Princípios de Análise de Algoritmo
Exercício #2 (continuação)

Solução

```
1 procedimento subirEscada1000Flexoes(n: inteiro)
2     var i, j: inteiro                    #controladores de laço para escada e flexões
3     para i ← 1 até n faça # c1
4         subirDegrau() # c2
5         para j ← 1 até 1000 faça # c3
6             fazerUmaFlexao() # c4
7     para i ← n até 1 com passo −1 faça # c5
8         descerDegrau() # c6
```

- $T = (n)(c_1 + c_2 + L) + (n)(c_5 + c_6)$, onde $L = (1000)(c_3 + c_4)$. Desenvolvendo teremos: $T = k_0 n + k_1 n \Rightarrow O(n)$.

Princípios de Análise de Algoritmo
Exemplo 5: recursividade

Algoritmo para somar os elementos de uma lista

```
1 função soma(L: arranjo de inteiro): inteiro
2     var N: inteiro ← tam L                    #recuperar o tamanho do vetor
3     var resposta: inteiro                     #guarda resultado da soma
4     se N = 0 então # c1
5         resposta ← 0 # c2
6     senão
7         resposta ← L[0]+soma(subVetor(L, 1, N − 1)) # c3
8     retorna resposta # c4
```

- Cada chamada recursiva da soma decrementa o tamanho da lista. Exceção quando a lista é zero (caso base da recursão).
- Se n é o tamanho inicial, então o número total de chamadas será $n + 1$.
- O custo de cada chamada é: $c_1 + c_2 + c_4$ (última chamada) e $(c_1 + c_3 + c_4)$ (demais chamadas).
- O tempo total $T = n \cdot (c_1 + c_3 + c_4) + c_1 + c_2 + c_4$, ou seja, $T \leq n \cdot c$, onde c é constante $\Rightarrow$ a complexidade é dita **linear**.

Princípios de Análise de Algoritmo
Exemplo 5: recursividade

- Para avaliar a complexidade em relação a memória necessária, é preciso compreender como a memória de microprocessadores é organizada.
- Cada chamada de função ocupa um espaço na pilha de execução, onde é reservado espaço para variáveis locais e parâmetros da função chamada.
- No caso do exemplo existem $n + 1$ chamadas de função. Portanto o consumo de memória é o somatório do comprimento das listas

$$n + (n-1) + (n-2) + \cdots + 1 + 0 =$$
$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \leq c \cdot n^2$$

- Portanto a **complexidade espacial** é uma função quadrática da entrada n.

Princípios de Análise de Algoritmo
Exercício #3

- Escreva uma função para obter o k-ésimo menor elemento de uma lista sequencial L com n elementos (pré-condição: $1 \leq k \leq n$). Podem haver elementos repetidos em L. Elabore sua função de modo a minimizar a complexidade de pior caso e determine esta complexidade. Por fim, descreva a situação correspondente ao pior caso considerado e forneça um exemplo ilustrativo com, pelo menos, $n = 5$ elementos.

Desafio de programação

Exercício #4

- Desenvolva um algoritmo (ou programa) que **recebe** como entrada as coordenadas Cartesianas (x, y) do pontos que definem dois segmentos de reta P_1Q_1 e P_2Q_2 e **determina** se os segmentos têm ou não um ponto em comum.

Referências


J. Szwarcfiter and L. Markenzon
Estruturas de Dados e Seus Algoritmos, 2ª edição, **Cap. 1**.
Editora LTC, 1994.


R. Sedgewick
Algorithms in C, Parts 1-4, 3rd edition. **Cap. 2**
Addision Wesley, 2004.


A. Drozdeck
Data Structures and Algorithms in C++, 2nd edition. **Cap. 2**
Brooks/Cole, Thomsom Learning, 2001.


D. Deharbe
Slides de Aula. **aula 2**
DIMAp, UFRN, 2006.


M. Siqueira
Slides de Aula. **aula 1**
DIMAp, UFRN, 2009.