Curso de Tecnologia em Sistemas de Computação
Disciplina: Estrutura de Dados e Algoritmos - GABARITO
AP1 - Segundo Semestre de 2005

Nome -
Assinatura -

Observações:

1. Prova sem consulta e sem uso de máquina de calcular.
2. Use caneta para preencher o seu nome e assinar nas folhas de questões e nas folhas de respostas.
3. Você pode usar lápis para responder as questões.
4. Ao final da prova devolva as folhas de questões e as de respostas.
5. Todas as respostas devem ser transcritas nas folhas de respostas. As respostas nas folhas de questões não serão corrigidas.

---

1. A sequência de Fibonacci é dada pela seguinte fórmula recorrente:

$$F_1 = 0, \quad F_2 = 1, \quad F_n = F_{n-1} + F_{n-2} \text{ (para } n \geq 3)$$

onde F_n é o n-ésimo termo da sequência.

Portanto, a sequência é $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$.

   a. (2,0) Escreva um algoritmo *recursivo* que gere o n-ésimo termo da sequência de Fibonacci. Comente sobre a complexidade deste algoritmo.

      função FIBO(n)
      se $n = 1$
          então retorne 0
          senão se $n = 2$
              então retorne 1
              senão retorne FIBO($n - 1$) + FIBO($n - 2$)

      A complexidade desta função é exponencial em n, pois o número de chamadas recursivas na árvore de recursão cresce exponencialmente. O número de chamadas recursivas $R(n)$ para $n = 1$ ou $n = 2$ é 0, e a partir daí $R(n) = R(n-1) + R(n-2) + 2$ para $n \geq 3$. A sequência $R(1), R(2), R(3), \ldots$ é portanto igual a $0, 0, 2, 4, 8, 14, 24, 36, \ldots$, que cresce exponencialmente.

   b. (2,0) Escreva um algoritmo *não recursivo* que gere o n-ésimo termo da sequência de Fibonacci. Comente sobre a complexidade deste algoritmo.

      FIBO[1]:= 0
      FIBO[2]:= 1
      para $j = 3, \ldots, n$ faça
          FIBO[j]:= FIBO[$j - 1$] + FIBO[$j - 2$]

      Nesta implementação, FIBO é um vetor com n posições. Vê-se claramente que este algoritmo é linear em n, pois são executadas $n - 2$ iterações do comando "para".

2. (2,0) Descreva o algorimo de *busca binária* numa lista ordenada com $n \geq 1$ elementos.

   função BUSCA-BIN(x, L) %procura chave x na lista L, ordenada

   $inf := 1$; $sup := n$; BUSCA-BIN:=0

   enquanto $inf \leq sup$ faça

   $meio := \lfloor (inf + sup)/2 \rfloor$

   se $L[meio].chave = x$

   então BUSCA-BIN $:= meio$; $inf := sup + 1$

   senão se $L[meio].chave < x$

   então $inf := meio + 1$

   senão $sup := meio - 1$

3. (2,0) Descreva o algoritmo de *inserção em listas simplesmente encadeadas*. Comente sobre a complexidade deste algoritmo.

   No algoritmo abaixo, assuma que o nó cabeça da lista encadeada está apontado pelo ponteiro *ptlista*, e que x é a chave a ser inserida. Obs: λ é o ponteiro nulo.

   $ant := ptlista$; $pont := \lambda$ % ant e $pont$ são ponteiros auxiliares

   $ptr := ptlista$ % ptr é um ponteiro que percorre a lista

   enquanto $ptr \neq \lambda$ faça

   se $ptr \uparrow .chave < x$

   então

   $ant := ptr$; $ptr := ptr \uparrow .prox$

   senão

   se $ptr \uparrow .chave = x$ então $pont := ptr$

   $ptr := \lambda$

   se $pont \neq lambda$

   então "elemento já está na lista"

   senão

$ocupar(pt)$ % alocar um novo nó

$pt \uparrow .info := novo - valor$

$pt \uparrow .chave := x$

$ant \uparrow .prox := pt$ % acertar lista

fim-do-algoritmo

A complexidade de pior caso do algoritmo anterior é $O(n)$, pois no pior caso pode-se ter que percorrer toda a lista para encontrar o ponto exato de inserção (ou concluir que o elemento já existe na lista).

4. (2,0) Determine quantas folhas possui uma árvore binária cheia com $n \geq 1$ nós. Justifique sua resposta.

Uma árvore binária cheia com $n \geq 1$ nós tem $\frac{n+1}{2}$ folhas.

Pois se a árvore é cheia e tem k níveis, então:

$$n = 1 + 2 + 4 + \ldots + 2^{k-1},$$

onde cada parcela acima corresponde aos nós presentes em cada nível, começando pelo primeiro até o k-ésimo.

Efetuando a soma dos termos da P.G. acima, vem:

$$n = 2^k - 1.$$

Mas o número f de folhas corresponde exatamente aos nós presentes no k-ésimo nível. Logo, $f = 2^{k-1}$, donde:

$$f = 2^{k-1} = \frac{2^k}{2} = \frac{n+1}{2}.$$