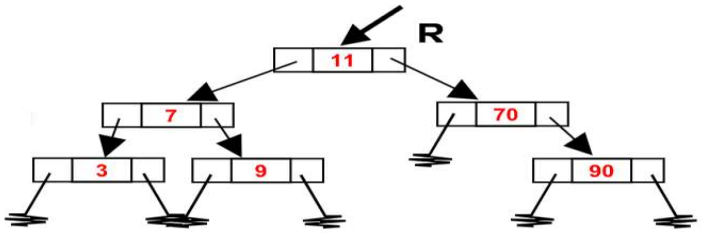
Giullio Emmanuel da Cruz Di Gerolamo

RA: 790965

Prova P2

Questão 1 (2,5 pontos) Considere uma ***Árvore Binária de Busca*** (ABB), de raiz R, implementada com alocação encadeada e dinâmica de memória, conforme os diagramas abaixo. A Árvore não contém elementos repetidos.



Considere ainda que o nó de a árvore binária de busca tenha a seguinte declaração:

```
typedef struct node {
        int chave;
        struct node *esq;
        struct node *dir;
} Node;
```

Implemente a operação:

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

Resposta:

```
Node* buscar(Node*R, int Ch){
    while(R){
        if(Ch < R->chave)
            R = R->esq;
        else if(Ch > R->chave)
            R = R->dir;
        else
            return R;
    }
    return NULL;
}

void insere(Node **R, int Ch){
    if(*R == NULL){
        *R = malloc(sizeof(Node));
        (*R)->chave = Ch;
        (*R)->esq = NULL;
        (*R)->dir = NULL;
    }
    else{
        if(Ch < (*R)->chave)
            insere(&((*R)->esq), Ch);
        else
            insere(&((*R)->dir), Ch);
    }
}
```

Na implementação(main):

```
Node *busca = NULL;
busca = buscar(R, Ch);
if(busca)
   printf("\n\tValor já se encontra na arvore");
else
   insere(&R, Ch);
```

Questão 2 (2,5 pontos) Considere uma ***Árvore Binária de Busca*** (ABB), de raiz R, implementada conforme os diagramas e declarações indicados na questão 1.

a) Escreva uma função que calcule a altura de uma árvore de raiz R. Por convenção, considere que a altura de uma árvore vazia é zero, e que a altura da árvore do diagrama a esquerda na Questão 1 é 3.

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

b) Qual é a ordem de eficiência de tempo dessa função, em termos de número de comparações?

Resposta:

a)

```
int GetAltura(Node *R){
    if(R == NULL){
        return -1;
    }
    else{
        int esquerda = GetAltura (R->esq);
        int direita = GetAltura (R->dir);
        if(esquerda > direita)
            return esquerda + 1;
        else
            return direita + 1;
    }
}
```

b)

O(n)
Pois todos os nós são acessados Nessa função.

Questão 3 (2,5 pontos)

a) Implemente a função abaixo, que ordena um vetor V de inteiros, de tamanho N, colocando os elementos em ordem crescente. Use para isso um dos algoritmos estudados na disciplina – ordenação por Inserção, Seleção ou Bolha.

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

b) Qual é nome do algoritmo que você implementou? Qual é a ordem de eficiência de tempo do mesmo, em termos de número de comparações?

Resposta:

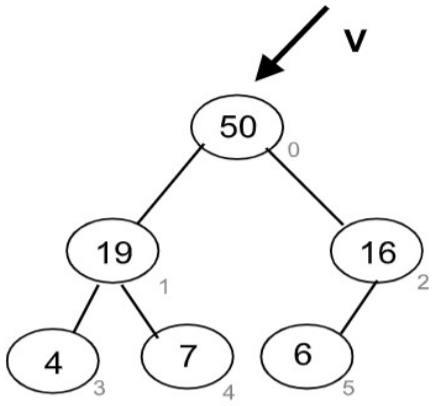
a)

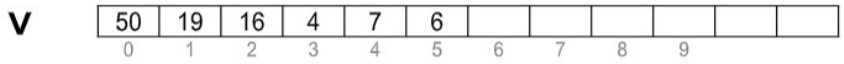
```
void ordena(int* V, int N)
{
   int i, j;
   for (i=N-1; i>0; i--)
      int troca = 0;
      for (j=0; j<i; j++){
         if (V[j]>V[j+1]) {
            int temp = V[j];
            V[j] = V[j+1];
            V[j+1] = temp;
            troca = 1;
         }
   if (troca == 0)
   return;
   }
}
```

b)

O algoritmo de ordenação utilizado foi do tipo bolha ou troca (bubble sort), e a ordem de eficiência de tempo é $O(n^2)$.

Questão 4 (2,5 pontos) Em um ***Heap-Binário-de-Máximo***, o elemento que está em um determinado ***nó*** da árvore tem valor maior ou igual do que o valor de seus ***filhos*** direito e esquerdo. Entre os nós ***irmãos***, não há necessariamente uma ordenação, como mostra o diagrama, a seguir.





Escreva uma função que verifica se um vetor V[0 .. LastPosition] é um heap-binário-de-máximo, ou não. A função recebe como parâmetro o inteiro LastPosition, que indica a última posição do vetor que efetivamente contém elementos.

```
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.  */
```

Use as definições abaixo em sua função:

```
#define pai(i) ((i – 1) / 2 )
#define fesq(i) (i * 2  +  1)
#define fdir(i) (i * 2  +  2 )
```

Resposta:

```
bool isHeap(int *V,  int LastPosition)
{
    int n = LastPosition;
    for (int i=0; i&lt;=(n-2)/2; i++)
    {
        if (arr[2*i +1] &gt; arr[i])
                return false;
        if (2*i+2 &lt; n &amp;&amp; arr[2*i+2] &gt; arr[i])
               return false;
    }
    return true;
}
```