UNIFACCAMP
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Análise de Algoritmos e Complexidade da Computação
Lista de Exercícios 2

Prof. Osvaldo.

1. Aplicando a técnica de *design* de algoritmos indutivos estudada, escreva algoritmos, calcule as suas complexidades, para resolver os seguintes problemas:

   a) Calcular o menor elemento de um vetor de n elementos.
   b) Multiplicar dois números naturais x e y ($x \geq 0$ e $y \geq 0$).
   c) Somar os elementos de um vetor A de n elementos ($n \geq 1$).
   d) Inverter os elementos de um vetor A de n elementos ($n \geq 1$), ou seja, se A = [1 3 5 -2 0 4] então, após invertido, A = [4 0 -2 5 3 1].
   e) Somar os elementos de uma matriz A de n linhas e m colunas ($n \geq 1$, $m \geq 1$).
   f) Calcular o n-ésimo termo da seqüência da Fibonacci (1, 1, 2, 3, 5, 8, 13, ...).

2. Desenvolva um algoritmo recursivo para obter o valor de um polinômio de grau n no ponto igual a x. Qual a complexidade do seu algoritmo? Você seria capaz de desenvolver um algoritmo com complexidade de tempo não superior a $O(n)$?

3. Desenvolva um algoritmo que calcule o maior e o menor elemento de um vetor de n elementos de tal modo que a quantidade de comparações do seu algoritmo não seja maior do que $3n/2$.

4. Desenvolva uma versão com redução do tipo "dividir para conquistar" para os seguintes algoritmos. Calcule as complexidades de pior caso de cada um deles e diga, em cada caso, se melhorou, piorou ou continuou igual às complexidades em relação àquelas outras de soluções diferentes desenvolvidas em sala de aula ou nesta lista de exercícios.

   a) Calcular a soma dos n primeiros naturais;
   b) Calcular o menor elemento de A, um vetor de n elementos.

5. Considere a equação de recorrência a seguir:

   $T(n) = T(n - 1) + 4$
   $T(1) = 4$

   Demonstre, por indução, que $T(n) = 4n$.

6. Use a indução matemática para mostrar que a solução da relação de recorrência abaixo é $T(n) = n \log n$.

$T(n) = 2\ T(n/2) + n$, para $n = 2^k$, $k > 1$.
$T(2) = 2$

7. Escreva um algoritmo para resolver o problema clássico das Torres de Hanói. Qual a complexidade do seu algoritmo?

8. Utilize os algoritmos de ordenação estudados em sala para ordenar o conjunto {4, 0, 3, 1, -1, 6, 5, 2}. Ilustre a sua resposta desenhando o vetor e o movimento dos elemntos.
    a) Ordenação por inserção (*Insertion sort*);
    b) Ordenação por seleção (*Selection sort*);
    c) Ordenação pelo método da bolha (*Bubble sort*);
    d) Ordenação por intercalação (*Merge sort*);
    e) Ordenação rápida (*Quick sort*).

9. Desenvolva exemplos de seqüências de números para os quais a complexidade de tempo do algoritmo `Quicksort` é $\Omega(n^2)$ considerando:

    a) o pivô a ser escolhido é o primeiro elemento;
    b) o pivô a ser escolhido é o último elemento;
    c) o pivô a ser escolhido é o elemento do meio da seqüência.

10. As complexidades de pior caso, melhor caso e caso médio do algoritmo de ordenação pelo método da bolha, desenvolvido em sala de aula, são todas iguais a $O(n^2)$. Desenvolva uma nova versão recursiva deste mesmo algoritmo de tal forma que a complexidade de melhor caso seja igual a $O(n)$. Qual a complexidade de caso médio da nova versão. [Sugestão: se em uma passada para verificação de trocas ("um borbulhamento") não houver troca, então o vetor pode ser considerado ordenado.

11. Desenvolva exemplos de seqüências de números para os quais o algoritmo *Bubble Sort* desenvolvido em questão anterior executa, para um vetor de *n* elementos:

    a) $\Omega(n^2)$ comparações;
    b) *O (n) comparações.*

12. O aluno João da Silva propôs um algoritmo para ordenação que utiliza como critério de redução a busca do menor e do maior elemento do vetor seguido da colocação destes elementos, respectivamente, na primeira e última posições. Assim, o tamanho inicial da solução do problema é reduzido em 2. Desenvolva este algoritmo e calcule a sua complexidade.

13. A aluna Maria das Graças esboçou a idéia de outro algoritmo para ordenação que tem como critério de redução a divisão do vetor em duas partes onde uma delas é indutivamente ordenada e os elementos da outra são, após isto, inseridos na parte ordenada. Desenvolva este algoritmo e calcule a sua complexidade.

14. Qual entre os algoritmos de José da Silva e de Maria das Graças é o mais eficiente (isto é, tem menor complexidade de pior caso)? Você seria capaz de formular um outro algoritmo de ordenação, diferente daqueles desenvolvidos em sala de aula, mais eficiente que os algoritmos de José da Silva e de Maria das Graças?

15. O algoritmo *Merge Sort* divide o conjunto a ser ordenado em duas partes, ordena indutivamente as duas partes e, por fim, intercala estas partes para obter a solução final. Esta estratégia de "divisão" se mostrou mais eficiente do que as reduções de um em um dos algoritmos *Insertion Sort* e *Selection Sort*. O professor Ganancioso está investigando a divisão do conjunto a ser ordenado em 4 partes em vez de duas, influenciado pela promissora estratégia do *Merge Sort*. O professor Ganancioso será bem sucedido em sua proposta de diminuir a complexidade de tempo de *Merge Sort*?

16. Escreva um algoritmo para intercalar *(merge*) dois vetores ordenados. Qual a complexidade de tempo deste algoritmo?

17. Um conhecido jogo consiste em fazer um jogador "advinhar" um certo número pensado por uma pessoa sabendo-se que este número situa-se no intervalo 1 a *n*. Estabeleça uma boa estratégia para "advinhar" o número pensado o mais rapidamente possível. Baseado em sua estratégia, qual é a menor quantidade de tentativas para "advinhar", no pior caso, o número pensado?

18. Resolva a seguinte variação do problema da pesquisa binária que trata de encontrar o menor elemento em uma seqüência ordena circularmente. Dado uma seqüência ordenada de elementos $x_i < x_{i+1} ... < x_n < x_1 < x_2 ... < x_{i-1}$, achar a posição do menor elemento. Por simplicidade, assuma que esta posição é única. Qual a complexidade do seu algoritmo?

19. Considere novamente outra variação do problema da pesquisa binária, escreva um algoritmo para resolvê-lo e calcule a sua complexidade. Desta vez a busca se dará em uma seqüência de tamanho desconhecido conforme descrevemos. Dado uma seqüência teoricamente infinita $x_1 < x_2 < x_3 < x_4 <$ ... e um elemento *z*, achar o índice *i* tal que $x_i = z$.

20. O problema deste exercício é conhecido como **método da bisseção** ou **método de Bolzano** para cálculo de raízes de equações e também pode ser considerado uma pesquisa binária. Este método é freqüentemente estudado em disciplinas como Cálculo Numérico ou Análise Numérica. São dadas uma função real $f(x)$ e dois números reais a e b tais que $a < b$ e $f(a) \cdot f(b) < 0$. O problema consiste em encontrar um valor de x para o qual $f(x) = 0$ no intervalo $[a .. b]$.

21. Desenvolva um algoritmo para computar a união de dois conjuntos de *n* elementos. Os conjuntos são dados como vetores de elementos. A saída deve ser um vetor de elementos distintos, isto é, nenhum elemento deve aparecer mais do que uma vez. A complexidade de tempo no pior caso do seu algoritmo deve ser $O(n \log n)$.

22. Desenvolva algoritmos para as operações abaixo e calcule a complexidade de tempo (pior caso) para os mesmos. Use um vetor como estrutura de dados.
    a) inserir um elemento em uma dada posição de um vetor.
    b) inserir *n* elementos em um vetor (por exemplo, lendo um a um a partir de uma entrada padrão).
    c) inserir um o elemento *x* em um vetor ordenado, mantendo a ordenação. Isto é eficiente?

23. Uma pilha é uma estrutura de dados que que admite operações de inserção e remoção segundo uma disciplina *LIFO* (*last input, first output*), i.e., o último elemento a entrar na estrutura é o primeiro a sair. Explique como se pode implementar duas pilhas em um vetor [1 .. *n*] de tal modo que nenhuma pilha "transborde" a menos que o número total de elementos nas pilhas seja igual a *n*. As operações de inserção e remoção devem ter complexidade de tempo (pior caso) igual a $O(1)$.

24. Mostre como implementar uma fila usando duas pilhas. Analise a complexidade de tempo (pior caso) das operações de inserção e remoção na fila implementada desta forma.

25. Calcule a complexidade de tempo (pior caso) para as seguintes operações realizadas sobre uma lista ligada simples:
    a) inserir um elemento na última posição da lista.
    b) remover o elemento da primeira posição da lista.
    c) remover o elemento de valor igual a *x* da lista.

26. Uma lista ligada circular é uma lista ligada em que os elementos podem ser percorridos circularmente. Desenvolva algoritmos para as operações de inserção, remoção e busca de um elemento em uma lista circular. Quais são as complexidades de tempo (pior caso) destes algoritmos?

27. Escreva um algoritmo para inverter a ordem dos elementos de uma lista ligada simples. Calcule a complexidade de tempo (pior caso) do seu algoritmo.

28. Uma palavra é um **palíndromo** se a seqüência de letras que a forma é a mesma, quer seja lida da esquerda para a direita ou da direita para a esquerda (como exemplos: ovo, raiar e osso). Desenvolva uma estrutura de dados baseada em listas e escreva um algoritmo $O(n)$ para resolver o problema de testar se uma palavra é um palíndromo.

29. Projete uma estrutura de dados dinâmica (listas) para representar Pilhas. A sua estrutura de dados deve permitir implementar em $O(1)$ os algoritmos para empilhar e desempilhar.

30. Projete uma estrutura de dados dinâmica (listas) para representar Filas. A sua estrutura de dados deve permitir implementar em $O(1)$ os algoritmos para inserir e retirar da fila.

Um *pool* é uma estrutura de dados que aceita as seguintes operações:
- Inserção (*x*): insere um elemento *x* na estrutura de dados;
- Remoção: remove algum elemento da estrutura de dados.

Os três exercícios a seguir referem-se à estrutura de dados *pool*.

31. Projete os algoritmos que implementam a estrutura de dados *pool* de tal forma que as operações de inserção e remoção tenham complexidade de tempo (pior caso) igual a $O(1)$. Admita que a estrutura de dados pode conter elementos duplicados.

32. Modifique a implementação da estrutura de dados *pool* da seguinte forma: assuma que todo elemento pode aparecer no máximo uma vez na estrutura de dados. A operação de inserção deve agora verificar a existência de duplicatas. Havendo uma duplicata, o elemento não deve ser inserido. Qual a complexidade de tempo (pior caso) de cada operação?

33. Uma outra variante da implementação da estrutura de dados *pool* é a seguinte: assuma que todos os elementos são identificados por inteiros no intervalo de 1 a *n*, e *n* é pequeno o suficiente para se poder alocar memória de tamanho $O(n)$. Cada elemento pode aparecer no máximo uma vez. Projete algoritmos para as operações de inserção e remoção que trabalhem com complexidade de tempo (pior caso) igual a $O(1)$.

34. Uma operação de união toma dois conjuntos disjuntos S_1 e S_2 como entrada e retorna um conjunto $S = S_1 \cup S_2$. Os conjuntos S_1 e S_2 são usualmente destruídos na operação. Mostre como a operação de união pode ser desenvolvida tendo uma complexidade de tempo igual a $O(1)$.

35. Desenhe árvores binárias de busca inserindo na ordem em que aparecem os seguintes elementos:
    a) 30 15 40 7 8 38 45 18.
    b) 7 8 15 18 30 38 40 45.

36. O percurso de uma árvore em *preorder* resultou na impressão da seqüência A B C F H D L M P N E G I, e o percurso da mesma árvore em *inorder* resultou em F C H B D L P M N A I G E. Construa uma árvore que satisfaça esses percursos. Ela é única?

37. Escreva um algoritmo para retornar a soma dos elementos de uma árvore binária. Qual a complexidade do seu algoritmo?

38. (ENADE 2011) Listas ordenadas implementadas com vetores são estruturas de dados adequadas para a busca binária, mas possuem o inconveniente de exigirem custo computacional de ordem linear para a inserção de novos elementos. Se as

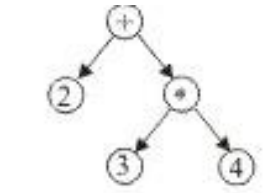
operações de inserção ou remoção de elementos forem freqüentes, uma alternativa é transformar a lista em uma árvore binária de busca balanceada, que permitirá a execução dessas operações com custo logarítmico. Considerando essas informações, escreva um algoritmo recursivo que construa uma árvore binária de busca completa, implementada por apontadores, a partir de um vetor ordenado, V, de *n* inteiros, em que $n = 2^m - 1$, $m > 0$. O algoritmo deve construir a árvore em tempo linear, sem precisar fazer qualquer comparação entre os elementos do vetor, uma vez que este já está ordenado. Para isso,

a) descreva a estrutura de dados utilizada para a implementação da árvore.

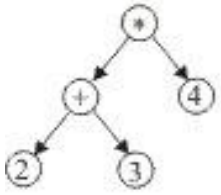
b) escreva o algoritmo para a construção da árvore. A chamada principal à função recursiva deve passar, como parâmetros, o vetor, índice do primeiro e último elementos, retornando a referência ou apontador para a raiz da árvore criada.

39. Mostre que se um nó em uma árvore binária de busca tem dois filhos, então o seu sucessor não possui filho à esquerda e o seu predecessor não possui filho à direita.

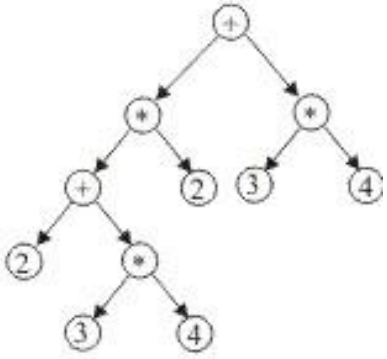
40. Árvores binárias podem ser utilizadas para representar expressões aritméticas identificando, sem ambigüidade, a ordem em que as sub-expressões devem ser avaliadas. Por exemplo, as árvores das figuras abaixo representam as expressões: a) 2 + 3*4; b) (2 + 3)*4 e c) (2 + 3*4)*2 + 3*4. Desenvolva um algoritmo que receba como entrada um ponteiro para a raiz de uma árvore binária, como descrevemos, e retorne a avaliação da expressão, isto é, o algoritmo deverá retornar os valores 14, 20 e 40 para as árvores das figuras (a), (b) e (c), respectivamente. Admita que as expressões conterão somente os operadores de adição e multiplicação. Calcule a complexidade do seu algoritmo.



(a)



(b)



(c)

41. Ilustre (desenho) a operação Inserir (*A*, 12, 3) em um *heap A* = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1]. Represente o *heap* na sua ilustração como uma árvore com ponteiros.

42. - Ilustre (desenho) a operação Remover (*A*, 12) em um *heap A* = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1]. Represente o *heap* na sua ilustração como uma árvore com ponteiros.

43. Qual a diferença entre as propriedades de um *heap* e de uma árvore binária de busca? É possível imprimir em ordem crescente todos os nós de um *heap* com complexidade de tempo igual a *O* (*n*)?

44. Mostre como um *heap* pode ser utilizado para criar um algoritmo de ordenação *O* (*n* log *n*).

45. Ilustre a inserção dos elementos {5, 28, 19, 15, 20, 33, 12, 17, 10} em uma tabela de *hashing* com colisões resolvidas por encadeamento. Considere uma tabela com 9 *slots*, numerados de 0 (zero) a 8, e tome $h(x) = x$ **mod** 9 como função de *hashing*.