Algoritmos e Estruturas de Dados

Exame Tipo Presencial • 20 de Janeiro de 2020

Este teste é composto por 7 páginas contendo 7 grupos de perguntas. **Para perguntas com resposta de escolha múltipla, respostas erradas com cotação *c* e *n* respostas possíveis descontam −*c* / (*n* − 1).** Identifique já todas as folhas do teste com o seu nome e número. Na mesa em que está a fazer o exame deve ter apenas lápis/caneta, identificação e este exame. Pode utilizar o verso das folhas como rascunho. Deve responder às perguntas no espaço deixado para o efeito.

Cotações Perguntas 1) _____ 2)_____ 3)_____4)_____5)_____6)______7)______

---

1. (2.0) Perguntas de escolha múltipla.

1.1 (0.4) Considere as seguintes afirmações:
1. O algoritmo *insertionSort* tem complexidade temporal $O(n^2)$.
2. O algoritmo *insertionSort* tem complexidade temporal $O(n^3)$.
3. Se o *array* estiver ordenado de forma crescente, o algoritmo *insertionSort* é linear em n.

Quais das afirmações acima são falsas?
a) 1 e 3.
b) 1 e 2.
c) 2 e 3.
d) 2.
e) 1.
f) 1, 2 e 3.
g) Nenhuma.

1.2 (0.4) Considere uma árvore de pesquisa binária de inteiros se encontra inicialmente vazia. Seguidamente, são inseridos nesta árvore, sequencialmente e por ordem crescente, os inteiros entre 1 e N. O número médio de operações necessárias para procurar por um nó que faz parte da árvore é:

a) N
b) lg N
c) N/2
d) lg N/2

1.3) (0.4) Qual dos seguintes algoritmos é mais eficiente para ordenar um *array* cujos elementos são todos iguais:
a) Selection sort
b) Mergesort
c) Quicksort
d) Insertionsort
f) Todos os algoritmos acima são igualmente eficientes para este caso

1.4) (0.4) Considere as seguintes aproximações tilde:
1. $n+1 \sim n$
2. $1 + 1/n \sim 1$
3. $2n^3 + 5n^3 - 15n^2 + n \sim 5n^3$
4. $1 + 3n + \log n \sim n$

Quais destas aproximações estão correctas?
a) Todas
b) Nenhuma
c) 1 e 2
d) 1 e 3
e) 2 e 4
f) 2 e 3
g 1, 2 e 3
h) 1, 2 e 4

1.5) (0.4) Considere as seguintes afirmações.
1 Na implementação de uma tabela de dispersão, a escolha de uma boa funcão de dispersão assegura sempre a inexistência de colisões.
2 A resolução de colisões através de encadeamento separado consiste na utilização de duas funções de dispersão.
3 Uma colisão ocorre quando duas chaves iguais geram o mesmo valor de dispersão (hash value).
4 Uma tabela de encadeamento separado com tamanho M igual ao número de elementos guardados N, tem em média N/4 elementos guardados em cada balde/lista.

Quais das afirmações acima são falsas?
a) 1 e 2
b) 1 e 3
c) 2 e 4
d) 2 e 3
e) 1, 2 e 4
f) Nenhuma
g) Todas

2. (4.5) Exercícios sobre a linguagem de programação Java.

a) (2.0) Escreva um método estático chamado *mdc* em linguagem Java que dados 2 argumentos inteiros positivos *x* e *y*, retorna um inteiro que corresponde ao máximo divisor comum entre *x* e *y*. Recorde que o máximo divisor comum entre dois números *x* e *y* é o maior valor inteiro n que é simultaneamente divisor de x e y (ou seja, o resto da divisão inteira de x e y por n, é 0).
Sugestão: uma forma simples de implementar este método é experimentar sucessivamente os vários valores possíveis.

Exemplo: mdc(4,6) = 2

R:

```
public static int mdc(int x, int y)
{
    int res;
    int min = x<y? x : y;

    for(int i = min; i > 1; i--)
    {
        if(x%i == 0 && y%i == 0) return i;
    }

    return 1;
}
```

b) (1.0) Considere o seguinte método em Java:

```
public static int incognito(int n)
{
    int res = 0;
    for(int k = 0; k < n; k++)
    {
        for(int j = n; j > 0; j--)
        {
            res++;
        }
    }
    return res;
}
```

Indique o valor retornado pelo método para as seguintes invocações:

i) incognito(0)
R: 0

ii) incognito(2)
R: 4

iii) incognito(4)

R: 16

c) (1.5) Considere uma pilha (*stack*), inicialmente vazia, sobre a qual é executada a seguinte sequência de instruções:

```
StackList<Integer> s = new StackList<Integer>();
int j;
s.push(0);
for(int i = 1; i< 5; i++)
{
    j = s.pop();
    s.push(i);
    s.push(j);
    s.push(i);
}

while(s.size()>0)
{
    System.out.print(s.pop() + ",");
}
```

Indique a sequência impressa no ecrã quando o código é executado.

R: 4,3,4,2,3,1,2,0,1

3. (2.0) Resolva os seguintes exercícios sobre análise de complexidade temporal.

a) (1.0) Admita que dispõe de duas árvores binárias de pesquisa *a1* e *a2*, ordenadas e balanceadas, com N elementos cada. Considere um algoritmo que verifica se a primeira árvore *a1* está contida na segunda árvore *a2*. Este algoritmo percorre a primeira árvore e, para cada elemento *e*, verifica se esse elemento se encontra na árvore *a2*. Indique, justificando, qual a complexidade temporal do algoritmo em função de N.

R: Se o algoritmo percorre a primeira árvore para todos os elementos N, e procura cada elemento na árvore a2, então T(N) = O(N * Tpesquisa). Como a árvore a2 tem também N elementos, e é balanceada, quer dizer que a profundidade da árvore a2 é dada por log N, e no pior caso o tempo de pesquisa é dado por O(log N). Assim sendo, T(N) = O(N * log N).

b) (1.0) Considere um algoritmo com complexidade temporal dada por $T(n) = O(n \log_2 n)$. Ao executar testes de razão dobrada, estime os valores esperados para a razão dobrada *r* no início e no fim dos testes.

R: $r = \frac{T(2n)}{T(n)} = \frac{2n\log_2 2n}{n\log_2 n} = \frac{2(\log_2 2 + \log_2 n)}{\log_2 n} = \frac{2 + 2\log_2 n}{\log_2 n} = \frac{2}{\log_2 n} + 2$

Para os testes de razão dobrada, escolhendo como n inicial 2, e n final +∞

n=2, r = 4
n=∞, r = 2

Ou seja, espera-se que r começe em 4, e termine em 2.

4. (2.0) Considere o método *partition* usado no algoritmo de ordenação *quicksort*, com a assinatura *int partition(Comparable[] a, int low, int high)*. Suponha que o método *partition é* invocado com os seguintes argumentos:

a=[16, 18, 10, 8, 14, 22, 15, 19, 17], low = 0, high=8

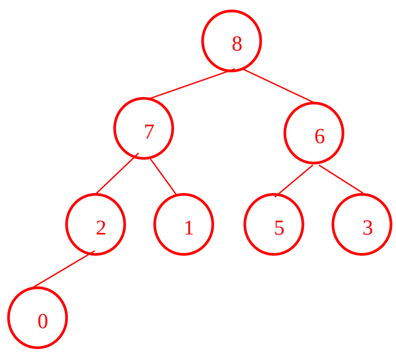
Indique qual o conteúdo do *array* após a execução do método *partition*, assumindo que o pivô é escolhido como sendo o elemento da posição low. Indique também qual a posição do pivô, retornada pelo método.

R: low=0,high=8,pivô=16
1.ª iteração:
i=1, j=6, exchange (i ++ até encontrar elemento >= pivô, j-- até encontrar elemento <= pivô)
2.ª iteração
i=5, j=4, sair do ciclo (i e j cruzaram-se)
Ao terminar, troca do pivô com j

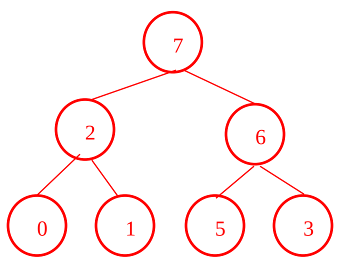
a=[14, 15, 10, 8, 16, 22, 18, 19, 17]
posição do pivô = 4

5. (3.5) Resolva os seguintes exercícios sobre montes (heaps).

a) (1.5) Desenhe a representação em árvore do *max-heap* obtido quando se insere os seguintes elementos num *heap* inicialmente vazio: 7, 2, 5, 0, 1, 6, 3, 8.



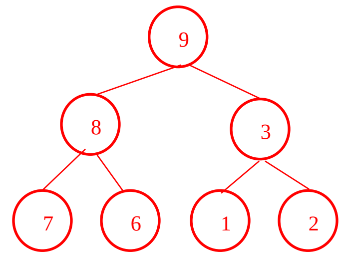
b) (1.0) Desenhe a representação em árvore do *max-heap* obtido quando se remove o maior elemento do *heap* obtido na alínea anterior.



c) (1.0) Indique, justificando, se o seguinte *array* pode ser considerado um *max-heap* (assumindo que a raiz do heap começa na posição 1 do *array*).

	9	8	3	7	6	1	2

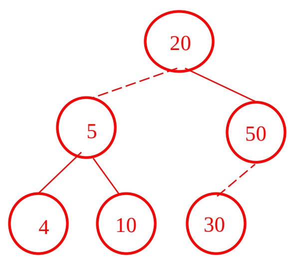
R: Sim, pois como se pode observar na sua representação em árvore, o array respeita a restrição dos max-heap, ou seja um pai é sempre maior ou igual que os seus filhos



6. **(3.0)** Considere uma árvore rubro-negra (*red-black*) inicialmente vazia, onde ligações vermelhas são representadas como linhas a tracejado e ligações negras como linhas contínuas.

a) (2.0) Desenhe a representação da árvore, depois de inseridos os elementos no array indicado abaixo, da esquerda para a direita.

[10, 20, 30, 4, 5, 50]



b) (1.0) Indique a sequência de nós examinados quando o método *max*() é executado sobre a árvore da alínea anterior.

R: 20, 50

7. (3.0) Considere uma tabela de dispersão de dimensão M = 9, com resolução de colisões por dispersão linear, e função de dispersão h1(k) = k mod M.

a) (2.0) Sabendo que inicialmente a tabela se encontra vazia, indique quais os elementos presentes em cada posição do array de chaves e de valores, apos a inserção da sequência de pares <chave , valor> abaixo. Indique uma posição vazia com —.

<4 , "a">, <17 , "b">, <18 , "c">, <0 , "d">, <4 , "e"> , <1 , "f"> , <5, "g">

i	0	1	2	3	4	5	6	7	8
chaves	18	0	1	—	4	5	—	—	17
valores	"c"	"d"	"f"	—	"e"	"g"	—	—	"b"

b) (1.0) Considere agora que o conteúdo da tabela de dispersão é a seguinte:

i	0	1	2	3	4	5	6	7	8
chaves	—	28	29	2	3	—	—	—	—
valores	—	"a"	"b"	"c"	"d"	—	—	—	—

Indique o que muda nos *arrays* de chaves e valores, após a remoção da chave *29* (recorde-se de que a exploração linear utiliza remoção imediata ao invés de remoção preguiçosa). Apenas precisa de assinalar o que é diferente. Indique uma posição vazia com —.

i	0	1	2	3	4	5	6	7	8
chaves			2	3	—				
valores			"c"	"d"	—				