```
1  if high < low then
2  |  return -1;
3  end
4  mid = ⌊(high + low)/2⌋;
5  if key > A[mid] then
6  |  return BinarySearch(A, mid + 1, high, key);
7  end
8  else
9  |  if key < A[mid] then
10 |  |  return BinarySearch(A, low, mid − 1, key);
11 |  end
12 |  else
13 |  |  return mid;
14 |  end
15 end
```

Algorithm 1: BinarySearch($A[1..n], low, high, key$)

(c) (3.0 pontos) A correção deste algoritmo pode ser estabelecida em duas etapas. A primeira dela consiste em provar que se a chave key não ocorre no vetor $A[1..n]$, então BinarySearch($A[1..n], 1, n, key$) retorna o valor -1. Prove o lema a seguir:

Seja $A[1..n]$ um vetor ordenado de inteiros distintos. Mostre que se a chave key não ocorre em $A[1..n]$, então BinarySearch($A[1..n], 1, n, key$) retorna o valor -1.

Dica: Indução (forte) sobre o tamanho n do vetor.

Prova: Indução em n, ou seja, no tamanho do vetor $A[1..n]$.

Base ($n=1$): Neste caso, low = high = 1 e portanto mid = 1 (linha 4). Como Key ≠ A[mid] já que estamos assumindo que Key não ocorre em A, temos 2 subcasos:

① Key > A[mid] (linha 5). Aqui temos a chamada recursiva BinarySearch(A[1], 2, 1, Key) que retorna −1 (linhas 1 e 2).

② Key < A[mid] (linha 9). Este caso é análogo ao caso anterior já que a chamada recursiva (linha 10) é dada por BinarySearch(A[1], 1, 0, Key) que também retorna −1 (linhas 1 e 2).

Passo indutivo ($n>1$): Considere a chamada inicial BinarySearch(A[1..n], 1, n, Key), e como $n>1$, a condição da linha 1 não é satisfeita. Logo mid = $\lfloor \frac{n+1}{2} \rfloor$ (linha 4), e temos 2 casos já que, por hipótese, Key não ocorre em A. Ou seja, Key ≠ A[mid].

① Key > A[mid] (linha 5): Neste caso, BinarySearch(A[1..n], mid+1, n, Key) (linha 6) retorna −1 por hipótese de indução.

② Key < A[mid] (linha 9): Análogo ao caso anterior, ou seja BinarySearch(A[1..n], 1, mid-1, Key) (linha 10) retorna -1 por hipótese de indução. ☒