LISTA 1
Francisco Braz de Souza

1-)
Base: $(1.1)^C = 1^C \cdot 1^C$
$1 = 1$

Hipótese de indução: Supomos que $(a_1 \cdot a_2 \cdot a_3 \ldots a_K)^C = a_1^C \cdot a_2^C \cdot a_3^C \ldots a_K^C$

Onde queremos chegar: $(a_1 \cdot a_2 \cdot a_3 \ldots a_{K+1})^C = a_1^C \cdot a_2^C \cdot a_3^C \ldots a_{K+1}^C$

Processo indutivo:
$(a_1 \cdot a_2 \cdot a_3 \ldots a_K \cdot a_{K+1})^C = (a_1 \cdot a_2 \cdot a_3 \ldots a_K)^C \cdot a_{K+1}^C$

$(a_1 \cdot a_2 \cdot a_3 \ldots a_K)^C$ **é igual a nossa hipótese de indução**

$a_1^C \cdot a_2^C \cdot a_3^C \ldots a_K^C \cdot a_{K+1}^C = a_1^C \cdot a_2^C \cdot a_3^C \ldots a_{K+1}^C$

2-)
Base:
$a1 = 1$
$a2 = 2$
$a3 = a_{3-2} + 2.a_{3-1} = 1 + 2 \cdot 3 = 7$

Hipótese de indução: Supomos que a_i é ímpar para todo inteiro i, $1 \leq i < K$

Caso geral: Se HI é verdade, então a afirmação é válida para K, ou seja, a_K é ímpar

Processo indutivo:
$a_K = a_{K-2} + 2.a_{K-1}$

$1 \leq K-2 < K$, logo a_{K-2} é ímpar
$1 \leq K-1 < K$, logo a_{K-1} é ímpar

Como uma multiplicação entre um número ímpar e um número par resulta em um número par, $2.a_{K-1}$ é um número par. ***(Prova auxiliar 1)***

Então, temos que:

a_K = ímpar + par

Sabemos que a soma de um número ímpar número par resulta em um número ímpar (*Prova auxiliar 2*), então a_K é ímpar.

Provas auxiliares:

Prova 1:

a = 2k

b = 2k + 1

a . b = 2k . (2k + 1)

= $4k^2$ + 2k

= 2 . (2k + k) , *podemos dizer que (2k + k) é k*

= 2k

Prova 2:

a = 2k

b = 2k + 1

a + b = 2k + 2k + 1

= 4k + 1

= 2 . (2k) + 1, *podemos dizer que (2k) é k*

= **2k + 1**

3-)

Não consegui fazer por indução

Pseudocódigo:

Obs - "ini" começaria sendo 1 e "fim" seria o total de elementos

```
função inverte(S, ini, fim)
início
        se ini ≥ fim retorna vazio
        senão
                aux := head(S)
                S[ini]  := S[fim]
```

```
        S[fim]  := aux
        inverte(S, ini + 1, fim - 1)
fim
```

4-)
Cada nó da nossa árvore terá uma nó à esquerda e um nó à direita que acessamos através do “.” (ponto);

Caso base: T = vazio, então altura = -1

Hipótese de indução: Supomos que sabemos calcular o problema para qualquer subárvore de altura K, vazio ≤ K < A, sendo A a altura de alguma árvore maior do que vazio, (A > vazio);

Caso geral: Para calcular a altura de qualquer subárvore de altura K, vazio ≤ K < A, usaremos mAlt, já que ele retornará a maior altura entre as subárvores da esquerda e direita. Por fim somaremos mais 1, para contabilizar o nó raiz. Ou seja, alturaArv = mAlt(alturaArv(T.esq), alturaArv(T.dir)) + 1.

Pseudocódigo:

```
função alturaArv(T)
inicio
    se T = vazio retorna -1
    senão retorna mAlt(alturaArv(T.esq), alturaArv(T.dir)) + 1
fim
```

5-)
Temos um vetor que chamaremos de X e ele terá um tamanho n, X[1...n];

Caso base: n = 1, menor = X[1]

Hipótese de indução: Supomos que sabemos calcular o problema para um valor de n-1, sendo n - 1 > 0;

Caso geral: Se a nossa HI for verdade, então podemos utilizar chamadas recursivas, passando para nossa função um valor de n-1. Supondo que temos uma função chamada menorEntreDois que retorna o menor valor entre dois números podemos fazer: menorElemento = menorEntreDois(X[n], menorElemento(X[1...n], n-1);

Pseudocódigo:

```
função menorEntreDois(a, b)
início
    se a < b retorna a
    senão retorna b
fim


função menorElemento(X[1...n], n)
início
    se n = 1 retorna X[1]
    senão
        y := menorEntreDois(V[n], menorElemento(X[1...n], n-1)
        retorna y
fim
```