já que o vetor possui tamanho n)
 - $T(n) = 2T(n/2) + c$ para $n > 2$
- Logo, por meio do método mestre (caso 1), para a função foi definida a complexidade de tempo e de espaço na ordem $\Theta(n)$

Divisão e Conquista

Exemplo 2: potenciação

- Solução Ingênua

potencia(a, n)

```
1:  $p \leftarrow a$   
2: for  $i \leftarrow 2$  até  $n$  do  
3:    $p \leftarrow p \times a$   
4: end for  
5: return  $p$ 
```

- Complexidade de tempo: $O(n)$
- Complexidade de espaço: $\Theta(1)$

Divisão e Conquista

Exemplo 2: potenciação

- Solução divisão e conquista

potencia(a, n)

```
1: if  $n = 0$  then  
2:   return 1  
3: else if  $n = 1$  then  
4:   return  $a$   
5: else  
6:   if  $n$  é par then  
7:      $x \leftarrow potencia(a, n/2)$   
8:     return  $x * x$   
9:   else  
10:     $x \leftarrow potencia(a, (n - 1)/2)$   
11:    return  $x * x * a$   
12:  end if  
13: end if
```

Divisão e Conquista

Exemplo 2: potenciação

- A complexidade $T(n)$ do algoritmo é uma fórmula de recorrência, tanto para a análise de tempo quanto de espaço
 - $T(0) = T(1) = c$
 - $T(n) = T(n/2) + c$ para $n > 1$
- Logo, por meio do método mestre (caso 2), para a função foi definida a complexidade de $\Theta(\log n)$
- Pior caso: $O(\log n)$

Divisão e Conquista

Exemplo 3: problema da mochila

- Se o vetor tiver o tamanho igual a 1, a função verifica se o peso do item não irá extrapolar a capacidade da mochila
 - Se a capacidade não for extrapolada, subtraia-a com o peso do item, já que o mesmo será adicionado na mochila e, em seguida, retorne o custo do item
 - Caso contrário, apenas retorne 0

Divisão e Conquista

Exemplo 3: problema da mochila

- Caso o tamanho do vetor seja maior que 1
 - 1 **Divisão**: divida o vetor ao meio
 - 2 **Conquista 1**: tente incluir, na mochila, um objeto da primeira metade recursivamente
 - 3 **Conquista 2**: tente incluir, na mochila, um objeto da primeira metade recursivamente
 - 4 **Combinação**: some o resultado das duas metades

Divisão e Conquista

Exemplo 3: problema da mochila

- Solução divisão e conquista

mochilaDQ(P, C, b, i, f)

```
1: if  $i = f$  and  $b - P[i] \geq 0$  then  
2:    $b \leftarrow b - P[i]$   
3:   return  $C[i]$   
4: else if  $i = f$  and  $b - P[i] < 0$  then  
5:   return 0  
6: else  
7:    $m \leftarrow (i + f)/2$   
8:   return  
        $mochilaDQ(P, C, b, i, m) + mochilaDQ(P, C, b, m + 1, f)$   
9: end if
```





Divisão e Conquista

Exemplo 3: problema da mochila

- A complexidade de tempo do algoritmo é uma fórmula de recorrência
 - $T(1) = c$
 - $T(n) = 2T(n/2) + c$ para $n > 1$
- A complexidade de espaço
 - $T(1) = n + c$
 - $T(n) = 2T(n/2) + c$ para $n > 1$
- Logo, por meio do método mestre (caso 1), para a função foi definida a complexidade de $\Theta(n)$
- **Aviso:** a solução do problema da mochila utilizando divisão e conquista apresentada em aula pode não gerar solução ótima!

- Outros exemplos de aplicação:
 - Ordenação por intercalação (*mergesort*)
 - Distância Euclidiana
 - Busca binária
 - Mediana
 - Quicksort

- Vantagens:
 - Altamente paralelizáveis
 - Fácil implementação
 - Simplificação de problemas complexos
- Desvantagens:
 - Necessidade de memória auxiliar
 - Repetição de subproblemas

-  Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Clifford, S.
Algoritmos: teoria e prática.
Elsevier, 2012.
-  Horowitz, E., Sahni, S. Rajasekaran, S.
Computer Algorithms.
Computer Science Press, 1998.
-  Szwarcfiter, J.; Markenzon, L.
Estruturas de Dados e Seus Algoritmos.
LTC, 2010.
-  Ziviani, N.
Projeto de Algoritmos - com implementações em Java e C++.
Thomson, 2007.

- Se o vetor tiver o tamanho igual a 1, a função apenas retorna o elemento
- Caso o tamanho do vetor seja maior que 1
 - 1 **Divisão**: divida o vetor ao meio
 - 2 **Conquista 1**: ordene a primeira metade recursivamente
 - 3 **Conquista 2**: ordene a segunda metade recursivamente
 - 4 **Combinação**: intercale as duas metades

Mergesort

mergesort(A, p, r)

```
1: if  $p < r$  then  
2:    $q = (p + r) / 2$   
3:   mergesort( $A, p, q$ )  
4:   mergesort( $A, q + 1, r$ )  
5:   merge( $A, p, q, r$ )  
6: end if
```

Mergesort

merge(A, p, q, r)

```
1: for i = p to q do
2:   B[i] = A[i]
3: end for
4: for j = q + 1 to r do
5:   C[j - q] = A[j] /*{Se for uma linguagem em que o primeiro elemento de
      vetores é acessado na posição 0, então C[j - q - 1] = A[j]}*/
6: end for
7: i = p
8: j = q
9: for k = p to r do
10:  if B[i] ≤ C[j] then
11:    A[k] = B[i]
12:    i = i + 1
13:  else
14:    A[k] = C[j]
15:    j = j + 1
16:  end if
17: end for
```

```
mergesort(A, p, r)
```

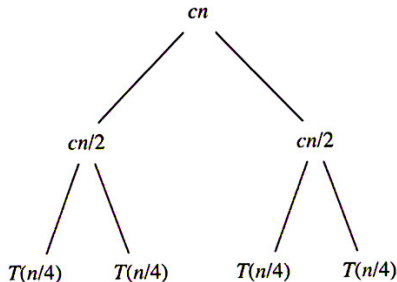
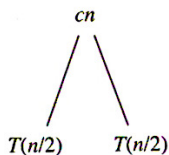
```
1: if  $p < r$  then  
2:    $q = (p + r) / 2$   
3:   mergesort(A, p, q)  
4:   mergesort(A, q + 1, r)  
5:   merge(A, p, q, r)  
6: end if
```

- A complexidade $T(n)$ do algoritmo é uma fórmula de recorrência
 - $T(1) = c$
 - $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn$ para $n > 1$
- Logo, a complexidade do *mergesort* é $\Theta(n \log n)$
- Pior caso: $O(n \log n)$

Mergesort

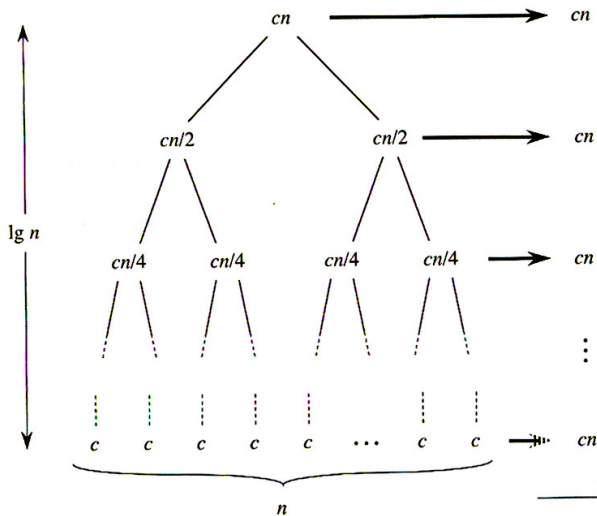
- Construção da árvore de recursão da recorrência do Mergesort
 $T(n) = 2T(n/2) + cn$

$T(n)$



Mergesort

- Custo total: $cn \log n + cn = \Theta(n \log n)$



Funcionamento do Mergesort

- Entrada

A	p				q			r	
	22	33	55	77	99	11	44	66	88

- Saída

A	p				q			r	
	11	22	33	44	55	66	77	88	99

Funcionamento do Mergesort

	p		q				r		
A	66	33	55	44	99	11	77	22	88

Funcionamento do Mergesort

	p		q				r		
A	66	33	55	44	99	11	77	22	88

	p		q		r			
A	66	33	55	44	99			

Funcionamento do Mergesort

	p				q			r	
A	66	33	55	44	99	11	77	22	88

	p		q		r				
A	66	33	55	44	99				

	p	q	r						
A	66	33	55						

Funcionamento do Mergesort

	p			q			r		
A	66	33	55	44	99	11	77	22	88

	p		q		r			
A	66	33	55	44	99			

	p	q	r					
A	66	33	55					

	p	r						
A	66	33						

Funcionamento do Mergesort

	p				q			r	
A	66	33	55	44	99	11	77	22	88

	p		q		r			
A	66	33	55	44	99			

	p	q	r					
A	66	33	55					

	p	r						
A	66	33						

	p = r							
A	66							

Funcionamento do Mergesort

	p				q			r	
A	66	33	55	44	99	11	77	22	88

	p		q		r				
A	66	33	55	44	99				

	p	q	r						
A	66	33	55						

	p	r							
A	66	33							

	p = r								
A		33							

Funcionamento do Mergesort

	p			q			r		
A	33	66	55	44	99	11	77	22	88

A

p		q		r			
33	66	55	44	99			

A

p	q	r					
33	66	55					

A

p	r						
33	66						

Funcionamento do Mergesort

	p				q			r	
A	33	66	55	44	99	11	77	22	88

A

	p		q		r				
	33	66	55	44	99				

A

	p	q	r						
	33	66	55						

A

			p = r						
			55						

Funcionamento do Mergesort

	p				q			r	
A	33	66	55	44	99	11	77	22	88

	p		q		r				
A	33	66	55	44	99				

	p	q	r						
A	33	66	55						

Funcionamento do Mergesort

	p				q			r	
A	33	55	66	44	99	11	77	22	88

	p		q		r				
A	33	55	66	44	99				

	p	q	r						
A	33	55	66						

Funcionamento do Mergesort

	p			q			r		
A	33	55	66	44	99	11	77	22	88

A

p		q		r			
33	55	66	44	99			

A

			p	r			
			44	99			

Funcionamento do Mergesort

	p			q			r		
A	33	55	66	44	99	11	77	22	88

A

p		q		r			
33	55	66	44	99			

A

			p	r			
			44	99			

Funcionamento do Mergesort

	p				q			r	
A	33	55	66	44	99	11	77	22	88

A

	p		q		r				
	33	55	66	44	99				

A

			p	r					
			44	99					

A

			p = r						
			44						

Funcionamento do Mergesort

	p				q			r	
A	33	55	66	44	99	11	77	22	88

A

	p		q		r				
	33	55	66	44	99				

A

			p	r					
			44	99					

A

				p = r					
				99					

Funcionamento do Mergesort

	p			q			r		
A	33	55	66	44	99	11	77	22	88

A

p		q		r			
33	55	66	44	99			

A

			p	r			
			44	99			

Funcionamento do Mergesort

	p			q			r		
A	33	55	66	44	99	11	77	22	88

A

p		q		r			
33	44	55	66	99			

Funcionamento do Mergesort

	p				q			r	
A	33	44	55	66	99	11	77	22	88

Funcionamento do Mergesort

	p				q			r	
A	33	44	55	66	99	11	77	22	88

					p	q		p
A					11	77	22	88

Funcionamento do Mergesort

	p				q			r	
A	33	44	55	66	99	11	77	22	88

					p	q		p
A					11	22	77	88

...

Funcionamento do Mergesort

	p	q				r			
A	11	22	33	44	55	66	77	88	99