Universidade Federal Fluminense
Disciplina: Análise e Síntese de Algoritmos
Professor: Luís Felipe

Gabarito (algumas ideias por alto de respostas) Lista de exercícios 1 – Eficiência computacional (parte I)

1. Escreva as seguintes funcões em notação O: $n^3 - 1$, $n^2 + 2\log n$, $3n^n + 5.2^n$, $(n-1)^n + n^{n-1}$, *googol*, $100^{100^{100^{100}}}$.

   Resposta: $n^3 - 1 = O(n^3)$; $n^2 + 2\log n = O(n^2)$; $3n^n + 5.2^n = O(n^n)$; $(n-1)^n + n^{n-1} = O(n^n)$; $googol = 10^{100} = O(1)$; $100^{100^{100^{100}}} = O(1)$

2. Ponha as funções em notação O obtidas na Questão 1 em ordem não decrescente.

   Resposta: Imediato do item anterior: $O(1)$, $O(1)$, $O(n^2)$, $O(n^3)$, $O(n^n)$, $O(n^n)$.

3. Determinar a expressão da complexidade média de uma busca não ordenada de n chaves, n par, em que as probabilidades de busca das chaves de ordem ímpar são iguais entre si, sendo que esse valor é igual ao dobro da probabilidade de qualquer chave par. Supor, ainda, que a probabilidade de a chave se encontrar na lista é igual a q.

   Resposta: Seja p a probabilidade de uma chave ímpar. Temos então que a probabilidade de uma chave par é $\frac{p}{2}$. Distribuindo a probabilidade q pelas n chaves, temos que:

   $$(n/2)p + (n/2)(p/2) = q.$$

   Concluímos que $p = \frac{4q}{3n}$. Logo, a expressão da complexidade média neste caso é dada pela expressão:

   $$C.M. = (1 + 3 + \cdots + n - 1)\frac{4q}{3n} + (2 + 4 + \cdots + n)\frac{2q}{3n} + n(1 - q),$$

   onde a parcela final $n(1-q)$ refere-se ao caso em que a informação procurada não se encontra na lista.

4. Seja $f_1, f_2, \ldots, f_n$ uma sequência de elementos definida do seguinte modo: $f_1 = 0$, $f_2 = 1$, $f_3 = 1$, $f_j = f_{j-1} - f_{j-2} + f_{j-3}$, para $j > 3$. Elabore um algoritmo não recursivo para determinar o elemento f_n da sequência. Calcule sua complexidade em função de n.

   Resposta:
   $f[1] := 0$;
   $f[2] := 1$;
   $f[3] := 1$;
   para $j = 4 \ldots n$ faça: $f[j] := f[j-1] - f[j-2] + f[j-3]$;

   Note que os elementos são preenchidos percorrendo o vetor uma única vez. Cada elemento é obtido com complexidade constante, pela consulta e soma dos três elementos anteriores. Sendo assim, a complexidade do algoritmo acima é $O(n)$.

5. Suponha você esteja escolhendo um dentre os três seguintes algoritmos para resolver um problema:

   - Algoritmo A resolve o problema dividindo-o em cinco subproblemas com metade do tamanho, resolve cada subproblema recursivamente e combina as soluções em tempo linear.
   - Algoritmo B resolve o problema de tamanho n recursivamente resolvendo dois subproblemas de tamanho $n - 1$ e combina as soluções em tempo constante.
   - Algoritmo C resolve o problema dividindo-o em nove subproblemas com um terço do tamanho, resolve cada subproblema recursivamente e combina as soluções em tempo quadrático.

   (a) Quais são as fórmulas recursivas dos tempos desses algoritmos?

   Resposta:
   Algoritmo A: $T(n) = 5T(n/2) + O(n)$;
   Algoritmo B: $T(n) = 2T(n-1) + O(1)$;
   Algoritmo C: $T(n) = 9T(n/3) + O(n^2)$.

   (b) Quais são as complexidades desses algoritmos?

   Resposta:
   Algoritmo A: Pelo Teorema Mestre, temos: $a = 5$, $b = 2$, $f(n) = O(n)$. $n^{\log_b a} = n^{log_2 5} > n^2$, assim $f(n) = O(n^{\log_b a - \varepsilon})$. Portanto, $T(n) = O(n^{log_2 5})$.
   Algoritmo B: Essa recorrência não se encaixa ao Teorema Mestre. Porén, observe que o número de subproblemas dobra n vezes combinando as soluções em tempo $O(1)$. Dessa forma, $T(n) = O(2^n)$.
   Algoritmo C: Pelo Teorema Mestre, temos: $a = 9$, $b = 3$, $f(n) = O(n^2)$. $n^{\log_b a} = n^{\log_3 9} = n^2$, assim $f(n) = O(n^{\log_b a})$. Portanto, $T(n) = O(n^2 \log n)$.

   (c) Qual desses algoritmos você escolheria caso quisesse o mais eficiente?

   Resposta:
   Algoritmo C.

6. Resolva as seguintes relações recorrência e notação O. Caso resolva pelo Teorema Mestre, exiba os valores de a, b e d do teorema.

   (a) $T(n) = 2T(n/3) + 1$

   (b) $T(n) = 7T(n/7) + n$

   (c) $T(n) = 49T(n/25) + n^{3/2 \log n}$

   (d) $T(n) = T(n-1) + n^c$, onde $c \geq 1$

   (e) $T(n) = 2T(n-1) + 1$

7. Considere a seguinte função `f(n)`:

```
função f(n)
   se n>1 entao:
      imprima(estudando)
      f(n/2)
      f(n/2)
```

   Assumindo que n seja uma potência de 2, quantas palavras `estudando` são impressas? Escreva uma relação de recorrência da função `f(n)` e resolva-a.

   Resposta:
   A recorrência é $T(n) = 2T(n/2) + 1$. Pelo Teorema Mestre, temos que são impressas $O(n)$ linhas.

8. Assuma a existência de um algoritmo `mult(x,y)` que faça a multiplicação de números binários com n bits cada, cujos inteiros associados são x e y, e que este algoritmo possua complexidade de tempo n^a, para $a = \log_2 3$. Queremos converter um inteiro decimal 10^n em sua representação binária (Considere n uma potência de 2).

   Dylan precisava realizar esta tarefa, mas ele estava com muita fome. Então ele escreveu uma parte do algoritmo naquele instante e deixou o restante do algoritmo, assim como sua análise, para fazer após sua refeição. Seu algoritmo ficou assim:

```
função pot2bin(n)
   se n=1 então:
      retorne 1010
   senão:
      z = ??? % depois termino
      retorne mult(z,z)
```

(a) Ajude Dylan. Termine seu algoritmo. Bom, acho que fica assim: z = pot2bin(n/2)

Resposta:
$z = pot2bin(n/2)$.

(b) Como Dylan fez um excelente curso de ASA e já estava bem alimentado, ele fez corretamente a análise de complexidade deste seu algoritmo. Qual resultado ele obteve?

Resposta:
$T(n) = T(n/2) + n^{log_2 3}$. Pelo Teorema Mestre, temos: $O(n^{log_2 3})$.

9. Dados dois inteiros a e b de n bits cada, podemos aplicar o algoritmo de Euclides para obter o $mdc(a, b)$. Ao estudar este algoritmo, Dylan foi desafiado a obter um algoritmo por divisão e conquista para obter $mdc(a, b)$.

Ele teve a seguinte inspiração divina:

$$mdc(a,b) = \begin{cases} 2mdc(a/2, b/2), & \text{se } a, b \text{ são pares} \\ mdc(a, b/2), & \text{se } a \text{ é ímpar, } b \text{ é par} \\ mdc((a-b)/2, b), & \text{se } a, b \text{ são ímpares} \end{cases}$$

Dessa fez, Dylan precisava ir para a academia, então ele deixou para concluir sua tarefa quando voltasse:

(a) Dylan precisa provar que sua ideia está correta. Prove, matematicamente, que ele tem uma intuição muito boa.

Resposta:
Se a e b são pares, 2 é um divisor comum de a e b. Assim, o $mdc(a, b)$ será igual a duas vezes $mdc(a/2, b/2)$. Se a é ímpar e b é par, então 2 não é divisor de a, e portanto, podemos remover 2 da fatoração de b sem problema, dado que não é divisor comum de a e b. Se a e b são ímpares, temos que $a-b$ é par. Além disso, $mdc(a, b) = mdc(a-b, b)$. Portanto, pelo caso anterior temos que $mdc(a, b) = mdc(a - b, b) = mdc((a - b)/2, b)$.

(b) Escreva um algoritmo recursivo baseado na ideia de Dylan.

Resposta:

```
mdc(a,b):
  se a=b então:
      retorne a
  senão se (a é par & b é par):
      retorne 2.mdc(a/2,b/2)
  senão se (a é ímpar & b é par):
      retorne mdc(a,b/2)
  senão se (a é ímpar & b é ímpar & a>b):
      retorne mdc((a-b)/2,b)
  senão se (a é ímpar & b é ímpar & a<b):
      retorne mdc(a,(b-a)/2)
```

(c) Qual a relação de recorrência do algoritmo o item 9b? Qual a complexidade deste algoritmo? (Dica: dado um inteiro x com n bits, $x/2$ possui $n-1$ bits).

Resposta:
Assuma que a e b são números com n-bits. Assim, o tamanho de a e b é $2n$ bits. Considerando as condições, todos os casos, exceto quando a é ímpar e b é par, diminui o tamanho de a e b para $2n-2$ bits, enquanto quando a é ímpar e b é par diminui para $2n-1$ bits. Cada operação possui tempo constante, dado que estamos multiplicação ou dividindo os números por 2. Para os casos de subtração de dois números n-bits cada, temos cn operações, para c é uma constante. Dessa forma, o pior caso da recorrência é dado por:

$$\begin{aligned}
T(2n) &= T(2n-1) + cn \\
T(2n-1) &= T(2n-2) + cn \\
T(2n-2) &= T(2n-3) + c(n-1) \text{ dois operandos possuem } n-1 \text{ bits} \\
T(2n-3) &= T(2n-4) + c(n-1) \\
&\cdots \\
T(2) &= T(1) + c
\end{aligned}$$

Assim, $T(2n) = 2c\sum_{i=1}^{n} i$. O que implica em $O(n^2)$.

(d) Dylan obteve algum ganho de complexidade em relação ao algoritmo de Euclides? (Dica: lembre da complexidade da multiplicação de dois inteiros de n bits cada, assim como da dica do item 9c).

Resposta:
O algoritmo de Euclides possui complexidade $O(n^3)$. Assim, o algoritmo do item anterior é mais eficiente.

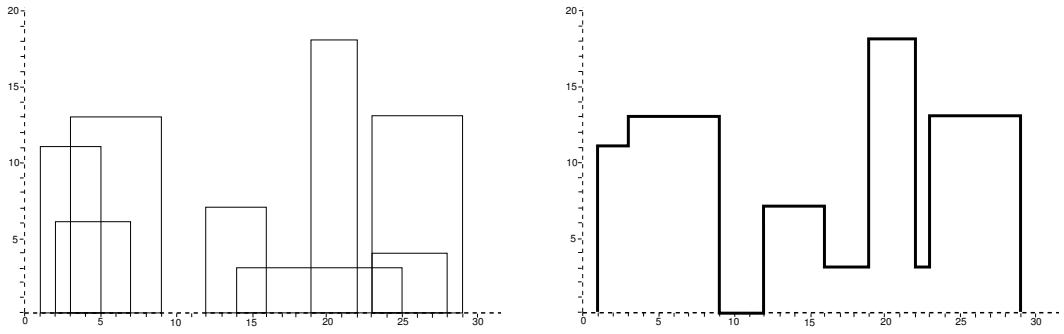
10. Queremos obter um algoritmo por divisão e conquista que compute o *skyline* de n prédios.

Um prédio P_i é representado pela tripla $(\mathbf{E_i}, A_i, \mathbf{D_i})$, onde $\mathbf{E_i}$ e $\mathbf{D_i}$ denotam as coordenadas x mais à esquerda e mais à direita de P_i, respectivamente, e A_i denota a altura de P_i.

O *skyline* de n prédios é uma lista de coordenadas x e alturas que conectam os prédios, ordenados da esquerda para direita. (note que esta lista tem tamanho no máximo $4n$).

Exemplo: O *skyline* dos prédios $(\mathbf{3}, 13, \mathbf{9}), (\mathbf{1}, 11, \mathbf{5}), (\mathbf{12}, 7, \mathbf{16}), (\mathbf{14}, 3, \mathbf{25}), (\mathbf{19}, 18, \mathbf{22}), (\mathbf{2}, 6, \mathbf{7}), (\mathbf{23}, 13, \mathbf{29})$ e $(\mathbf{23}, 4, \mathbf{28})$ é

$(\mathbf{1}, 11, \mathbf{3}, 13, \mathbf{9}, 0, \mathbf{12}, 7, \mathbf{16}, 3, \mathbf{19}, 18, \mathbf{22}, 3, \mathbf{23}, 13, \mathbf{29}, 0)$



prédios *skyline*

(a) Sabendo que o tamanho do *skyline* é o total de elementos (coordenadas e alturas) existentes na sua lista (No exemplo apresentado, o tamanho da *skyline* é 18), descreva um algoritmo que combina o *skyline* A de tamanho n_1 e o *skyline* B de tamanho n_2 no *skyline* de tamanaho $O(n_1 + n_2)$. Seu algoritmo deve ter complexidade $O(n_1 + n_2)$.

Resposta:

Criamos uma lista L, inicialmente vazia, e fazemos uma varredura em A e B da seguinte forma: obtemos o menor dentre os E_1's de A e de B e atualizamos L, juntamente com sua altura respectiva; após isso, obtemos a coordenada x seguinte entre A e B, atualizamos em L se altura correspondente for maior do que a corrente. Prosseguimos com esse procedimento enquanto houver coordenadas em A e B. Como cada coordenada de A e B é visitada uma única vez, temos a complexidade de $O(n_1 + n_2)$.

(b) Descreva um algoritmo de complexidade $O(n \log n)$ para obter o *skyline* de uma sequência de n prédios.

Resposta:

```
Skyline(n prédios):
    Se n = 1:
      retorne o prédio
    A = Skyline(os primeiros n/2 prédios)
    B = Skyline(os últimos n/2 prédios)
    merge A e B como Item (a) acima e retorne skyline de A e B
```

Note que a recorrência é a mesma do Mergesort, com solução $O(n \log n)$.