20230001178

8,25

1-) - a -

```
queve GetQueve(sent st){
    q *aux;          ñ pode ser ponteiro
    queve *Fila;
    Fila->head = fila->tail = NULL;   ← ñ existe em C (atribuição múltipla)
                                      → perigoso

    For(st.head; st.head != NULL; st.head = st.head->next){
        aux = (q*)malloc(sizeof(q));  ✓
        aux->n = st.head->n;  ✓
        aux->next = NULL;  ✓
        if(Fila->head == NULL){
            Fila->head = aux;  ✓
            Fila->tail = aux;  ✓
        }
        else{
            Fila->tail->next = aux;        7,5
            Fila->tail = aux;
        }
        return Fila;
}
```

1-) — b —

```
stack getStack(sent st){
    S *aux;
    stack *pilha;
    pilha->top = NULL;
    for(st.tail; st.tail != NULL; st.tail = st.tail->prev){
        aux = (s *)malloc(sizeof(s));
        aux->n = st.tail->n;
        aux->next = pilha->top;
        pilha->top = aux;
    }
    return pilha;
}
```

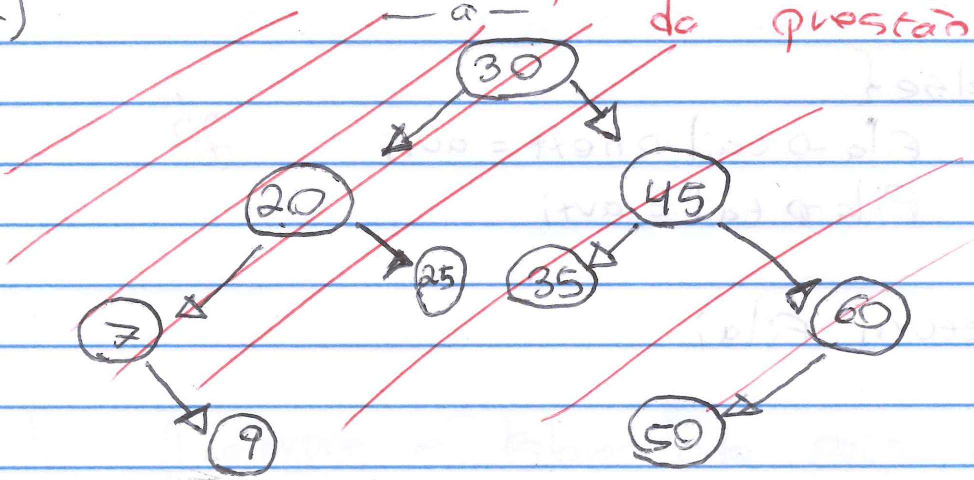
ñ pode ser ponteiro

teria que alocar memória!

2,5

Favor considerar a árvore que eu desenhei no final da questão.

2-) — a —



— b —

~~A árvore é balanceada pois o nodo mais profundo da primeira sub árvore à esquerda da raiz possui a mesma altura que o nodo mais profundo da primeira subárvore à direita da raiz.~~ Resolução da b na última página

2- — c —

```
void printTree (node *root){
    if (root == NULL){
        return;
    }
    printf("%d",root->n);
    printTree (root->left);
    printTree (root->right);
}
```

a aplicação n bate com o algoritmo

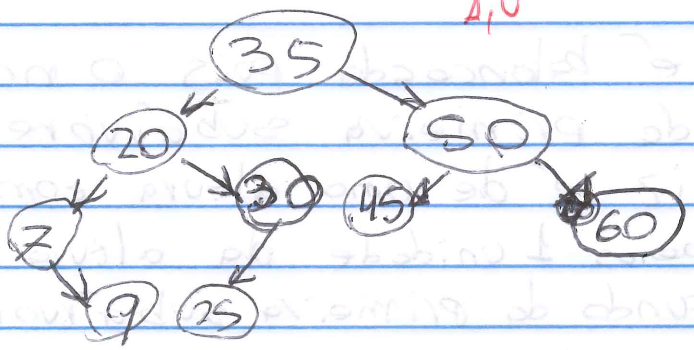
0,25

A função que realiza o percurso da árvore percorre de nodo a nodo pela esquerda até encontrar o NULL quando não pode ir mais para a esquerda ela retorna e vai à direita. Caso o ponteiro da direita for NULL, a função retorna novamente, e vai fazendo isso até percorrer toda a árvore

A saída dessa função pré-ordem é:
30 20 7 9 30 25 50 45 60

2- — a. —

4,0



2b →

3-)

$1 . c_1 + \log i . c_2 + \log i . c_3$ 10

A função Xmen é da ordem $O(\log_2 i)$.

2 b-) A árvore é balanceada pois o nodo mais profundo da primeira subárvore à esquerda da raiz é de uma altura com uma diferença de apenas 1 unidade da altura do nodo mais profundo da primeira subárvore à direita da raiz.

10

```
void push(pilha *pilha, item item){
    node *aux;
    aux = (node*)malloc(sizeof(node));
    aux->item = item;
    aux->next = pilha->top;
    pilha->top = aux;

}
void pop(pilha*pilha, item item){
    node *aux;
    *item = pilha->top->item;

    aux = pilha->top;
    pilha->top = pilha->top->next;
    free(aux);
}
void limparPilha(pilha *pilha){
    node *aux;
    while(pilha->top!=NULL){
        aux = pilha->top;
        Pilha->top = pilha->top->next;
        Free(aux);
    }
}
```

```
node *insertNode(node *root, node *new){
    if (root ==NULL){
        ~~root = new;~~
        return new;
    }
    if(root->valor > new->valor){
        root->left = insertNode(root->left, new);
    }
    else{
        root->right = insertNode(root->right, new);
    }
    return root;
}
```

```
int buscaBin(int*vet, int inicio, int fim, int Key)
  int meio;
  if(inicio>fim) return -1;
  meio =(inicio + fim)/ 2;
  if (vet[meio]==Key) return meio;
  if(Key > vet[meio]) {
     return buscaBin(vet, meio+1, fim, Key);
  }
  else{
     return buscaBin(vet, inicio, meio-1, Key)
  }
}
```

```
int buscaBin(int*vet, int len, int Key){
  int ini=0, fim = len, meio=(ini+fim)/2;
  while(ini<=fim){
    if(vet[meio]==Key){
      break;}
    if(vet[meio]<Key){
      ini~~cio~~= meio +1;}
    else{
      fim = meio -1;}
    meio =(ini+fim)/2;
  }
  if(ini>fim){
    return -1;}
  else{
    return vet[meio];}
}
```

```
void enQueue(fila*fila, int num){
    node *aux;
    aux = ~~(fila*)~~ malloc(sizeof(~~fila~~));
    aux->num = num;
    aux->next = NULL;
    if(fila->head == NULL){
        fila->head = aux;
        fila->tail = aux;
    }
    else{
        fila->tail->next = aux;
        fila->tail = aux;
    }
}
```

```
void deQueue(fila *fila, int *item){
    node *aux;
    *item = fila->head->num;
    aux = fila->head;
    if(fila->head == fila->tail){
        fila->head = NULL;
        fila->tail = NULL;
    }
    else{
        fila->head = fila->head->next;
    }
    free(aux);
}
```

```
int isEmpty(fila *fila){
    return (fila->head == NULL);
}
```

20230001178

```
void FreeAll(fila *fila){
    node *aux;
    while(fila->head != NULL){
        aux = fila->hood;
        fila->head = fila->head->next;
        Free(aux);
    }
    fila->tail = NULL;
}
```

- enQeve(&fila, num);
- deQueue(&fila, &item);

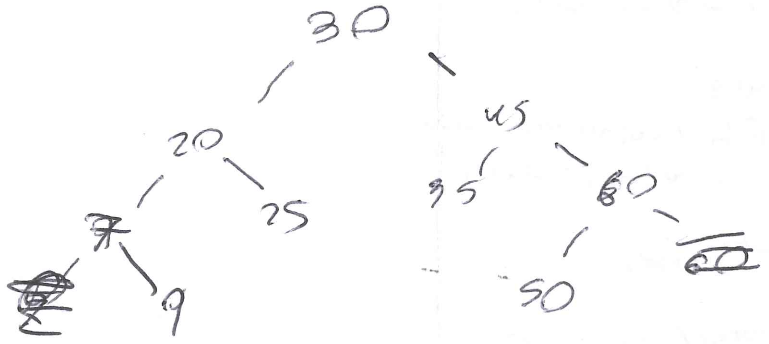
```
void printTree(node *root){
    if(root == NULL){
        return;
    }
    printf // ordem inserção
    printTree(root->left);
    printf // ordem crescente
    printTree(root->right);
    printf// ordem inversa
}
```

```
Queue GetQueue(sent st){                        Queue *fila;
  Q *aux;                                       Fila->head = fila->tail = NULL;
  aux = (Q *)malloc(sizeof(q));
  for(st.head; st.tail != NULL; st.head = st.head->next){
       aux->n = st.head->n
       aux->next = NULL;
       if (fila->head == NULL){
          fila->head = aux;
          fila->tail = aux;
       }
       else{
          Fila->tail->next = aux;
          Fila->tail = aux;
       }
  }    return fila;
```

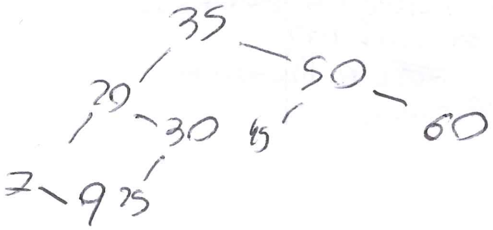
```
Stack GetStack(sent st){
     S*aux;
     Stack *Pilha
     Pilha.top = NULL
                       tail
     for(st.tail; st.~~head~~ != NULL; st.tail = st.tail->prev){
         aux = (S *)malloc(sizeof(S));
         aux->n = st.tail->n
         aux->next = ~~NULL~~; pilha->top;
         pilha->top = aux;
     }
     return pilha;
}
```

~~35~~ ~~20~~ ~~7~~ ~~9~~ ~~30~~ ~~25~~ ~~50~~ ~~45~~ ~~60~~

7 9 20 25 30 35 45 50 60



30 -> 20 - 7 -> 9 - 25 - 45 - 35 - 60 - 50



30 20 7 9 30 25

50 45 60