Nome: Matheus Goulart Ranzani
RA: 800278

Prova 2

Questão 1

void insere(Node * R, int Ch);
/* esta função deve inserir o elemento Ch na ABB R, caso o elemento já não estiver na árvore. */

```c
void insere(Node* R, int Ch) {
    if (R == NULL) {
        Node* novo = (Node*) malloc(sizeof(Node));

        novo->esq = NULL;
        novo->dir = NULL;
        novo->chave = Ch;

        R = novo;
        novo = NULL;
    } else {
        if (Ch == R->chave) {
            return;
        } else if (Ch < R->chave) {
            insere(R->esq, Ch);
        } else {
            insere(R->dir, Ch);
        }
    }
}
```

Questão 2

a) int getAltura(Node * R); //retorna a altura da árvore de raiz R.

```c
int getAltura(Node* R) {
    if (R == NULL) {
        return 0;
    } else {
        int altura_esq = getAltura(R->esq);
        int altura_dir = getAltura(R->dir);

        if (altura_esq > altura_dir) {
            return altura_esq + 1;
        } else {
            return altura_dir + 1;
        }
    }
}
```

b) A eficiência de tempo dessa função é da ordem de O(n).

Questão 3

a) void ordena(int * V, int N); /* ordena o vetor V de tamanho N */

```
// Ordenação por Bolha/Troca
void ordena(int* V, int N) {
    int aux;

    for (int i = 0; i < N; i++) {
        for (int j = 1; j < N; j++) {
            if (V[j - 1] > V[j]) {
                aux = V[j];
                V[j] = V[j - 1];
                V[j - 1] = aux;
            }
        }
    }
}
```

b) O algoritmo implementado chama-se Bubble Sort (ordenação por bolha/troca). Esse algoritmo, em termos de eficiência de tempo, é da ordem de $O(n^2)$.

Questão 4

bool IsHeap(int * V, int LastPosition);
/* Verifica se o vetor V é um um heap-binário-de-máximo, retornando true caso sim, e false caso não. */

```
bool isHeap(int* V, int LastPosition) {
    int n = LastPosition + 1; // Total de elementos no vetor

    for (int i = 0; i <= (n - 2) / 2; i++) {
        // Se o filho da esquerda for maior, não é Heap
        if (V[fesq(i)] > V[i]) {
            return false;
        }

        // Se o filho da direita for maior, não é Heap
        if (i * 2 + 2 < n && V[fdir(i)] > V[i]) {
            return false;
        }

        return true;
    }
}
```