1. Prove que o algoritmo BubbleSort a seguir é correto.

```
1 for i = 0 to n − 2 do
2 |  for j = n − 1 downto i + 1 do
3 |  |  if A[j] < A[j − 1] then
4 |  |  |  swap A[j] and A[j − 1];
5 |  |  end
6 |  end
7 end
```

Algorithm 1: BubbleSort($A[0..n-1]$)

$$T_w(n) = T_b(n) = \sum_{i=0}^{n-2}\left(\sum_{j=i+1}^{n-1} 1\right) =$$

$$= \sum_{i=0}^{n-2}(n\cancel{-1}-(i+1)+\cancel{1}) = \sum_{i=0}^{n-2}(n-i-1)$$

$$= \sum_{i=1}^{n-1} i = \Theta(n^2).$$

Invariante para o laço interno (linha 2-6)
Antes de cada iteração, o subvetor $A[i+1 \ldots n-1]$ são maiores ou iguais a $A[j]$.

Inicialização: Antes da primeira iteração ($j=n-1$) o subvetor $A[j+1 \ldots n-1]$ é vazio e temos a invariante por vacuidade.

Manutenção: Suponha que antes da k-ésima iteração ($j=n-k$) os elementos do subvetor $A[n-k+1 \ldots n-1]$ são $\geq$ que $A[n-k]$. Durante a iteração temos 2 casos:

① $A[n-k] < A[n-k-1]$: Neste caso, $A[n-k-1]$ vai para posição $n-k$, e portanto todos os elementos do subvetor $A[n-k \ldots n-1]$ são $\geq$ que $A[n-k-1]$.

② $A[n-k] \geq A[n-k-1]$: Neste caso, todos os elementos de $A[n-k \ldots n-1]$ são $\geq A[n-k-1]$.

E portanto a invariante está satisfeita em ambos os casos.

Terminação: Ao final da última iteração ($j=i$) todos os elementos do subvetor $A[i+1 \ldots n-1]$ são $\geq$ do que $A[i]$.

Invariante do laço externo:
Antes de cada iteração, o subvetor $A[0..i-1]$ está ordenado e possui os i menores elementos de A.

2. $f(n) = O(g(n))$ se, e somente se $g(n) = \Omega(f(n))$;

($\Rightarrow$) Se $\boxed{f(n) = O(g(n))}$ então existem constantes positivas c e n_0 tais que

$$0 \leq f(n) \leq c \cdot g(n), \quad \forall n \geq n_0$$

div. $c>0$

$$\Downarrow$$

$$\boxed{g(n) \geq \frac{1}{c} \cdot f(n), \quad \forall n \geq n_0.}$$

$$\Updownarrow \text{ def.}$$

$$g(n) = \Omega(f(n)).$$

3. $\boxed{T(1) = 1,}$ $T(n) = 2T(n-1) + 1$, $n \geq 2$

$$\begin{aligned}
T(n) &= 2 \cdot T(n-1) + 1 \\
&= 2\left(2 \cdot T(n-2) + 1\right) + 1 \\
&= 2^2 \cdot T(n-2) + 2 + 1 \\
&= 2^2\left(2 \cdot T(n-3) + 1\right) + 2 + 1 \\
&= 2^3 \cdot T(n-3) + 2^2 + 2 + 1 \\
&= \ldots
\end{aligned}$$

$$= 2^{n-1} \cdot \underbrace{T(n-(n-1))}_{1} + 2^{n-2} + 2^{n-3} + \cdots + 2^2 + 2 + 1$$

$$= 2^{n-1} + 2^{n-2} + \cdots + 2 + 1 = \sum_{i=0}^{n-1} 2^i$$

$$= 2^n - 1.$$

$$\Rightarrow \boxed{T(n) = 2^n - 1} \quad \checkmark$$

Indução em n:

(h.i) $\boxed{T(n-1) = 2^{n-1} - 1}$

$\boxed{n=1}$: $\checkmark$

$\boxed{n \geq 2}$: $T(n) = 2 \cdot \underline{T(n-1)} + 1$

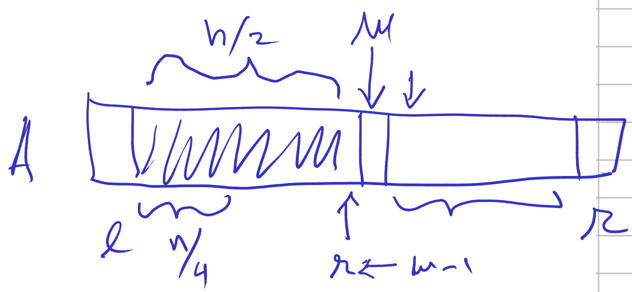
$$\overset{h.i}{=} 2 \cdot (2^{n-1} - 1) + 1$$

$$= \underline{\underline{2^n - 1}}$$

$\boxed{\text{fim}}$

```
while l ≤ r do
    m = ⌊(l + r)/2⌋;
    if key = A[m] then
        return m;
    end
    else
        if key < A[m] then
            r ← m − 1;
        end
        else
            l ← m + 1;
        end
    end
end
return l
```

Algorithm 3: BinarySearchPos($\underbrace{A[0..n-1]}$, l, r, key)

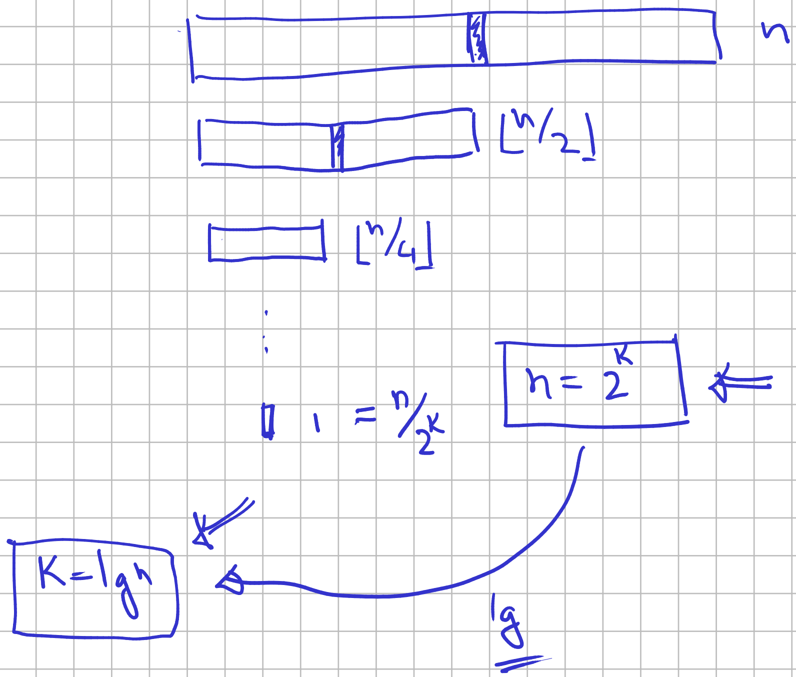


ordenado

$A[l..r]$

$T(n) = O(\lg n)$

$$T(n) = T(n/2) + \Theta(1)$$




```
for  j = 1 to n − 1 do
    pos ← BinarySearchPos(A[0..n − 1], 0, j − 1, A[j]);
    while  j > pos do
        swap A[j − 1] and A[j];
        j ← j − 1;
    end
end
```

lg

n/2

Algorithm 2: BInsertionSort($A[0..n-1]$)

$$T_w(n) = \sum_{i=1}^{n-1} O\left(\lg n + n/2\right) = \sum_{i=1}^{n-1} O(n/2) =$$

$$O\left(\frac{n}{2}\sum_{i=1}^{n-1} 1\right) = O\left(\frac{n}{2}\cdot(n-1)\right) = O(n^2). \ \blacksquare$$

$$\boxed{T_w(n) = O(n^2)}$$