Algoritmos e Estrutura de Dados

Aula 03 – Exercícios e Análise de Algoritmos de Dividir e Conquistar

Prof. Tiago A. E. Ferreira

Exercício 1

- Determine o custo em tempo computacional, utilizando a notação O(.), dos algoritmos:

a)sum ← 0

for i ← 1 **to** n

do sum ← sum +i

b)sum ← 0

for i ← 1 **to** n

do for j ← 1 **to** i

do sum ← sum +1

Exercício 2

- Escreva em notação O(.) as seguintes funções:
  - n^3-1
  - $n^2+2\log(n)$
  - $3n^n+5*2^n$
  - $(n-1)^n+n^{n-1}$
  - 302

Projeto de Algoritmos

- Abordagem de dividir e conquistar
  - **Dividir** o problema em um determinado número de subproblemas
  - **Conquistar** os subproblemas, reescrevendo-os recursivamente
  - **Combinar** as soluções dadas aos subproblemas

Exemplo de Dividir e Conquistar

- Algoritmo de **ordenação por intercalação**
  - **Dividir:** divide a seqüência de n elementos a serem ordenados em duas subseqüências de n/2
  - **Conquistar:** Classifica as duas subseqüências recursivamente
  - **Combinar:** faz a intercalação das duas subseqüências ordenadas

Algoritmo de Ordenação por Intercalação

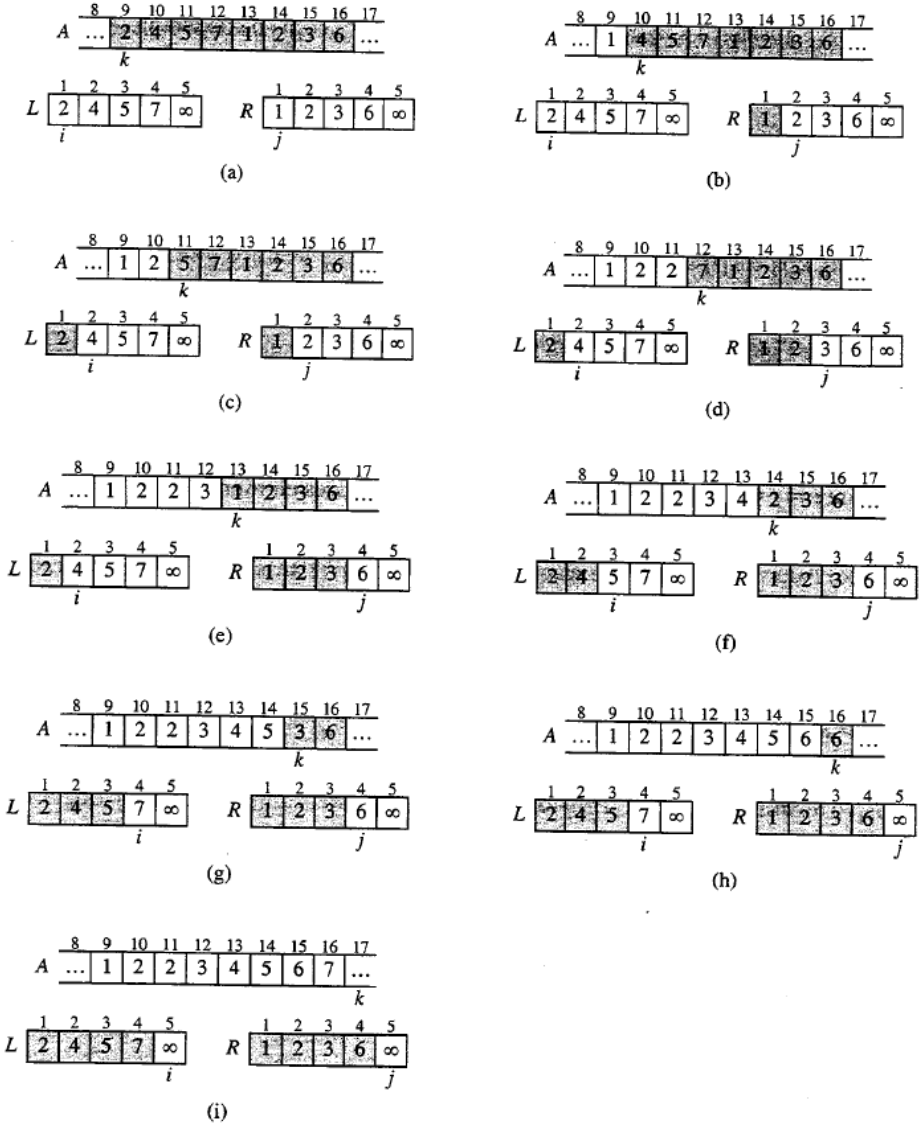
- A operação chave está no passo de combinação, onde são intercaladas (merge) duas subseqüências já ordenadas
  - Será utilizado o procedimento MERGE(A,p,q,r)
    - Onde A é um arranjo, p,q e r são índices de enumeração dos elementos do arranjo, tais que $p \leq q < r$
  - É pressuposto que os sub-arranjos A[p..q] e A[q+1..r] estejam em seqüência ordenada

Combinação

- Suponha a existência de duas pilha de cartas com as numerações para cima
  - Será gerada uma pilha de saída, onde será depositada a carta de menor valor dentre as que estão expostas nas duas pilhas iniciais. Esta pilha de saída é formada com as cartas viradas para baixo.
  - Ao término, será gerada uma única pilha ordenada com todas as cartas das duas pilhas iniciais
  - Sendo n o número total de cartas (duas pilhas iniciais), o custo em tempo será $\Theta(n)$

Algoritmo MERGE(A,p,q,r)

```
MERGE(A, P, q, r)
 1  n1 ← q − p + 1
 2  n2 ← r − q
 3  criar arranjos L[1..n1 + 1] e R[1..n2 + 1]
 4  for i ← 1 to n1
 5        do L[i] ← A[p + i − 1]
 6  for j ← 1 to n2
 7        do R[j] ← A[q + j]
 8  L[n1 + 1] ← ∞
 9  R[n2 + 1] ← ∞
10   i ← 1
11   j ← 1
12   for k ← p to r
13         do if L[i] ≤ R[j]
14               then A[k] ← L[i]
15                    i ← i + 1
16               else A[k] ← R[j]
17                    j ← j + 1
```

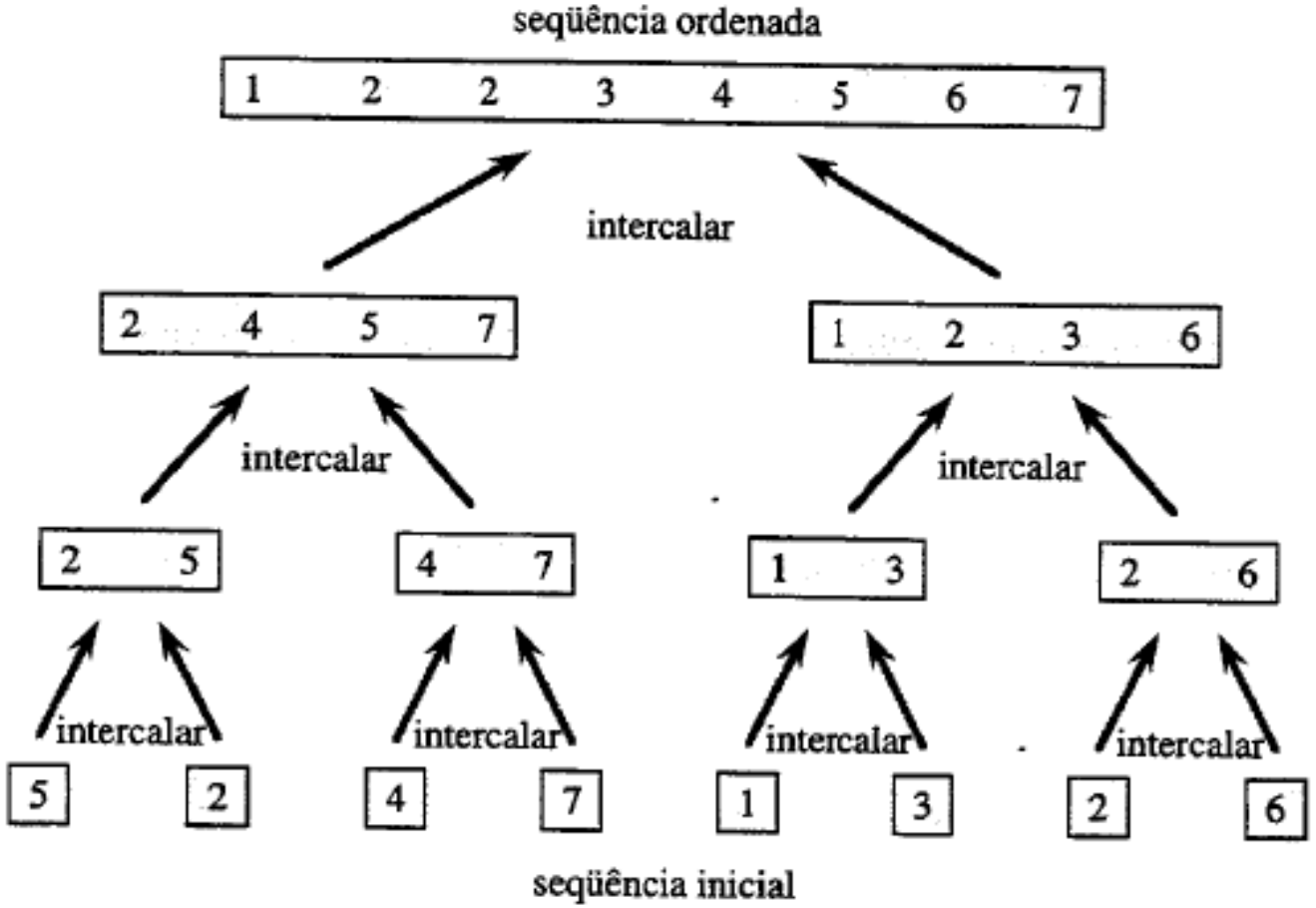


Algoritmo de ordenação por intercalação

- O Algoritmo MERGE-SORT(A,p,r) ordena os elementos do sub-arranjo A[p..r]
  - Se p≥r, então o arranjo tem 1 elemento
  - Caso contrario, é calculado o índice q que particiona o arranjo A[p..r] em dois sub-arranjos: A[p..q], com $\lceil n/2 \rceil$ elementos, e A[q+1,r] com $\lfloor n/2 \rfloor$ elementos

```
MERGE-SORT(A, p, r)
1  if p < r
2     then q ← ⌊(p + r)/2⌋
3          MERGE-SORT(A, p, q)
4          MERGE-SORT(A, q + 1, r)
5          MERGE(A, p, q, r)
```

MERGER-SORT(A,1,Comprimento[A])



Algoritmos Recursivos

- Um algoritmo que tem uma chamada a si próprio, seu tempo de execução freqüentemente pode ser descrito por uma **equação de recorrência** ou **recorrência**
  - Para n pequeno, n≤c (c é uma cte qq), a solução direta demorará um tempo constante, Θ(1)
  - Caso contrário, o problema será dividido em *a* subproblemas de comprimento 1/b do comprimento total
    - E, seja D(n) o tempo para dividir o problema, e C(n) o tempo para combinar as soluções

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \le c, \\ aT(n/b) + D(n) + C(n) & \text{em caso contrário.} \end{cases}$$

Análise da Ordenação por Intercalação

- Sem perda de generalidade, vamos supor que o comprimento do arranjo é uma potência de 2
  - A ordenação por intercalação sobre um único elemento demora um tempo constante
  - Quando n>1, deve-se desmembrar o tempo de execução:
    - Dividir: Calcula o ponto médio do subarranjo, demorando um tempo constante, $D(n)=\Theta(1)$
    - Conquistar: são resolvidos dois problemas recursivamente, cada um com tamanho de n/2, custo de $2T(n/2)$
    - Combinar: algoritmo MERGE, $C(n)=\Theta(n)$

Análise da Ordenação por Intercalação

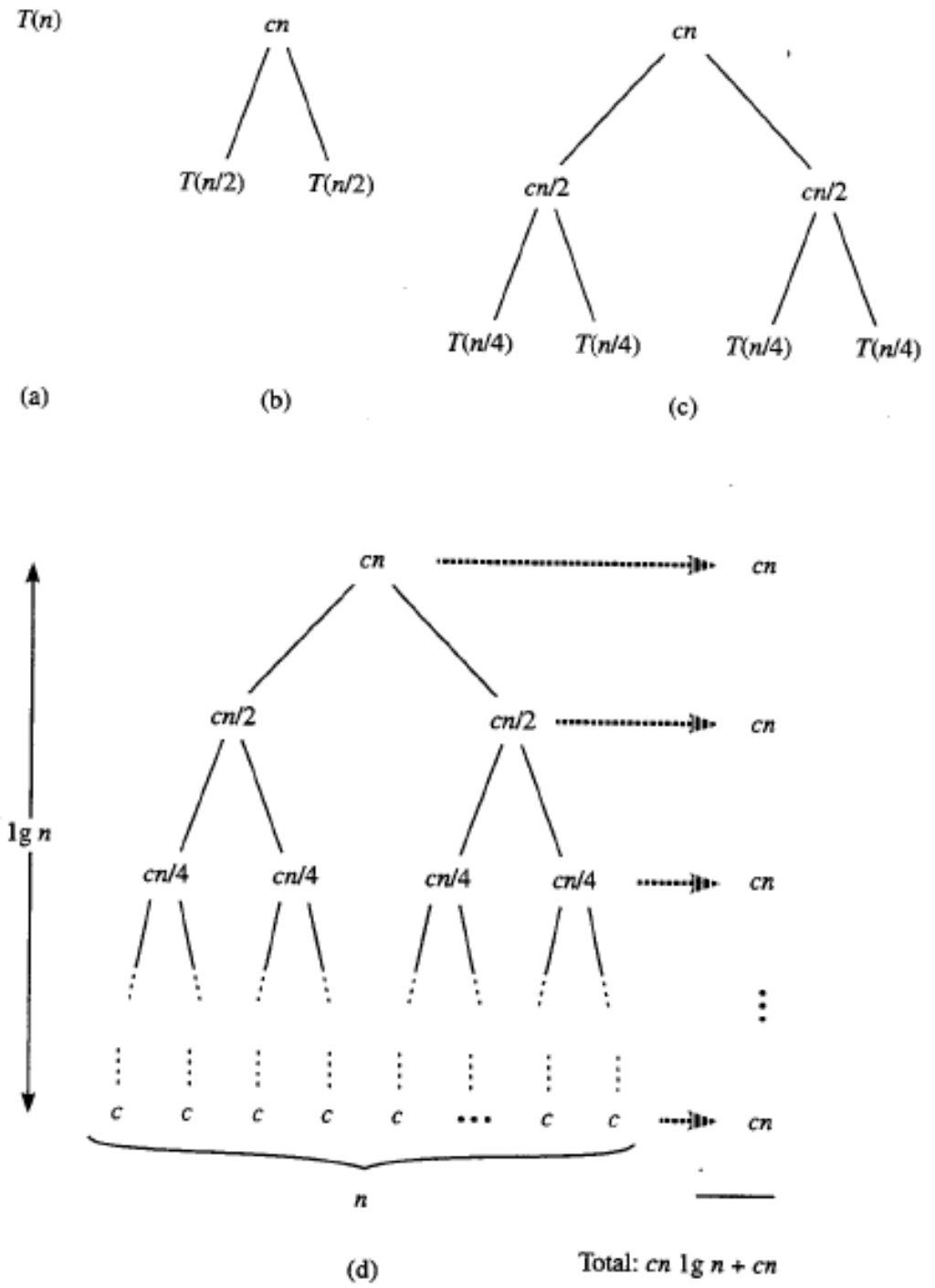
- Custo total:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1, \\ 2T(n/2) + \Theta(n) & \text{se } n > 1. \end{cases}$$

  - Se chamarmos *c* uma cte que represente o tempo exigido para resolver problemas de tamanho 1:

$$T(n) = \begin{cases} c & \text{se } n = 1, \\ 2T(n/2) + cn & \text{se } n > 1. \end{cases}$$

- É possível perceber que T(n)=O(n lg n), lg é o log na base 2?


T(n)
cn
T(n/2)
T(n/2)
cn
cn/2
cn/2
T(n/4)
T(n/4)
T(n/4)
T(n/4)
(a)
(b)
(c)
cn
cn
cn/2
cn/2
cn
cn/4
cn/4
cn/4
cn/4
cn
lg n
c
c
c
c
c
c
c
cn
n
(d)
Total: cn lg n + cn

Exercício 3

- Ilustre, esquematicamente, a operação de ordenação por intercalação sobre o arranjo A=[3, 41, 52, 26, 38, 57, 9, 49]

Exercício 4

- Dada um seqüência numérica ordenada de forma crescente, deseja-se realizar a busca por um determinado. Um forma de realizar esta tarefa é com o usa da **busca binária**, onde o vetor de números é dividido pela metade até se encontrar o elemento desejado. Escreva um pseudo-código para a busca binária. Determine o custo em tempo para o pior caso.