Indução:

A soma dos n primeiros números naturais pode ser representada pela seguinte fórmula?

$S(n) = n*(n+1)/2$

$S(1) = 1$ verdadeiro (caso base)

$S(n-1) = (n-1)*((n-1)+1)/2$ H.I. é verdade

$S(n-1) = (n-1)*(n)/2$ H.I. é verdade

$S(n) = S(n-1) + n$ (a diferença entre S(n-1) e S(n))

$S(n) = (n-1)*(n)/2 + n$

$S(n) = (n^2-1*n)/2 + n$

$S(n) = (n^2-1*n+2*n)/2$

$S(n) = (n^2 + n)/2$ => está demonstrado que $S(n) = (n^2 + n)/2$ é a fórmula para a soma dos n primeiros números naturais.

$2^n = 2^{n-1} + 2^{n-2} + \ldots + 2^2 + 2^1 + 2^0 + 1$

$ST(n) = 2^n$

Passo base: $ST(1) = 2$ confirmado valor esperado: $2^0 + 1 = 2$

$ST(n-1) = 2^{n-1}$ (por H.I. isso é verdadeiro)

$ST(n) = ST(n-1) + 2^{n-1}$ (a diferença entre ST(n-1) e ST(n))

$ST(n) = 2^{n-1} + 2^{n-1}$

$ST(n) = 2^1 * 2^{n-1}$

$ST(n) = 2^{(n-1)+1}$

$ST(n) = 2^n$ provamos o passo indutivo, consequente (já havíamos provado o passo base) demonstramos que ST(n) é igual a 2^n

Queremos calcular m*n utilizando apenas adições

n = 0 -> caso base, resultado da multiplicação igual a zero

Mult(0) = 0 (se n == 0 então retornar 0) 0 = m*0

Mult(n-1) funciona por H.I.

m*n = m*(n-1) + m -> o que falta para resolver o problema a partir de mult(m,n-1): falta é somar m.

```
int mult(int m, int n){
    if (n==0) return 0;                    // caso base
    return m + mult(m, n-1);               // H.I.
}
```

Implementação iterativa da função de multiplicação por somas sucessivas.

m*n

```
int multIterativa(int m, int n){
    int resp = 0;
    for (int i=0;i<n;i++) resp += m;
    return resp;
}
```

fat(n)

n = 0

n = 1

por H.I. fat(n-1)

fat(n) = fat(n-1) * n

```
int fat(int n){
    if (n==0 || n==1) return 1;          // caso base
    return n * fat(n-1);                 // recursiva
}
```

T(n) = 0 n<=1

T(n) = 1*T(n-1) + 1 n>1

```
int fatIterativo(int n){
    int resp = 1;
    for(int i=2;i<=n;i++) resp*=i;
    return resp;
}
```

NúmeroDeMultiplicaçoes(n) = n-1

Fibonacci

```
int fib(int n){
    if (n <= 1) return n;              // caso base
    return fib(n-1) + fib(n-2);
}
```

$T(n) = 0 \qquad n<=1$

$T(n) = T(n-1) + T(n-2) + 1 \qquad n>1$

```
int fibIterativo(int n){
    if (n<=1) return n;
    int[] temp = new int[n+1];
    temp[0] = 0;
    temp[1] = 1;
    for (int i=2;i<=n;i++) temp[i] = temp[i-1] + temp[i-2];
    return temp[n];
}
```

NúmeroDeSomas(n) = n – 1 válido para n > 0

```
int buscaIterativa(int[] A, int valor, int n){
    for(int i=0;i<n;i++) if(A[i] == valor) return i;
    return -1;
}
```

$NúmeroDeComparações_{MelhorCaso}(n) = 1$ (elemento buscado é o primeiro)

$NúmeroDeComparações_{PiorCaso}(n) = n$ (elemento buscado não existe ou é o último)

caso base n=1

condição de parada: o elemento atual é quem a gente está procurando

sei fazer por H.I.: busca(A, valor, n-1)

O que falta fazer é olhar o n-ésimo elemento (lembrando que a busca para n-1 pode ter encontrado ou não o elemento buscado)

```
int busca(int[] A, int valor, int n){
    if (n==1) {
        if (A[0] == valor) return 0;
        else return -1;
    }else{
        if (A[n-1] == valor) return n-1;
        return busca(A, valor, n-1);
    }
}
```

$T(n) = 1$ p/ $n = 1$

$T(n)_{MelhorCaso} = 1$ p/ $n > 1$ (elemento buscado é o último)

$T(n) = 1$ $n = 1$

$T(n)_{PiorCaso} = 1*T(n-1) + 1$ $n > 1$ (elemento buscado não existe ou é primeiro)

Busca Binário

```
int buscaBinIterativa(int[] A, int valor, int n){
    int ini, meio, fim;
    ini = 0;
    fim = n-1;
    while (ini<=fim){
        meio = (ini+fim)/2;
        if (A[meio] < valor) ini = meio+1;
        else    if (A[meio] > valor) fim = meio-1;
                else return meio;
    }
    return -1;
}
```

$NúmeroDeComparações_{MelhorCaso} = 2$ (elemento buscado é o do meio)

$NúmeroDeComparações_{PiorCaso} = 2*(1+\log_2 n)$ (elemento buscado não está o arranjo)

Caso base: se só restou um número (ou se não restou nenhum)

Condição de parada: o elemento atual (“o do meio” da chamada recursiva atual) é quem eu estou procurando

Por H.I.: busca(A, valor, ini, meio-1) ou busca(A, valor, meio+1, fim)

O que falta é olhar o elemento atual (o do meio):

busca(A,valor, 0, n-1):

olhar para o elemento atual + busca(A, valor, ini, meio-1)

ou olhar para o elemento atual + busca(A, valor, meio+1, fim)

```
int buscaBin(int[] A, int valor, int ini, int fim){
    if (fim<ini) return -1;
    int meio = (ini+fim)/2;
    if (A[meio] < valor) return buscaBin(A, valor, meio+1, fim);
    else    if (A[meio] > valor) return buscaBin(A, valor, ini, meio-1);
            else return meio;
}
```

$T(n) = 0 \quad n = 0$

$T(n)_{MelhorCaso} = 2 \quad n > 0$ (elemento buscado está no meio)

$T(n) = 0 \quad n = 0$

$T(n)_{PiorCaso} = 1*T((n-1)/2) + 2 \quad n > 0$ (elemento buscado não está no arranjo)

```
int buscaBin(int[] A, int valor, int ini, int fim){
    if (ini==fim) {
        if (A[ini] == valor) return ini;
        else return -1;
    }
    int meio = (ini+fim)/2;
    if (A[meio] < valor) return buscaBin(A, valor, meio+1, fim);
    else    if (A[meio] > valor) return buscaBin(A, valor, ini, meio-1);
            else return meio;
}
```

$T(n) = 1 \quad n = 1$

$T(n)_{MelhorCaso} = 2 \quad n > 1$ (elemento buscado está no meio)

$T(n) = 1 \qquad n = 1$

$T(n)_{PiorCaso} = 1*T((n-1)/2) + 2 \qquad n > 1$ (elemento buscado não está no arranjo)