CCM-001 – Análise de Algoritmos e Estruturas de Dados
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Profs. Carla N. Lintzmayer e Guilherme O. Mota

Lista 1

- Seja o mais **formal** possível em todas as respostas.
- A lista é uma forma de treino para a prova, que não terá consulta. Evite plágio!
- Fique à vontade também para procurar exercícios nos livros recomendados na bibliografia. CLRS é sigla para o livro "Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C.. *Introduction to Algorithms*. 2nd ed. MIT Press. 2002."

1. (CLRS) Como podemos modificar praticamente qualquer algoritmo para ter um bom tempo de execução no melhor caso?

2. Podemos afirmar que o tempo de execução do Insetion Sort é $\Theta(n^2)$? Justifique.

3. Defina as notações O, Ω e Θ.

4. (CLRS) Sejam $f(n)$ e $g(n)$ funções assintoticamente não negativas. Usando a definição básica da notação Θ, prove que $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$.

5. Em cada situação a seguir, prove se $f(n) = O(g(n))$ ou $f(n) \neq O(g(n))$, e se $f(n) = \Omega(g(n))$ ou $f(n) \neq \Omega(g(n))$. Comente quando $f(n) = \Theta(g(n))$. Considere que a e b são constantes positivas e que log está na base 2:

   (a) $f(n) = \sqrt{n}$ e $g(n) = n^{2/3}$
   (b) $f(n) = n \log n$ e $g(n) = 10n \log 10n$
   (c) $f(n) = \frac{n}{1000}$ e $g(n) = 50^{100}$
   (d) $f(n) = \log_a n$ e $g(n) = \log_b n$
   (e) $f(n) = 1000 \log n$ e $g(n) = \log(n^2)$
   (f) $f(n) = 100^{n+a}$ e $g(n) = 100^n$
   (g) $f(n) = 99^{n+a}$ e $g(n) = 100^n$
   (h) $f(n) = 100^{an}$ e $g(n) = 99^n$
   (i) $f(n) = n^{1.01}$ e $g(n) = n \log^2 n$
   (j) $f(n) = 3^n$ e $g(n) = 2^n$
   (k) $f(n) = \log \sqrt{n}$ e $g(n) = \log n$
   (l) $f(n) = \log n!$ e $g(n) = n \log n$ (dica: compare com n^n e com $(n/2)^{n/2}$)

6. Seja F_n o n-ésimo número da sequência de Fibonacci. Assim, $F_0 = 0$, $F_1 = 1$ e $F_n = F_{n-1} + F_{n-2}$ para $n \geq 2$. Use indução para provar que $F_n \geq 2^{0.5n}$ para todo $n \geq 6$.

7. Prove que o algoritmo Merge Sort visto em aula está correto, isto é, que ele corretamente ordena qualquer vetor dado na entrada. *Dica:* primeiro prove que o algoritmo Merge está correto por invariante de laço e em seguida use indução para provar a corretude do Merge Sort.

8. Prove que o algoritmo Quicksort visto em aula está correto, isto é, que ele corretamente ordena qualquer vetor dado na entrada (independente de como é feita a escolha do pivô). Faça isso de forma similar ao exercício anterior (primeiro prove o algoritmo Partition e então prove o Quicksort).

9. Modifique o algoritmo Partition e o Quicksort vistos em sala para que eles explicitamente considerem elementos repetidos no vetor de entrada. Em particular, faça com que o Partition particione o vetor em três partes ao invés de duas. Mantenha o Partition executando em tempo linear.

10. Suponha que $T(1) = c$, onde c é uma constante positiva (você pode assumir que $c = 1$ se preferir). Resolva as seguintes recorrências com notação O (quando não indicado, use o método que lhe for mais conveniente):

    (a) $T(n) = T(\frac{n}{3}) + n$ (método de iteração)

    (b) $T(n) = aT(\frac{n}{a}) + n$, onde a é inteiro positivo não nulo (método de iteração)

    (c) $T(n) = 2T(n-1) + n$ (método de iteração)

    (d) $T(n) = 4T(\frac{n}{2}) + n$ (método de substituição, suponha $T(n) = \Theta(n^2)$)

    (e) $T(n) = T(n-1)+T(n-2)+3$ (método de substituição, suponha $T(n) = O(2^n)$)

    (f) $T(n) = 4T(\frac{n}{2}) + \sqrt{n}$ (árvore de recursão e método de substituição)

    (g) $T(n) = 7T(\frac{n}{3}) + n^2$ (árvore de recursão e Teorema Mestre)

    (h) $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + n$ (árvore de recursão e método de substituição)

    (i) $T(n) = 64T(\frac{n}{8}) + 7n^3$ (Teorema Mestre)

    (j) $T(n) = 4T(\frac{n}{8}) + \sqrt{n}$ (Teorema Mestre)

    (k) $T(n) = T(\sqrt{n}) + 1$

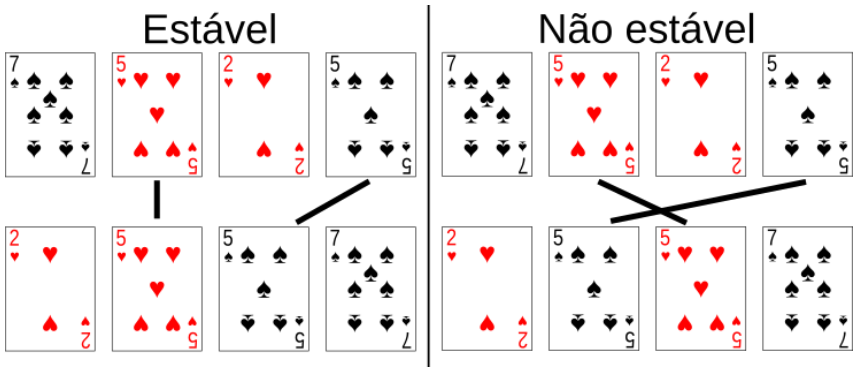
    (l) $T(n) = 2T(\frac{n}{2}) + n \log n$

11. Suponha que temos um vetor A com n elementos *que está ordenado*. Para determinar se um elemento k está armazenado em A ou não podemos usar uma busca chamada de *binária*. A ideia é que se o elemento de uma certa posição i do vetor não é o elemento k procurado, então (se k estiver no vetor) certamente k está em alguma posição à esquerda se $A[i] > k$ ou está em alguma posição à direita se $A[i] < k$. O que a busca binária especificamente faz é sempre comparar k com o elemento que está armazenado na posição do meio do vetor que lhe é passado (i.e., $A[\lfloor (n-1)/2 \rfloor]$). Considerando que podemos ter apenas três casos, que são (i) $k = A[\lfloor (n-1)/2 \rfloor]$, (ii) $k > A[\lfloor (n-1)/2 \rfloor]$ ou (iii) $k < A[\lfloor (n-1)/2 \rfloor]$, a busca (i) termina, (ii) continua procurando por k no subvetor $A[\lfloor (n-1)/2 \rfloor + 1..n-1]$ ou (iii) continua procurando por k no subvetor $A[0..\lfloor (n-1)/2 \rfloor - 1]$. Escreva um pseudocódigo recursivo para o algoritmo de busca binária explicado acima que retorna o índice onde k está armazenado, se k está no vetor, ou o índice onde k deveria estar armazenado, caso contrário. Dê a recorrência para o tempo de execução desse algoritmo e prove pelo método de iteração que seu tempo de execução é $\Theta(\log n)$.

12. Vimos em sala que $T(n) = 2T(\frac{n}{2}) + \Theta(n)$ é $\Theta(n \log n)$ sempre que n é potência de 2. Suponha agora que $n \geq 3$ não é potência de 2. Prove que, ainda assim, $T(n) = \Theta(n \log n)$. *Dica:* se n não é potência de 2, então existe um inteiro $k \geq 2$ tal que $2^{k-1} < n < 2^k$.

13. Seja A um vetor qualquer de inteiros de tamanho n e k um inteiro qualquer. Mostre um algoritmo que verifica se existem posições i e j tais que $A[i]+A[j] = k$ em tempo $O(n \log n)$.

14. Considere o problema de ordenação quando os elementos podem se repetir no vetor de entrada. Dizemos que um algoritmo de ordenação é *estável* se ele não altera a posição relativa dos elementos que têm o mesmo valor.



Fale sobre a estabilidade dos algoritmos de ordenação vistos em sala. Caso algum deles não seja estável, argumente *se e como* é possível deixá-lo estável.