**Universidade Federal do ABC**
**Bacharelado em Ciência da Computação**
**Algoritmos e Estrutura de Dados I - Prof. Fabrício Olivetti de França e Paulo Henrique Pisani**
**Turma A1/A2 Diurno - Prova A**

**NOME/RA :** .....

**Instruções: responda as seguintes questões na folha de prova. Entregue as duas folhas com nome e RA.**

**Questão 01** (1.0 pto). Um algoritmo de interseção de duas listas ordenadas, ambas de tamanho $n$, requer a operação de busca binária na segunda lista para cada elemento da primeira lista. Qual é a complexidade desse algoritmo em notação $O$?

R.: $O(nlogn)$

**Questão 02** (1.5 pto). Dada a seguinte estrutura de lista simplesmente ligada:

```
typedef struct linked_node linked_node;
struct linked_node {
    int data;
    linked_node *next;
};
```

Crie a função `void muda_lista(linked_node *inicio);` que remove cada dois elementos na sequência e insere a média de seus valores. Exemplo, a lista `2 -> 30 -> 4 -> 5 -> 6` se tornaria `16 -> 4.5 -> 6`. Importante: a função **não** deve criar outra lista, apenas deve modificar a lista passada no parâmetro *inicio* e deve ser implementada na linguagem C.

r.:

```
void muda_lista(linked_node * inicio) {
    linked_node * tmp;

    if (inicio == NULL || inicio->next == NULL) return;

    inicio->data = (inicio->data + inicio->next->data)/2.0;
    tmp = inicio->next;
    inicio->next = inicio->next->next;
    muda_lista(inicio->next);
    free(tmp);
}
```

**Questão 03** (1.5 pto). Dada uma sequência de trens $1, 2, 3, 4$ e $S$ representando a operação de empilhar e $U$ de desempilhar, a sequência $SSUSSUUU$

transforma a ordem 1234 em 2431. Mostre a sequência de operações para transformar a ordem 123456 em 325641.

R.: `SSSUUSSUSUUU`

**Questão 04** (1.0 pto). Um determinado sistema utilizará uma lista. Você precisa decidir entre implementá-la como uma array sequencial ou como uma lista ligada. Qual você escolheria? Quais fatores seriam levados em consideração?

R.: Para os casos em que a lista não cresce ou cresce em "blocos", e a operação mais frequente é a de busca, a array sequencial é mais interessante. Já nos casos em que inserção e remoção de elementos são as operações mais frequentes, a lista ligada é melhor.

**Questão 05** (1.5 pto). Mostre que na busca binária se `right < left`, então necessariamente `right = left - 1`:

```
int * busca_bin (int k, int * x, int n) {
  int left = 0, right = n-1;
  int mid = (left+right)/2;

  while (x[mid] != k)
  {
    if (x[mid] > k) right = mid-1;
    else left = mid+1;

    mid = (left+right)/2;
    if (left > right) return NULL;
  }
  return &x[mid];
}
```

R.: como durante a busca temos que `left <= mid <= right`, temos que ou `right` será atualizado para `mid-1`, ou `left` para `mid+1`. No primeiro caso, se `left = mid`, então `right = left - 1`. Se `left < mid`, `mid-1` ainda será maior ou igual a `left`. O caso em que `left=mid+1` segue o mesmo raciocínio.

**Questão 06** (1.5 pto). Se alterarmos o nosso algoritmo de percurso para visitar os nós na seguinte ordem: `raiz, right, left`. Que relação ela tem com uma das outras formas de percurso?

R.: É o inverso do percurso pós-ordem.

**Questão 07** (1.0 pto). Suponha que temos os números de 1 a 1000 em uma árvore binária de busca e queremos buscar pelo número 363. Qual das seguintes sequências não podem ser a sequência de nós examinados:

a. $2, 252, 401, 398, 330, 344, 397, 363$
b. $924, 220, 911, 244, 898, 258, 362, 363$
c. $925, 202, 911, 240, 912, 245, 363$
d. $2, 399, 387, 219, 266, 382, 381, 278, 363$

e. $935, 278, 347, 621, 299, 392, 358, 363$

R.: (c) e (e)

**Questão 08** (1.5 pto). Implemente um algoritmo na linguagem C que verifica se uma árvore binária é completa.

R.: implementa o algoritmo de altura e verifica a propriedade que toda subárvore esquerda deve ter uma altura no máximo a altura da direita +1.

```
int completa(tree * t){
    if(t->left==NULL && t->right==NULL) return 0;

    if(t->left!=NULL && t->right != NULL)
    {
        int leftheight = completa(t->left);
        int rightheight = completa(t->right);
        /* ou as duas subárvores possuem a mesma altura
         * ou a esquerda é uma unidade mais alta que a direita.
        */
        if((leftheight == rightheight || leftheight == rightheight + 1)
               && leftheight != -1)
            return leftheight+1
    }

    return -1;
}
```