PF — Estruturas de Dados — Prof. Vinícius Gusmão — 2021/2

Cada questão abaixo tem 3 alternativas: (a), (b), (c). Marque a única alternativa correta. Cada resposta correta vale 1,0 (um ponto). Cada resposta errada perde 0,5 (meio ponto). Questões deixadas em branco não ganham nem perdem nada.

Q1: Seja f(n) = 6n(n+7) + 50n +800. *Não é correto* afirmar que

(a) f(n) = O(n log n) (b) f(n) = $O(n^3)$ (c) f(n) = $O(n^2)$

Q2: Quando implementamos uma fila (de *n* elementos) utilizando um array, a inserção de um novo elemento é feita em tempo constante, pois o elemento é colocado no final do array. Já a operação conhecida como "*pop"*, que consiste em processar o primeiro elemento da fila, dando vez a outro elemento,

(a) roda em tempo O(n), demandando um "*shift left*" dos demais elementos
(b) pode ser implementada para rodar em tempo O(log n), usando uma estrutura auxiliar como uma heap, e essa complexidade é a melhor possível
(c) pode ser implementada para rodar em tempo constante

Q3: Sejam as chaves 114, 65, 100, 1234, 148, 61313, 825, 111, 112, 113, e seja h(x) = x mod 50 a função de dispersão utilizada em uma tabela hash (com encadeamento exterior) implementada sobre um array com m = 50 posições. Considerando *operação básica* como sendo uma comparação entre chaves ou uma verificação de lista vazia, o número médio de operações básicas na busca por uma chave *ausente* dessa tabela, supondo que todas as chaves ausentes tem imagens por *h* uniformemente distribuídas no intervalo [0, 49], é:

(a) 6/5 (b) 13/5 (c) 13/50

Q4: Suponha que certo conjunto *A* de *n* chaves está armazenado usando uma das estruturas abaixo. Em que caso é possível retornar a mediana de *A* em tempo O(1)?

(a) lista duplamente encadeada (b) array ordenado (c) heap de mínimo

Q5: Conseguimos inserir/remover elementos em uma pilha mais eficientemente se implementarmos a pilha utilizando

(a) uma árvore binária de busca balanceada
(b) uma tabela hash
(c) uma lista duplamente encadeada

Q6: Para ordenar elementos selecionados *aleatória e uniformemente* de um intervalo conhecido, conseguimos maior eficiência, na prática, usando:

(a) MergeSort (b) BucketSort (c) QuickSort

Q7: Para selecionar o milésimo *maior* elemento presente em um arquivo com n = 10.000.000.000 elementos, a maneira mais eficiente *dentre as opções abaixo* é:

(a) criar uma max heap com os n elementos e remover 1000 vezes a raiz
(b) copiar a lista para um array, ordená-lo, e retornar o elemento de índice k
(c) fazer O(n) inserções em uma min heap de tamanho 1000

Q8: A frase que melhor descreve o objetivo da técnica da *programação dinâmica* é:

(a) evitar programas recursivos, que podem consumir tempo exorbitante
(b) evitar alocação estática de memória sempre que possível
(c) evitar resolver um mesmo subproblema repetidas vezes

Q9: Uma vantagem das tabelas hash sobre arrays com endereçamento direto é

(a) economizar memória
(b) simplicidade de implementação
(c) gastar menos tempo para inserção e consulta

Q10: Dadas as chaves (1, 2, 3, 4) e as respectivas frequências de acesso (9, 1, 3, 2), o número médio de comparações numa busca por uma desses chaves numa árvore binária de busca ótima é:

(a) 7/5 (b) 8/5 (c) 9/5