Diogo Akio Balboni Miyake – RA: 21048813 – PROVA C – AED I - UFABC

Obs.: As respostas estão em destaque em amarelo, botei toda a função para ficar mais fácil o teste. Somente a 3 que não consegui implementar em recursão.

Questão 1

```
#include <stdio.h>
/*Função recursiva de encontrar o
maior inteiro*/
int recursive_major_digit(int num) {
int num_1 = num%10;
int num_2 = num/10;
/*caso base */
if (num_2 == 0) {
return num_1;
}
/*chama a recursão para encontrar o
maior digito*/
int num_3 =
recursive_major_digit(num_2);

return num_1 > num_3 ? num_1 :
num_3;
}
/*Função principal*/
int main(void) {
int n;
scanf("%d", &n);
printf("%d\n",
recursive_major_digit(n));
return 0;
}
/*
Entrada
Basta digitar m valor inteiro
como
678950

Saida
9

*/
```

Diogo Akio Balboni Miyake – RA: 21048813 – PROVA C – AED I - UFABC

Questão 2

```c
#include <stdio.h>
#include <stdlib.h>

void recursive_insertion_sort(int* array,
int size)
{
// Caso Base
if (size <= 1)
return;
// Sorteia os n-1 elementos
recursive_insertion_sort( array, size-1 );
// Insere o último elemento na posição
correta
// no array classificado.
int last = array[size-1];
int j = size-2;
/* Move elementos do array, que são
maior que a chave, para uma posição à
frente
de sua posição atual */
while (j >= 0 && array[j] > last)
{
array[j+1] = array[j];
j--;
}
array[j+1] = last;
}

/*Imprime o vetor do insertion*/
void print_array(int* array, int size){
for(int i = 0; i < size; i++){
   printf("%d ", array[i]);
  }
  printf("\n");
}

/*Função principal*/
int main (void){
  int size, value, *array;
  // Insira primeiro a quantidade de itens a
lista, depois os valores na linha de baixo
  while(scanf("%d", &size) != EOF){
   array = malloc (size * sizeof (int));
    for(int i = 0; i < size; i++){
     scanf("%d", &value);
     array[i] = value;
    }
    print_array(array, size);
recursive_insertion_sort(array, size);
    print_array(array, size);
    free(array);
  }
 return 0;
}
/*
Exemplo de inserção
10
5 4 3 6 90 10 45 78 50 12

Saída (consiste do vetor normal e depois
da saida reranjada)
5 4 3 6 90 10 45 78 50 12
3 4 5 6 10 12 45 50 78 90

*/
```

Diogo Akio Balboni Miyake – RA: 21048813 – PROVA C – AED I - UFABC

Questão 3

```
#include <stdio.h>
#include <stdlib.h>

struct cel {
int chave;
struct cel *prox;
};
typedef struct cel no;

struct lista {
no* inicio;
};
typedef struct lista lista;

no* menorElemento (lista* L){
L = L->prox;
int menor_valor = L->chave;
while (L != NULL) {
if (menor_valor > L->chave) {
menor_valor = L->chave;
}
L = L->prox;
}
return menor_valor;
}
```

Diogo Akio Balboni Miyake – RA: 21048813 – PROVA C – AED I - UFABC

Questão 4

```
#include <stdio.h>
#include <stdlib.h>

// Estruturas
struct node{
int key;
struct node *left, *right;

};
typedef struct node Node;
// estrutura para usar um booleano
typedef enum {false, true} bool;


int is_binary(Node *root) {
return (!root->right && !root->left)
|| (root->right && root->left &&
is_binary(root->left) && is_binary(root-
>right));
}


//Criação da ABB

Node *newNode(int item){
Node* temp =
(Node*)malloc((sizeof(Node)));
temp->key = item;
temp->left = temp->right = NULL;
return temp;
}
// imprime inordem
void inOrder(Node *root)
{
if (root != NULL)
{
inOrder(root->left);
printf("%d ", root->key);
inOrder(root->right);
}
}
// imprime pre ordem
void preOrder(Node *root)
{
if (root != NULL)
{
printf("%d ", root->key);
preOrder(root->left);
preOrder(root->right);
}
}
//imprime pos ordem
void posOrder(Node *root)
{
if (root != NULL)
{
posOrder(root->left);
posOrder(root->right);
printf("%d ", root->key);
}
}

//inserção dos nós na ABB

Node* insert(Node* no, int key){
//Se esta vazio retorna new Node
if(no == NULL)
return newNode(key);
// senãão vai inserir o no na esquerda
if (key<no->key)
no->left = insert(no->left, key);
//senao insere na direita
else if(key > no->key)
no->right = insert(no->right, key);
//retorna no do ponteiro
return no;
}

// busca

Node* search(Node *root, int key)
{
if(root == NULL || root->key == key)
//if root->data is x then the element is
found
return root;
else if(key > root->key ) // x is
greater, so we will search the right
subtree
return search(root->right, key);
else //x is smaller than the data, so we
will search the left subtree
return search(root->left,key);
}


// min value
```

```
Node* minValueNode(Node* node){
Node* current = node;
// loop para encontrar o menor item
while(current && current->left != NULL)
current = current->left;
return current;
}


// deletar no
Node* deleteNode(Node* root, int key)
{
// caso base
if(root == NULL){
return NULL;
}
//if(search(root,key)) return NULL;
// Se a chave a ser excluída for menor que a chave da raiz, então fica na sub-árvore esquerda
if (key < root->key){
root->left = deleteNode(root->left, key);
}
// Se a chave a ser excluída for maior que a chave da raiz, então está na sub-árvore direita
else if (key > root->key){
root->right = deleteNode(root->right, key);
}
// se a chave é igual à chave da raiz, então este é o nó para ser deletado
else{
// Nos com somente uma folha ou sem folhas
if (root->left == NULL){
Node* temp = root->right;
free(root);
return temp;
}
else if(root->right == NULL){
Node* temp = root->left;
free(root);
return temp;
}
// nó com dois filhos: obtenha o sucessor inorder (menor na subárvore direita)
Node* temp = minValueNode(root->right);
// usa uma copua co conteudoo do sucessor inordem para este nó
root->key = temp->key;
// deleta o sucessor inordem
root->right = deleteNode(root->right, temp->key);
}
return root;
}

// quantidade de nós

int countNodes(Node* root) {

if (root == NULL)
return 0;
return countNodes(root->left) + countNodes(root->right) + 1;
}

// altura

int height(Node* root){
int u, v;
if(root == NULL)
return -1;
u = height(root->left);
v = height(root->right);
if (u>v)
return u+1;
else
return v+1;
}
int height_v2(Node *root){
if (root == NULL)
return 0;
int height_left = height_v2(root->left);
int height_right = height_v2(root->right);
if (height_left > height_right)
return (height_left + 1);
else
return(height_right + 1);
}
```