Algoritmos e Estruturas de Dados – 2018.1
Prova A1 – Prof. Alexandre Rademaker

1. * Lembre-se da definição de que $f = O(g)$ se existe uma constante $c > 0$ tal que $f(n) \leq cg(n)$ para todo n. O que significa que "f não cresce mais rápido que g". Dada a lista de funções abaixo, ordene as funções em ordem crescente de "taxa crescimento". Isto é, se g estiver logo depois de f na sua ordenação, então $f = O(g)$.

$$f_1(n) = n^{2.5} \quad f_2(n) = \sqrt{2n} \quad f_3(n) = n + 10 \quad f_4(n) = 10^n \quad f_5(n) = 10^{10^n} \quad f_6(n) = n^2 \log_2 n$$

Solution:

$$f_2 \leq f_3 \leq f_6 \leq f_1 \leq f_4 \leq f_5$$

2. * Suponha que você tem 6 algorítmos com tempos de execução dados abaixo. Assuma que cada uma das funções abaixo retorna exatamente o número de instruções executadas para uma entrada de tamanho n. Suponha que você tem um computador que pode executar 10^{10} operações por segundo e você precisa computar a resposta em até uma hora. Para cada algorítmo, qual é a maior entrada, n, que você conseguiria obter o resultados em 1 hora?

$$A_1 = n^2 \qquad A_2 = n^3 \qquad A_3 = 100n^2 \qquad A_4 = n \log_2 n \qquad A_5 = 2^n \qquad A_6 = 2^{2^n}$$

Lembre-se que você pode computar $f(n) = X/log_2 n$ com interações sucessivas até o valor convergir. Isto é, $f(\ldots f(f(chute))\ldots)$ irá ser um valor bem próximo da solução. O chute inicial pode ser qualquer valor maior que 1 e menor que X.

Solution: O objetivo da questão não era obter os valores precisos, mas avaliar se os alunos entendem como a complexidade do algorítmo afeta o tempo de execução em função do tamanho das entradas. Os valores aproximados são: (1) $\sqrt{36 \times 10^{12}} = 6 \times 10^6$; (2) 3.3×10^4; (3) 6×10^5; (5) $\log_2 36 + \log_2 10^{12}$; (6) $\log_2 x$ onde x é a resposta do item 5.

O item 4 é o mais complicado. Sabemos que este é o algorítmo mais eficiente e, consequentemente, o que conseguirá tratar a maior entrada em 1 hora. O valor aproximado é 9.06×10^{11}. Vide `http://en.wikipedia.org/wiki/Lambert_W_function`.

Usando ponto fixo e começando com um chute inicial de 10^5 seria possível obter este valor aproximado. Por que este chute inicial? Sabemos que este algorítmo poderá lidar com uma entrada maior que o algorítmo quadrático em 1 hora.

```
> i <- 10^11
> for(j in 1:1000) i <- (36*10^12)/log2(i)
> i
[1] 906316482853
```

3. * Aprendemos duas formas de resolver relações de recorrência. Uma usando o "Master Theorem" e outra usando o método de substituições sucessivas (exercícios).

Teorema 1 (Master Theorem) *If* $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ *for some constants* $a > 0$, $b > 1$, *and* $d \geq 0$ *then*

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Resolva usando o teorema a relação $T(n) = 3T(n/2) + O(n)$.

Solution: $a = 3$, $b = 2$ e $c = 1$. $\log_2(3) \approx 1.58 > 1$. Logo $O(n^{log_2 3})$.

```
Input: k = 3, n =  4
arr[][] = [ {1, 3, 5, 7},
            {2, 4, 6, 8},
            {0, 9, 10, 11} ] ;
Output: 0 1 2 3 4 5 6 7 8 9 10 11
```

Figura 1: Exemplo entrada

4. Suponha que você tem k listas ordenadas, cada uma com n elementos. Você deseja combinar todos eles em um único array com kn elementos. Vide Figura 1

   (a) * Uma possível estratégia: Usando a função *merge-1* (arquivo *respostas.lisp*) junte os primeiros 2 arrays, depois junte o resultado com o terceiro, depois com o quarto e assim por diante. Implemente este algorítmo na função *questao-4-a* no arquivo *respostas.lisp*. Sua função deve receber listas de listas, usando o exemplo acima:

   ```
   (questao-4-a '((1 3 5 7) (2 4 6 8) (0 9 10 11)))
   ```

   (b) * Qual a complexidade de tempo desta função do item anterior em termos de k e n? Responda na função *questao-4-b*.

   Solution: $O(k^2 n)$

   (c) Dê soluçãos mais eficientes implementando as funções *linear-k-merge*, *heap-merge* e *dc-merge* no arquivo *respostas.lisp* conforme descritas.

   Solution: atualizar.

   Bastata usar uma estrutura auxiliar (heap) para manter os menores elementos de cada lista. Para $i = 1 \ldots n$. Faça: remover o menor elemento da estrutura auxiliar e colocar na lista solução ($O(\log k)$). Assumir que este elemento veio da lista j. Remover o primeiro elemento da lista j e colocar no heap (que mantém ordenação), $O(\log k)$. total $O(n \log k)$

   Vide também: `http://cs.uno.edu/people/faculty/bill/k-way-merge-n-sort-ACM-SE-Regl-1993.pdf`

5. Considere um grafo direcionado onde as únicas arestas negativas são aquelas que saem do nó s. Todas as outras arestas são positivas. Pode o algoritmo de Dijkstra [1], começando em s, falhar neste grafo? Ele precisa ser adaptado como? Implemente o algoritmo de Dijkstra adaptado (se necessário).

   Um exemplo de chamada da função é dado abaixo. O grafo é dado por uma lista de adjacências onde cada nó na lista de adjacência de um nó v é dado por uma lista binária (up) onde u é o nó destino e p é o peso da aresta. Note que apenas as arestas que saem de 's podem conter peso negativo.

   ```
   (dijkstra '((s ((a -1) (b 2))) (a ((b 2) ...)) ...) 's)
   ```

   Solution: O algorítmo continua funcionando. Neste caso, como apenas as arestas que saem de s podem ser negativas, basta somar um valor grande, digamos L a todas elas e aplicar o algorítmo. A resposta do algorítmo para uma distância d de um nó qualquer à s deverá ser respondida descontando L, isto é, $d - L$. Observe que neste caso, qualquer caminho de s para qualquer nó $w \neq s$ do grafo irá passar por alguma das arestas que saem de s, ou seja, estas arestas não poderiam não ter sido escolhidas. Na primeira iteração o valor negativo destas arestas não irá atrapalhar a atualização correta da distância dos vizinhos de s.

6. Desenvolva um algoritmo que receba com entrada um grafo direcionado acíclico $G = (V, E)$ e dois vértices $u, v \in V$ e devolva como resposta o número de diferentes caminhos de u para v. (Dica: você pode partir do DFS [2] e adaptá-lo.)

[1] `http://bit.ly/2HOsjsq`
[2] `https://en.wikipedia.org/wiki/Depth-first_search`

Solution: Alguma modificação do DFS para guardar armazenar os caminhos alternativos. Não existe uma única resposta.

7. Em Barbacena, no interior de Minas, existe uma longa e calma rua com casas posicionadas de forma bastante espaçada. Existem apenas casas nesta rua. Podemos pensar na rua com um segmento de reta indo do norte para o sul. Embora seja uma cidade calma de interior, os moradores desta rua são compulsivos usuários de seus celulares que reclamam constantemente da falta de sinal. Sua missão, como novo engenheiro da XIM Celulares é definir onde deverão ser posicionadas antenas de transmissão ao longo desta rua de tal forma que, cada casa esteja a uma distância não superior à 4 Km de alguma antena.

   (a) Implemente na função *questao-7-a* um algoritmo eficiente que calcule o posicionamento das antenas usando o menor número possível de antenas. Considerando a rua como um segmento reto, a entrada do algoritmo será uma lista de números reais representando as posições de cada casa neste segmento.

   Exemplo de formato de entrada e saída, os valores são aleatórios não necessariamente as saídas ótimas.

```
(questao-7-a '(10 20.3 30))
=> (10 20 30)

(questao-7-a '(1 5 6))
=> (4 7)
```

   Solution: Execute o seguinte algoritmo:
   H ← conjunto de todas as casas
   while o conj H tem alguma casa **do**
   Coloque uma base b 4 km à direita da casa mais à esquerda de H.
   Remova todas as casas de H cuja distância até b é de no máximo 4 km.
   end while

   (b) * Qual a complexidade do seu algoritmo? Responda na função *questao-7-b*.

   Solution: $O(nlog_2 n)$ onde n é a quantidade de casas.

8. Implemene o algoritmo guloso na função *questao-8* para a satisfatibilidade de uma formula Horn (Seção 5.3) em tempo linear no comprimento da fórmula (o número de ocorrências de literais) (Dica: use um gráfico direcionado, com um nó por variável, para representar as implicações). Teste seu algorítmo resolve para a entrada da Fórmula 1.

$$(w \wedge y \wedge z) \rightarrow x, (x \wedge z) \rightarrow w, x \rightarrow y, \rightarrow x, (x \wedge y) \rightarrow w, (\bar{w} \vee \bar{x} \vee \bar{y}), (\bar{z}) \quad (1)$$

   Exemplos de formatos de entrada e saída abaixo. Se não tiver solução, retornar `nil`. Notem que no primeiro exemplo abaixo pode haver mais de um *assigment* possível, seu algorítmo só precisa retornar um *assigment* válido. O primeiro exemplo representa a entrada $(y \wedge w) \rightarrow x, (\bar{z})$. O segundo exemplo a entrada $\rightarrow x, (\bar{x})$.

```
(questao-8 '(((y w) x) (z)))
=> ((x 1) (z 0) (y 1) (w 1))

(questao-8 '((() x) (x)))
=> nil
```

   Solution: `http://bit.ly/2HtdimF` `http://www.martinbroadhurst.com/greedy-hornsat-in-python.html`