Estrutura de Dados - 1o. período de 2014

Gabarito da Primeira Avaliação à Distância

1. (1.5) Para cada par de funções f, g abaixo, determinar se $f = O(g)$, $f = \Omega(g)$ e/ou $f = \Theta(g)$.

   a. $f = n^2 + n \log n$, $g = n \log n$

   Resposta: $f = \Omega(g)$.

   b. $f = n^2 + \log n$, $g = n^2 + \sqrt{n}$

   Resposta: $f = \Theta(g)$, $f = O(g)$ e $f = \Omega(g)$.

   c. $f = n \log n$, $g = n\sqrt{n}$

   Resposta: $f = O(g)$.

2. (2.0) Para cada item abaixo, responda "certo" ou "errado", justificando em ambos os casos.

   a. Se a complexidade de caso médio de um algoritmo for $\Theta(f)$, então o número de passos que o algoritmo efetua no pior caso é $\Omega(f)$.

   Resposta: Certo. A complexidade de pior caso, por definição, é sempre superior ou igual a complexidade de caso médio.

   b. Se a complexidade de pior caso de um algoritmo for $\Theta(f)$, então o número de passos que o algoritmo efetua, qualquer que seja a entrada, é $O(f)$.

   Resposta: Certo. Se a complexidade de pior caso de um algoritmo for $\Theta(f)$, então o número de passos que o algoritmo efetua, qualquer que seja a entrada, é limitado superiormente por f.

   c. Se um limite inferior para um problema P é n^2, então todo algoritmo para P tem complexidade de pior caso $\Theta(n^2)$.

   Resposta: Errado. Se um limite inferior para um problema P é n^2, então todo algoritmo para P tem complexidade de pior caso $\Omega(n^2)$.

   d. Se um limite inferior para um problema P é n^2, então todo algoritmo ótimo para P tem complexidade de pior caso $O(n^2)$.

   Resposta: Errado. Se existir um outro limite inferior maior que n^2, por exemplo, n^3 para P, então, pela definição de limite inferior, não é possível existir um algoritmo para P com complexidade de pior caso $O(n^2)$.

3. Considere o seguinte algoritmo de ordenação de um vetor V cujas posições inicial e final são i e j, respectivamente, onde $i < j$:

   - Descubra a posição k onde o elemento máximo de V se encontra

- Troque os conteúdos das posições j e k

- Repita os dois passos acima, aplicando-os agora às posições i até $j - 1$.

Resolva os itens a seguir:

a. (1.0) Escreva um algoritmo *recursivo* que implementa o processo acima.

Resposta:

Algoritmo:

```
Selecao(V, i, j)
    Se (i < j) então
        k := j;
        Para aux_2 := i até (aux_1 − 1) Faça
            Se (V[aux_2] > V[k]) então
                k := aux_2;
        temp := V[j];
        V[j] := V[k];
        V[k] := temp;
        Selecao(V, i, j − 1);
```

b. (1.0) Escreva um algoritmo *iterativo* que implementa o processo acima.

Resposta:

Algoritmo:

```
aux_1 := j;
Enquanto (i < aux_1) Faça
        k := aux_1;
        Para aux_2 := i até (aux_1 − 1) Faça
                Se (V[aux_2] > V[k]) então
                        k := aux_2;
        temp := aux_2;
        V[aux_1] := V[k];
        V[k] := temp;
        aux_1 := aux_1 − 1;
```

c. (0.5) Quais as complexidades dos algoritmos, em relação ao número de trocas?

Resposta: Ambos algoritmos efetuam $O(n)$ trocas.

4. (1.5) Seja V um vetor com n posições. Escreva um algoritmo que construa uma lista encadeada L, com nó cabeça, a partir de V, de forma que os elementos de L sejam os mesmos de V, de forma ordenada crescente. Por exemplo, se V contiver os elementos 1 7 3 5 8, nesta ordem, a lista L deverá conter os elementos 1 3 5 7 8, nesta ordem.

Resposta:

Algoritmo:

Para $i := 1$ até n Faça
  $V_{aux}[i] := V[i]$;
Selecao$(V_{aux}, 1, n)$; % O procedimento "Selecao" foi apresentado em $3(a)$.

ocupar(PTLISTA);
Pont:=PTLISTA;
Para $i := 1$ até n faça
  ocupar(PT);
  PT↑.info:=$V_aux[i]$;
  PT↑.prox:=λ;
  Pont↑.prox:=PT;
  Pont:=Pont↑.prox;

5. (1.5) Sejam L_1 e L_2 duas listas ordenadas, simplesmente encadeadas com nó-cabeça. Apresentar um algoritmo que construa uma lista ordenada contendo todos os elementos que pertencem a apenas uma das listas, mas não a ambas.

Resposta:

Algoritmo:

$pont1 := ptlista1 \uparrow .prox$ % ponteiro para a lista L_1
$pont2 := ptlista2 \uparrow .prox$ % ponteiro para a lista L_2
$ptaux := ptnovo$ % a lista resultante iniciará em *ptnovo*

enquanto $pont1 \neq \lambda$ e $pont2 \neq \lambda$ faça
  se $pont1 \uparrow .info < pont2 \uparrow .info$ então
    $ocupar(pt)$
    $pt \uparrow .info := pont1 \uparrow .info$
    $pt \uparrow .prox := \lambda$
    $ptaux \uparrow .prox := pt$
    $ptaux := pt$ % ptaux aponta para o último nó
    $pont1 := pont1 \uparrow .prox$
  senão
    se $pont1 \uparrow .info > pont2 \uparrow .info$ então
      $ocupar(pt)$
      $pt \uparrow .info := pont2 \uparrow .info$
      $pt \uparrow .prox := \lambda$
      $ptaux \uparrow .prox := pt$
      $ptaux := pt$
      $pont2 := pont2 \uparrow .prox$
    senão
      $pont1 := pont1 \uparrow .prox$
      $pont2 := pont2 \uparrow .prox$

enquanto $pont1 \neq \lambda$ faça

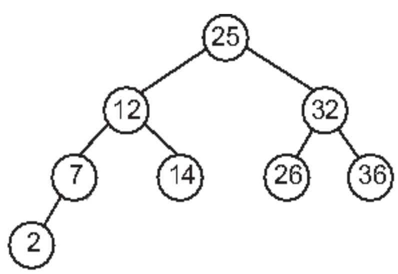
$ocupar(pt)$
$pt \uparrow .info := pont1 \uparrow .info$
$pt \uparrow .prox := \lambda$
$ptaux \uparrow .prox := pt$
$ptaux := pt$
$pont1 := pont1 \uparrow .prox$

enquanto $pont2 \neq \lambda$ faça

$ocupar(pt)$
$pt \uparrow .info := pont2 \uparrow .info$
$pt \uparrow .prox := \lambda$
$ptaux \uparrow .prox := pt$
$ptaux := pt$
$pont2 := pont2 \uparrow .prox$

6. (1.0) Desenhe uma árvore binária T que satisfaça os requisitos pedidos, em cada caso.

   a. T é uma árvore completa com altura 4 e número mínimo de nós.
   Resposta:



   b. T é uma árvore estritamente binária, com 3 níveis e número máximo de nós.
   Resposta: