Universidade Estadual de Maringá
Departamento de Informática – Bacharelado em Informática
5184 – Projeto e Análise de Algoritmos / Prof. Daniel Kikuti

Aluno(a): _____

Primeira avaliação (Valor: 10,0)

1. [Valor: 2,0] Para cada item a seguir, assinale **V**erdadeiro ou **F**also. **Justifique** sua resposta usando as definições de notação assintótica. [Respostas sem justificativa não serão consideradas.]

   (a) ■ **V** ☐ F Se $f(n) = \log_{32} n$ então $f(n) = \Theta(\lg n)$?

   Solução:
   Verdadeiro, pois $\log_{32} n = \frac{\lg n}{\lg 32} = \frac{\lg n}{5}$. Temos que mostrar que: $\frac{\lg n}{5}$ pertence a $O(\lg n)$ e a $\Omega(\lg n)$. No primeiro caso, $\frac{1}{5}\lg n \leq c_2 \lg n$. Portanto, para $c_2 = 1$ e $n_0 = 1$ (qualquer valor de c_2 maior que 1/5 serve) temos $f(n) = O(\lg n)$. No segundo caso, $\frac{1}{5}\lg n \geq c_1 \lg n$ (c_1 deve ser menor ou igual a 1/5). Fixando $c_1 = 1/5$ e usando o mesmo valor para n_0, temos que $f(n) = \Omega(\lg n)$. Logo, $f(n) = \Theta(\lg n)$.

   (b) ■ **V** ☐ F $2^{n+a} = \Theta(2^n)$? Onde $a \in \mathbb{N}$ (conjunto dos números naturais) é uma constante.

   Solução:
   Verdadeiro. Basta observar que $2^{n+a} = 2^n 2^a$. Então para $c_1 = 1/2^a$, $c_2 = 2^a$ e $n_0 = 1$ temos: $c_1 2^n \leq f(n) \leq c_2 2^n$.

   (c) ■ **V** ☐ F $\frac{n^2}{4} - 3n - 16 = \Omega(n^2)$?

   Solução:
   Verdadeiro. Precisamos mostrar que existem constantes positivas c e n_0 tal que: $n^2/4 - 3n - 16 \geq cn^2$, $\forall n \geq n_0$. Dividindo os dois lados da desigualdade $n^2/4 - 3n - 16 \geq cn^2$ por n^2 temos: $c \leq 1/4 - 3/n - 16/n^2$. Observando que $\lim_{n\to\infty}(1/4 - 3/n - 16/n^2) = 1/4$, sabemos que c deve ser menor ou igual a 1/4. Para que c seja positivo, defina $n_0 = 20$ (por exemplo). Assim, $(1/4 - 3/20 - 16/400) = 0.06$ Ou seja, $n_0 = 20$ e $c = 0.06$.

   (d) ■ **V** ☐ F $7n^2 + 13n = O(n^2)$

   Solução:
   Verdadeiro. Precisamos mostrar que existem constantes positivas c e n_0 tal que: $7n^2 + 13n \leq cn^2$. Observe que $7n^2 + 13n \leq 7n^2 + 13n^2 = 20n^2$, portanto, para $c = 20$ e $n_0 = 1$, temos $7n^2 + 13n \leq cn^2$ $\forall n > n_0$.

2. [Valor: 2,0] Assinale Verdadeiro ou Falso.

   (a) ■ **V** ☐ F Suponha que precisamos ordenar n números cujos valores estão no intervalo $[1..n^2]$. É possível afirmar que nesta situação o algoritmo `Counting Sort` não apresenta comportamento linear.

   Solução:
   Verdadeiro, pois neste caso o vetor auxiliar usado na contagem teria n^2 elementos, e percorrer este vetor resultaria em tempo $\Theta(n^2)$.

   (b) ☐ V ■ **F** Suponha que precisamos ordenar n números cujos valores estão no intervalo $[1..n^2]$. É possível afirmar que nesta situação o algoritmo `Radix Sort` possui a mesma complexidade assintótica do `Counting Sort`.

Solução:
Falso, pois para representar um número n são necessários 2^k bits ($k = \lg n$, considerando base binária) e, portanto, para representar um número de tamanho n^2, são necessários 2^{2k} bits. A complexidade do `Radix Sort` é $\Theta(d(n+k))$. Se d (que é o número de dígitos) depende de n (neste caso, $d = 2\lg n$, $k = 2$), então a complexidade do `Radix Sort` seria $\Theta(n \lg n)$.

(c) ☐ V ■ **F** O algoritmo `Insertion Sort` mantém como invariante uma parte inicial $A[1..j-1]$ com os mesmos elementos do vetor original, mas ordenada. Ao passo que um novo elemento j é analisado, é preciso saber qual seria a posição correta deste elemento neste subvetor. Se usássemos uma ideia parecida com busca binária para encontrar esta posição (ao invés de percorrermos este vetor ordenado de maneira sequencial), é possível afirmar que a complexidade do algoritmo seria no pior caso $\Theta(n \lg n)$.

Solução:
Falso, pois apesar de encontrar a posição rapidamente, seria ainda necessário deslocar todos os elementos maiores que j, o que no pior caso ainda tem custo linear, ou seja, a complexidade $\Theta(n^2)$ ainda se mantém.

(d) ■ **V** ☐ F Suponha que um pesquisador desenvolveu um algoritmo de ordenação por comparação cuja complexidade de tempo é dada pela função $f(n) = \frac{n}{2} \lg \frac{n}{2}$. Sabendo que o limite assintótico para algoritmos de ordenação por comparação é $\Omega(n \lg n)$, podemos afirmar que este algoritmo é ótimo.

Solução:
Verdadeiro. Não tem como fazer melhor que $\Omega(n \lg n)$.

3. [Valor: 2,0] Suponha que, para entradas de tamanho n, você tenha que escolher um dentre os três algoritmos A, B e C, descritos a seguir.

   (a) Algoritmo A resolve problemas dividindo-os em cinco subproblemas de tamanho $n/2$, recursivamente resolve cada subproblema, e então combina suas soluções em tempo linear para obter uma solução do problema original.

   (b) Algoritmo B resolve problemas dividindo-os em dois problemas de tamanho $n-1$, recursivamente resolve cada um dos subproblemas e então combina as soluções em tempo constante para obter a solução do problema original.

   (c) Algoritmo C resolve problemas dividindo-os em nove subproblemas de tamanho $n/3$, recursivamente resolve cada subproblema e então combina suas soluções em tempo $O(n^2)$ para obter uma solução do problema original.

   Qual é o consumo de tempo de cada um destes algoritmos (em notação assintótica)? Qual algoritmo é assintóticamente mais eficiente?

Solução:
Precisamos definir as recorrências e o consumo de tempo de cada algoritmo.

(a) A recorrência do Algoritmo A é $T(n) = 5T(n/2) + \Theta(n)$. Usando o Teorema Mestre para identificar o custo, temos que $a = 5$, $b = 2$, $f(n) = n$. Comparamos $f(n)$ com $n^{\log_b a} = n^{\log_2 5} = n^{2,32}$. Caso 1 do Método Mestre, pois $f(n) = O(n^2)$ para $\epsilon = 0,32$. Portanto a complexidade de tempo do Algoritmo A é $\Theta(n^{2,32})$.

(b) A recorrência do Algoritmo B é $T(n) = 2T(n-1) + \Theta(1)$. Não podemos usar o Método

Mestre. Usaremos o método iterativo (poderíamos usar árvore de recursão também).

$$
\begin{aligned}
T(n) &= 2.T(n-1) + c \\
&= 2.(2.T(n-2) + c) + c \\
&= 2.2.T(n-2) + 2c + c \\
&= 2.2.(2.T(n-3) + c) + 2c + c \\
&= 2.2.2.T(n-3) + 2.2c + 2c + c \\
&= \vdots \\
&= 2^k.T(n-k) + \textstyle\sum_{i=0}^{k-1} 2^i c && (T(n-k) = T(1) = \Theta(1) \text{ quando } k = n-1) \\
&= 2^{n-1}.c + \textstyle\sum_{i=0}^{n-2} 2^i c && (\textstyle\sum_{i=0}^{n-2} 2^i = 2^{n-1} - 1) \\
&= 2^{n-1}.c + c(2^{n-1} - 1) \\
&= 2^{n-1}.c + c2^{n-1} - c \\
&= 2.2^{n-1}.c - c \\
&= c.2^n - c.
\end{aligned}
$$

Portanto, a complexidade do Algoritmo B é $\Theta(2^n)$.

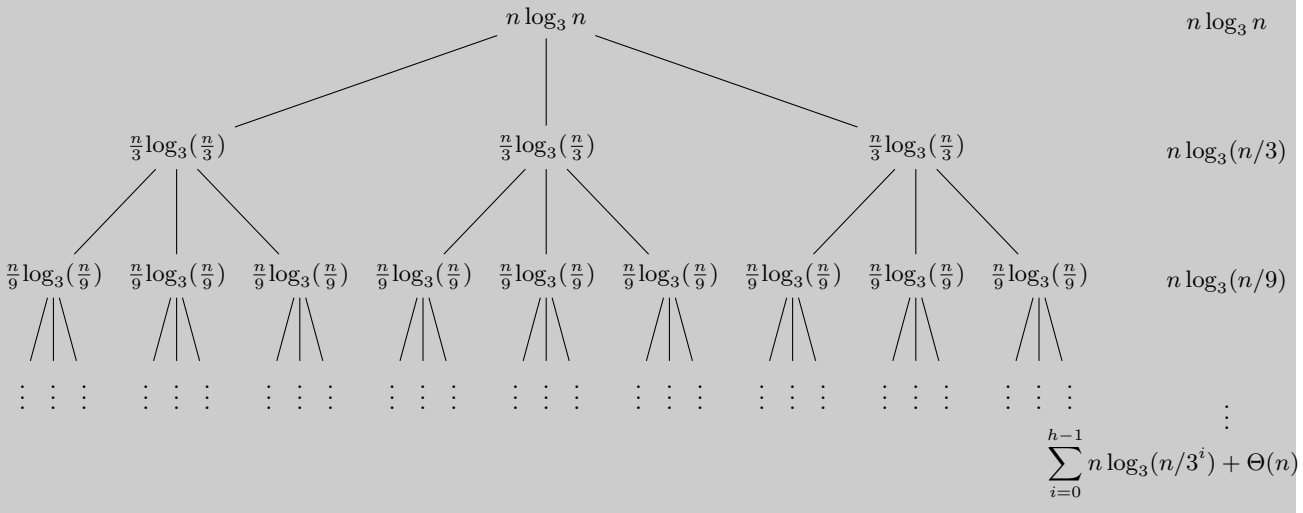
(c) A recorrência do Algoritmo C é $T(n) = 9T(n/3) + \Theta(n^2)$. Usando o Teorema Mestre para identificar o custo, temos que $a = 9$, $b = 3$, $f(n) = n^2$. Comparamos $f(n)$ com $n^{\log_b a} = n^{\log_3 9} = n^2$. Caso 2 do Método Mestre, pois $f(n) = \Theta(n^2)$. Portanto a complexidade de tempo do Algoritmo C é $\Theta(n^2 \lg n)$.

O algoritmo assintoticamente mais eficiente é o Algoritmo C.

4. [Valor: 2,0] Utilize o método de árvore de recursão ou o método iterativo para supor um limite assintótico superior restrito para a recorrência $T(n) = 3T(n/3) + n \log_3 n$. Depois verifique pelo método de substituição que este limite está correto.

Solução:

Construindo a árvore de recursão (ver figura a seguir), observamos que cada nível i possui 3^i nós. O tamanho de um problema em cada nó no nível i é $\frac{n}{3^i}$. O custo de cada nó é $\frac{n}{3^i} \log_3 \frac{n}{3^i}$. A altura da árvore é $h = \log_3 n$. A quantidade de nós folhas é dada por $3^h = 3^{\log_3 n} = n^{\log_3 3} = n$. Assim, sabemos que no último nível temos n nós com custo $\Theta(1)$, ou seja, o custo somado de todas as folhas é $\Theta(n)$.



Somando o custo de todos os níveis obtemos:

$$\begin{aligned}
T(n) &= \sum_{i=0}^{h-1} n\log_3(n/3^i) + \Theta(n) \\
&= n\sum_{i=0}^{h-1} \log_3(n/3^i) + \Theta(n) \\
&= n\sum_{i=0}^{h-1} (\log_3 n - \log_3 3^i) + \Theta(n) \\
&= n\left(\sum_{i=0}^{h-1} \log_3 n - \sum_{i=0}^{h-1} \log_3(3^i)\right) + \Theta(n) \\
&= n\sum_{i=0}^{h-1} \log_3 n - n\sum_{i=0}^{h-1} i + \Theta(n) \\
&= n(\log_3 n - 1)\log_3 n - n\left(\frac{\log_3 n(\log_3 n + 1)}{2} - \log_3 n\right) + \Theta(n) \\
&= n\log_3 n \log_3 n - n\log_3 n - \frac{n}{2}\log_3 n \log_3 n - \frac{n}{2}\log_3 n + n\log_3 n + \Theta(n) \\
&\leq n\log_3^2 n \\
&\leq cn\lg^2 n
\end{aligned}$$

Vamos mostrar por substituição que a recorrência $T(n) = 3T(n/3) + n\log_3 n$ é $O(n\lg^2 n)$.
Hipótese: $T(k) \leq cn\lg^2 n$ para $k < n$ (em particular para $k = n/3$).

$$\begin{aligned}
T(n) &\leq 3(c\frac{n}{3}\lg^2\frac{n}{3}) + n\log_3 n \\
&= cn\lg^2\frac{n}{3} + n\log_3 n \\
&= cn(\lg n - \lg 3)(\lg n - \lg 3) + n\log_3 n \\
&= cn\lg n\lg n - cn\lg n\lg 3 - cn\lg n\lg 3 + cn\lg 3\lg 3 + n\log_3 n \\
&\leq cn\lg n\lg n - 3,16cn\lg n + 2,49cn + n\lg n \\
&= cn\lg n\lg n - (3,16cn\lg n - 2,49cn - n\lg n) \\
&= cn\lg n\lg n \qquad \text{(para } c = 3 \text{ e } n_0 = 2\text{)}.
\end{aligned}$$

Caso base: Como $\lg 1 = 0$, iremos adotar como base para a indução $T(2) = 3.1 + 2.\log_3 2 = 4,26$ e $T(3) = 3.1 + 3.\log_3 3 = 6$. Substituindo temos $T(2) = 4,26 \leq 3 * 2 * 1 * 1 = 6$ e $T(3) = 6 \leq 3 * 3 * 1,58 * 1,58 = 22,47$.

5. [Valor: 2,0] O i-ésimo menor elemento de um conjunto de n elementos é chamado de *i-ésima estatística de ordem*. Por exemplo, o mínimo de um conjunto de elementos é a primeira estatística de ordem ($i = 1$), e o máximo é a n-ésima estatística de ordem ($i = n$). Dado um conjunto A de n números (distintos) e um número i, com $1 \leq i \leq n$, definimos o *problema de seleção* como sendo o problema de encontrar o elemento $x \in A$ que é maior que exatamente $i - 1$ outros elementos de A. Este problema pode ser resolvido no tempo $O(n\lg n)$, pois podemos ordenar os números usando o Mergesort (por exemplo) e então indexar o i-ésimo elemento no vetor de saída. Outra forma de fazer isto é usando o algoritmo descrito a seguir, sendo `RANDOMIZED-PARTITION` o mesmo algoritmo usado no Quicksort aleatório. Suponha que `RANDOMIZED-PARTITION` sempre divide o conjunto e elementos de forma que 1/10 dos elementos são menores que o pivô e 9/10 são maiores que o pivô (sabemos que isto dificilmente aconteceria na prática). Explique quando ocorreria o pior caso e qual seria a complexidade de tempo (usando a notação assintótica).

```
RANDOMIZED-SELECT(A,p,r,i)
1  if p == r
2     return A[p]
3  q = RANDOMIZED-PARTITION(A,p,r)
4  k = q - p + 1
5  if i == k       //O valor pivô é a resposta
6     return A[q]
7  else if i < k
8     return RANDOMIZED-SELECT(A,p,q-1,i)
9  else return RANDOMIZED-SELECT(A,q+1,r,i-k)
```

Solução:

O pior caso ocorrerá quando o elemento que buscamos está sempre no conjunto que contém 9/10 dos elementos restantes. Embora a situação pareça ruim, a recorrência para o pior caso seria: $T(n) = T(\frac{9}{10}n) + \Theta(n)$.

Se aplicarmos o Método Mestre, $a = 1$, $b = 10/9$ e $f(n) = n$. Assim, $n^{\log_{10/9} 1} = n^0 = 1$ e portanto temos o caso 3 do Método Mestre $f(n) = \Omega(1)$. A condição de regularidade $af(n/b) \leq cf(n)$ para $c < 1$ e n suficientemente grande é satisfeita, pois, $1.\frac{n}{10/9} \leq cn$, dividindo os dois lados por n temos que $c \geq 9/10$, ou seja, $9/10 \leq c \leq 1$ satisfaz a condição.

A complexidade no pior caso será $\Theta(n)$.

Boa Prova!!!

Universidade Estadual de Maringá
Departamento de Informática – Bacharelado em Informática
5184 – Projeto e Análise de Algoritmos / Prof. Daniel Kikuti

Aluno(a): _____