Instituto Federal de Educação, Ciência e Tecnologia de Brasília – Câmpus Taguatinga
Ciência da Computação – Análise de Algoritmos
Lista de Exercícios – Divisão e Conquista
Prof. Daniel Saad Nogueira Nunes

Aluno: _____
Matrícula: _____

## Exercício 1

**(Exponenciação Rápida)** Dados $x \in \mathbb{R}$ e um $n \in \mathbb{N} \cup \{0\}$ , compute $x^n$. O seu algoritmo deverá levar tempo $o(n)$.

- Entrada: $x \in \mathbb{R}$ e $n \in \mathbb{N} \cup \{0\}$.
- Saída $x^n$.

## Exercício 2

Dada uma matriz quadrada $M_{n \times n}$, elabore um algoritmo para obter $(M_{n \times n})^k$, com $k \in \mathbb{N} \cup \{0\}$. Seu algoritmo deverá levar tempo $O(n^3 \cdot \lg k)$.

## Exercício 3

Elabore um algoritmo que obtenha o $n$-ésimo número de Fibonacci em tempo $O(\lg n)$.

## Exercício 4

O Fluminense finalmente teve que pagar a série B decorrente da sua não disputa em dois campeonatos dessa mesma série. Dada uma tabela contendo o placar de cada jogo do Fluminense, compute o maior intervalo em que o time possuiu o maior saldo de gols. O seu algoritmo deve levar tempo $\Theta(n)$.

- Entrada: um vetor com valores $((p_0, c_0), (p_1, c_1), \ldots, (p_i, c_i), \ldots, (p_{n-1}, c_{n-1}))$, em que $(p_i, c_i)$ é um par que indica que o Fluminense fez $p_i$ gols e sofreu $c_i$ gols na i-ésima partida.
- Saída: Índices $[l, r] \subseteq [0, n-1]$ representando o maior intervalo em que o Fluminense possuiu o maior saldo de gols.

## Exercício 5

**(Distância entre dois pontos)** Projete um algoritmo por indução que dado $n$ pontos, diga qual é o par de pontos que possui a menor distância em tempo $o(n^2)$. É possível obter um algoritmo com cota $\Theta(n \lg n)$ ?

- Entrada: Um conjunto de pontos $P = \{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$ sobre $\mathbb{R}^2$.

- Saída: os pontos $(x_a, y_a)$ e $(x_b, y_b)$ de $P$ com menor distância.

## Exercício 6

**(Skyline)** Dê o pseudocódigo do algoritmo do Skyline discutido em sala de aula. O seu algoritmo deve ter custo $\Theta(n \lg n)$.

- Entrada: um vetor de prédios retangulares $((l_0, h_0, r_0), \ldots, (l_{n-1}, h_{n-1}, r_{n-1}))$ em que $(l_i, h_i, r_i)$ indica o $i$-ésimo prédio que começa em $l_i$ tem altura $h_i$ e termina em $r_i$. O vetor está ordenado pelos valores $l$.
- Saída: O Skyline dos prédios.

## Exercício 7

**(Dominância de Pontos)** Um ponto $(x_a, y_a)$ domina outro ponto $(x_b, y_b)$ se, e somente se, $x_a < x_b \wedge y_a < y_b$. Projete um algoritmo $o(n^2)$ que conta o número de relações de dominância.

- Entrada: Um conjunto de pontos $P = \{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$ sobre $\mathbb{R}^2$.
- Saída: o número de relações de dominância em $P$.

## Exercício 8

Dados dois vetores ordenados $X$ e $Y$ com tamanhos $n$ e $m$ respectivamente, determine o $k$-ésimo menor elemento da união de $X$ e $Y$ em tempo $\Theta(\lg n + \lg m)$.

- Entrada: $X = (x_0, \ldots, x_{n-1})$ e $Y = (y_0, \ldots, y_{n-1})$ vetores ordenados e um parâmetro $k$.
- Saída: O $k$-ésimo menor elemento da união de $X$ e $Y$.

**Solution:**

A ideia deste exercício é realizar uma busca binária nos dois vetores.

Suponha que os vetores $X$ e $Y$ estejam sendo analisados nos intervalos $[l_x, r_x]$ e $[l_y, r_y]$ respectivamente. Obviamente os primeiros intervalos a serem analisados em $X$ e $Y$ são $[0, n-1]$ e $[0, m-1]$.

Vamos projetar a solução via indução. O caso base corresponde à situação em que o intervalo de um dos vetores é o intervalo vazio, assim o $k$-ésimo menor elemento poderá ser consultado no vetor restante.

O passo de indução pode ser desenvolvido utilizando a ideia da busca binária. Sejam $mid_x := \lfloor (l_x + r_x)/2 \rfloor$ e $mid_x := \lfloor (l_y + r_y)/2 \rfloor$ e sejam $n_x := mid_x - l_x + 1$ e $n_y := mid_y - l_y + 1$. Ou seja, $mid_x$ e $mid_y$ indicam o índice do meio do intervalo analisado em $X$ e $Y$ e $n_x$ e $n_y$ indicam o número de elementos desde o início do intervalo até o meio. Temos quatro situações:

- $n_x + n_y > k$: o número de elementos das duas metades inferiores é superior a $k$. Conclusão: o $k$-ésimo menor elemento tem que estar em algum lugar das duas metades inferiores.

- Se $X[mid_x] > Y[mid_y]$, podemos descartar a metade superior de $X$ e procurar o $k$-ésimo menor em sua metade inferior e em $Y$ .
- Senão, podemos descartar a metade superior de $Y$ e procurar o $k$-ésimo menor em sua metade inferior e em $X$.

- $n_x + n_y \leq k$: o número de elementos das duas metades inferiores é menor ou igual a $k$. Conclusão: o $k$-ésimo menor elemento não está em uma das metades inferiores. Devemos descartar uma delas e atualizar o valor de $k$ para $k := k - n_d$, em que $n_d$ corresponde ao número de elementos descartados.
  - Se $X[mid_x] > Y[mid_y]$, podemos descartar a metade inferior de $Y$ e procurar em sua metade superior e em $X$.
  - Senão, descartamos a metade inferior de $X$ e procuramos em sua metade superior e em $Y$.

Isso da margem ao seguinte algoritmo:

**Algorithm 1:** KTH-SMALLEST$(X, Y, l_x, r_x, l_y, r_y, k)$

```
1  if l_x > r_x then
2      return Y[l_y + k − 1]
3  if l_y > r_y then
4      return X[l_x + k − 1]
5  mid_x = (l_x + r_x)/2;
6  mid_y = (l_y + r_y)/2;
7  n_x = mid_x − l_x + 1;
8  n_y = mid_y − l_y + 1;
9  if n_x + n_y > k then
10     if X[mid_x] > Y[mid_y] then
11         return KTH-SMALLEST(X, Y, l_x, mid_x − 1, l_y, r_y, k)
12     else
13         return KTH-SMALLEST(X, Y, l_x, r_x, l_y, mid_y − 1, k)
14 else
15     if X[mid_x] > Y[mid_y] then
16         return KTH-SMALLEST(X, Y, l_x, r_x, mid_y + 1, r_y, k − n_y)
17     else
18         return KTH-SMALLEST(X, Y, mid_x + 1, r_x, l_y, r_y, k − n_x)
```

A função KTH-SMALLEST$(X, Y, l_x, r_x, l_y, r_y, k)$ não pode ser chamada mais do que $\Theta(\lg n + \lg m)$ vezes, pois a cada iteração, a metade de um dos dois vetores é descartada.

## Exercício 9

**(Seleção em tempo linear*)**

Dado um vetor $V$ de tamanho $n$ e um parâmetro $k$, determine qual o $k$-ésimo menor valor de $V$ em tempo $\Theta(n)$.

- Entrada: $V = (v_0, v_1, \ldots, v_{n-1})$ e um parâmetro inteiro $k$.
- Saída: o $k$-ésimo menor elemento de $V$.

## Exercício 10

**(Torre de Hanoi)** Torre de Hanoi é um quebra-cabeça com 3 estacas. A primeira estaca é chamada de estaca origem e a terceira, estaca destino. São empilhados $n$ discos, do maior para o menor, na primeira estaca, de modo que o menor esteja no topo. A segunda e a terceira estaca estão vazias. Cada disco recebe um número de 1 a $n$ do menor para o maior.

A Figura 1 ilustra a configuração inicial do jogo quando $n = 4$.

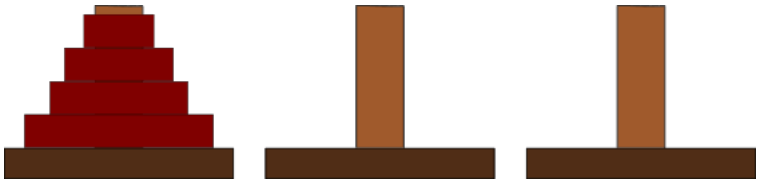

Figura 1: Configuração inicial.

O objetivo do quebra-cabeça é passar todos os discos da primeira para a terceira estaca, obedecendo as seguintes regras:

- Você pode movimentar apenas um disco por jogada.
- Uma jogada consiste em mover um disco no topo de uma estaca para outra estaca.
- Nenhum disco pode ser colocado em cima de outro disco menor.

A Figura 2 ilustra uma solução para o quebra-cabeça.

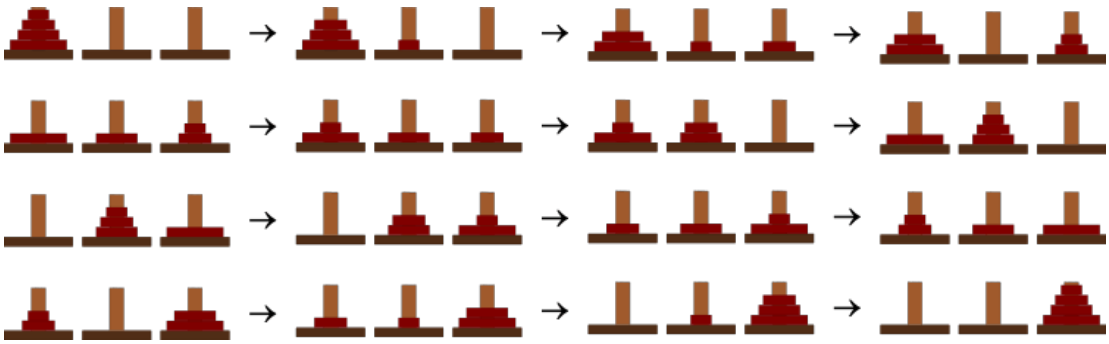

Figura 2: Solução para o quebra-cabeça.

Projete um algoritmo que resolva o problema da torre de Hanoi indicando os movimentos que estão sendo feitos.

- Entrada: um inteiro $n$, indicando a quantidade de discos.
- Saída: Movimentos feitos para completar o quebra-cabeça.

## Exercício 11

**(Sequência de Palavras Fibonacci*)**

Tome a sequência de palavras Fibonacci, definida da seguinte forma:

$$F_i = \begin{cases} a, & i = 0 \\ b, & i = 1 \\ F_{i-1} \cdot F_{i-2}, & n > 1 \end{cases}$$

Isto é, as duas primeiras palavras desta sequência são $a$ e $b$ e as demais podem ser construídas concatenando as duas anteriores. Baseando-se nisso, as sete primeiras palavras de Fibonacci são: $a$, $b$, $ba$, $bab$, $babba$, $babbabab$ e $babbababbabba$.

Tome $F_\infty$ como a palavra infinita gerada utilizando estas regras, isto é: $F_\infty = babbababbabba\ldots$

Elabore um algoritmo que leve tempo $o(n)$ para dizer qual o $n$-ésimo caractere de $F_\infty$;