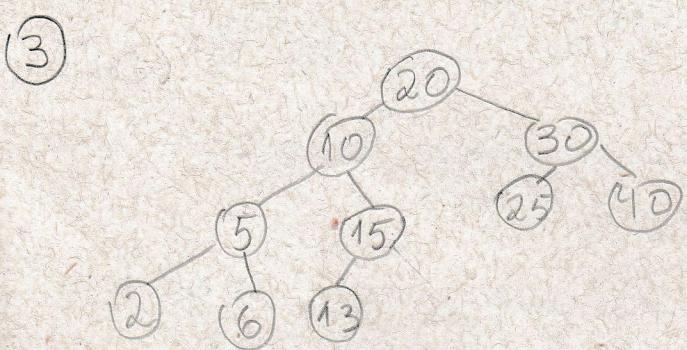


- ① a) A complexidade é $n \log^2 n$, porque:
- O primeiro for tem complexidade $O(\log n)$, porque o contador do for está sendo multiplicado por 2;
- O segundo for tem complexidade $O(\log n)$, porque o contador do for está sendo dividido por 2;
- (ou seja, nos dois casos acima, os problemas estão sendo divididos em problemas menores)
- O terceiro for tem complexidade $O(n)$, porque o contador do for está sendo incrementado de 2.
- Como um for está dentro do outro, multiplica as complexidades.
- b) no primeiro for, a complexidade é $O(\log n)$, porque a operação feita sobre o contador é o shift de 2 bits;
- no segundo for a complexidade é $O(n)$ porque está incrementando o contador;
- no terceiro for também, porque está comando 2 ao contador;
- no quarto for a complexidade é $O(\log n)$, porque a operação sobre o contador é multiplicação por 2.
- Como os dois últimos for estão com a mesma intensidade, soma complexidade delas; de resto um for está dentro do outro, então multiplica as complexidades.
- então ficamos com $\underbrace{(\log n + n)}_{\text{o } \log n \text{ pode ser desprezado}} \cdot n \cdot \log n \Rightarrow O(n^2 \log n)$

(2) Essa função só procura primeiro nos nós à esquerda e depois apenas nos nós à direita e retorna. Assim, se for preciso ir à esquerda depois de ir à direita para encontrar a chave, ela não será encontrada.
 As chaves 20 e 50 não seriam encontradas.
 Para a chave 20, a função entra no primeiro while, vai até o 12, entra no segundo while, vai até o 26 e retorna `msdados`.
 Para a chave 50, a função entra no segundo while, vai até o 60 e retorna `msdados`.



ordem de inserção:

20, 10, 30, 5, 15, 25, 40, 2, 6, 13

Pseudocódigo para os itens (a) e (b):

Define struct `com`:
 campo para chave (`int`), campo para posição relativa (`int`), campo para qtd de nós à esquerda (`int`) e à direita (`int`), ponteiro para subárvore à esquerda e ponteiro para subárvore à direita.

Cria árvore e insere os nós;

Início da função para colocar a posição relativa da chave: recebe um nó;
 Cria variável estática `p=0` # p identificar a posição da chave quando entrar recursivamente.

Caso base: se nó = null, retorna;

Chama recursivamente a função com subárvore esquerda;

Incrementa `p` e coloca no campo posição do nó;

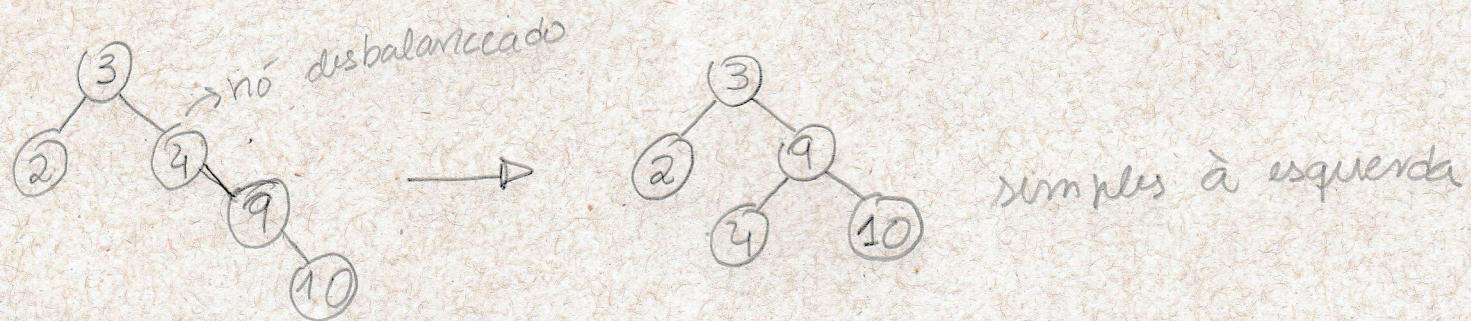
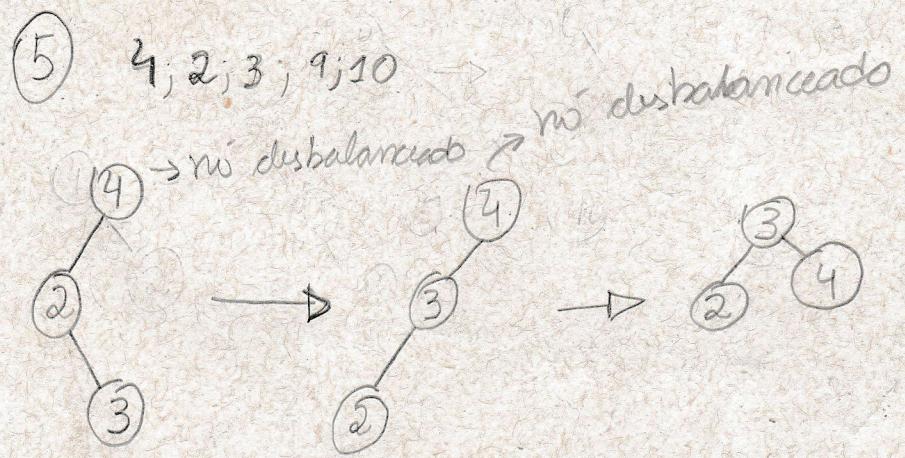
Chama recursivamente a função com a subárvore direita

③ (CONTINUACAO)

início da função para colocar a qtd de subárvores à esquerda e à direita da arvore: recebe um nó;

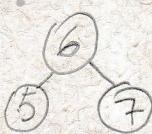
se nó = null, retorna 0; # caso base

diz a subárvore à esquerda obtendo a qtd de nós à esquerda;
diz a subárvore à direita obtendo a qtd de nós à direita;
retorna qtd de nós à esq + qtd de nós à dir + 1;
(código em c abaixo):



- 6) Início da função de exclusão: recebe árvore e chave;
- se raiz = null, retorna a raiz;
 - se chave < raiz, entra recursivamente na subárvore esquerda;
 - se não se chave > raiz, entra recursivamente na subárvore direita;
 - senão: # chave = raiz
 - retira esse nó da árvore;

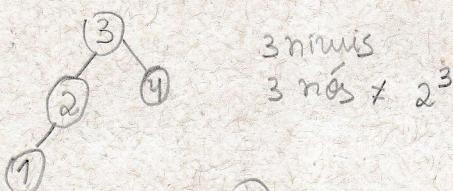
⑦ a) falso. contraexemplo:



nível da raiz = 0
altura da árvore = 1

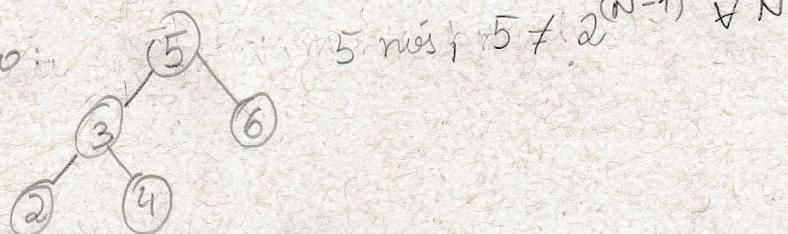
b) verdadeiro, pois o nível do é a altura dele mesmo

c) falso. contraexemplo:



3 níveis
3 nós $\neq 2^3$

d) falsa. contraexemplo:



5 nós; 5 $\neq 2^{(N-1)} \wedge N$

```

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

struct nodo {
    int info;
    int pos;
    int nEsq, nDir;
    struct nodo* esq;
    struct nodo* dir;
};

typedef struct nodo Nodo;

Nodo* insereNo(Nodo* no, int val) {
    Nodo* aux;
    if (!no) {
        aux = (Nodo*) malloc(sizeof(Nodo));
        aux->esq = NULL;
        aux->dir = NULL;
        aux->info = val;
        return aux;
    }
    if (no->info > val) {
        no->esq = insereNo(no->esq, val);
    }
    else if (no->info <= val) {
        no->dir = insereNo(no->dir, val);
    }
    return no;
}

void acrescPos(Nodo* no) {
    static int p = 0;
    if (!no) { return; }
    acrescPos(no->esq);
    no->pos = ++p;
    acrescPos(no->dir);
}

int acresQtde(Nodo* no) {
    if (!no) { return 0; }
    no->nEsq = acresQtde(no->esq);
    no->nDir = acresQtde(no->dir);
    return no->nEsq + no->nDir + 1;
}

void pre_ordem(Nodo* arvore) {
    if (!arvore) { return; }
    pre_ordem(arvore->esq);
    printf("valor %d pos %d esq %d dir %d\n", arvore->info, arvore->pos,
arvore->nEsq, arvore->nDir);
    pre_ordem(arvore->dir);
}

int main()
{

```

```
Nodo *arvore = NULL;
int vet[] = {20,10,30,5,15,25,40,2,6,13};
int i;
for (i = 0; i < 10; i++) {
    arvore = insereNo(arvore, vet[i]);
}
acrescPos(arvore);
acresQtde(arvore);
pre_ordem(arvore);
printf("nao deu merda");

return 0;
}
```