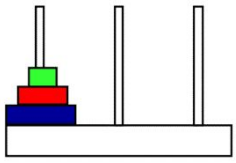
GABARITO SEMANA 3
Monitor: Christian Delgado Polar

0.5 Pontos

**Questão 1**
Torres de Hanói

Considere uma torre de Hanói com três hastes e *n* discos de diferentes tamanhos. O jogo consiste em mover os discos desde a primeira haste até a última. Em cada passo é permitido mover apenas um disco e um disco só pode ser colocado sobre a base ou sobre um disco maior.



O algoritmo a seguir promete imprimir os movimentos necessários para resolver o jogo. Prove por indução a corretude do mesmo, isto é prove que o algoritmo faz o que ele promete.

```
1  void Hanói(char ori, char dst, char aux, int n) {
2          if(n == 1) {
3                  System.out.print("Move de "+ori+ “para "+dst);
4          }
5          else {
6                  Hanói(ori, aux, dst, n-1);
7                  Hanói(ori, dst, aux, 1);
8                  Hanói(aux, dst, ori, n-1);
9          }
10 }
```

**Solução**

A. **Base da Indução:**
Para n=1, a linha 3 do algoritmo é executada e imprime que move o único disco desde a primeira haste até a última. Assim, o algoritmo está correto para n=1.

B. **Passo Indutivo:**
Para n>=2 as linhas 6, 7 e 8 serão executadas. Por hipótese de indução supomos que o algoritmo funciona para n-1.

A linha 6 do algoritmo imprime os movimentos necessários para mover n – 1 discos da primeira haste “ori” até a segunda haste “aux” corretamente pela hipótese de indução, deixando a terceira haste “dst” vazia.

A linha 7 imprime o movimento necessário para mover o único disco (o maior) que sobrou na hasta “ori” até a última haste “dst”, deixando a primeira haste “ori” vazia.

A linha 8 imprime os movimentos necessários para mover n – 1 discos da haste “aux” até a última haste “dst” corretamente pela hipótese de indução, isto é sobre o disco maior.

Assim, após a execução das linhas 6, 7 e 8, os n discos estão na haste “dst”.

Portanto o algoritmo está correto para qualquer n>=1.

**Observação:** Note que na questão 1 não foi solicitado demonstrar que a quantidade mínima de movimentos é $2^n$ - 1, nem foi solicitado calcular o consumo de tempo do algoritmo.

0.5 Pontos

**Questão 2**

Considere o pseudocódigo do algoritmo de ordenação por inserção:

```
1 ORDENA-POR-INSERÇÃO (A, n)
2       para j ← 2 até n faça
3               chave ← A[j]
4               i ← j − 1
5               enquanto i ≥ 1 e A[i] > chave faça
6                       A[i + 1] ← A[i]
7                       i ← i − 1
8               A[i + 1] ← chave
```

( F ) Depois da primeira iteração do algoritmo ORDENA-POR-INSERÇÃO para o vetor A = [4,6,2,10,9,8,15,18,12,5] e n=10, o vetor fica assim A = [6,4,2,10,9,8,15,18,12,5].

( V ) Em algum passo do algoritmo ORDENA-POR-INSERÇÃO para o vetor A = [4,6,2,10,9,8,15,18,12,5] e n=10, o vetor fica assim A = [2,4,6,8,9,10,12,15,18,5]

( F ) Em qualquer iteração do algoritmo ORDENA-POR-INSERÇÃO o sub-vetor da esquerda da posição "j" está desordenado.

A quantidade de vezes que são executadas as linhas 6 e 7 do algoritmo ORDENA-POR-INSERÇÃO quando o vetor tem "n" elementos e está ordenado de forma crescente é: (Escolha apenas uma alternativa)

( ) n - 1

( X ) 0

( ) n

( ) n/2