Solution 1 (the sample solution)

```
function FindMin(A[.], c, lo, hi)
    if lo > hi then
        return -1
    mid = lo + (hi - lo) / 2
    if A[mid] == c then
        if mid > lo then
            if A[mid-1] != c then
                return mid
            else
                return FindMin(A[.], c, lo, mid-1)
        else
            return mid
    if A[mid] > c then
        return FindMin(A[.], c, lo, mid-1)
    else
        return FindMin(A[.], c, mid+1, hi)

function FindMax(A[.], c, lo, hi)
    if lo > hi then
        return -1
    mid = lo + (hi - lo) / 2
    if A[mid] == c then
        if mid < hi then
            if A[mid+1] != c then
                return mid
            else
                return FindMax(A[.], c, mid+1, hi)
        else
            return mid
    if A[mid] > c then
        return FindMax(A[.], c, lo, mid-1)
    else
        return FindMax(A[.], c, mid+1, hi)
```

```
function CountOccurrences(A[.], c)
    mini = FindMin(A[.], c, 0, len(A)-1)
    if mini == -1 then
        return 0
    maxi = FindMax(A[.], c, 0, len(A)-1)
    return maxi - mini + 1
```

Mark

5.0 / 5.0

Comment

This is the solution provided to students. A working algorithm with required time complexity.

Solution 2 (slightly modified from a student's submission)

```
function CountNum(A[.], c, lo, hi)
    if A[lo] > c or A[hi] < c
        return 0
    if A[lo] = c and A[hi] = c
        return hi - lo + 1
    mid := lo + (hi - lo) / 2
    return CountNum(A[.], c, lo, mid) + CountNum(A[.], c, mid, hi)

function Solution(A[.], c)
    // suppose there are n elements in A[]
    return CountNum(A[.], c, 0, n-1)
```

Mark

5.0 / 5.0

Comment

A different approach but indeed a working and elegant one. This algorithm does satisfy $O(logn)$ time complexity.

Solution 3

function Solution(A[0..n-1], c)

$c = 0$

for $i = 0$ **to** n-1 **do**

if $A[i] == c$ **then**

$c = c + 1$

return c

Mark

3.0 / 5.0

Comment

An algorithm that works for all possible inputs. It’s good to have less lines of code, but time complexity does matter in this subject.

Solution 4 (a student's submission)

function occur(A[.], c, n)
    i = first(A[.], 0, n-1, c, n)
    j = last(A[.], 0, n-1, c, n)
    return j - i + 1

function first(A[.], lo, hi, c, n)
    if hi >= lo **then** mid = (hi + lo) / 2
        if (mid == 0 or c > A[mid - 1]) **and** A[mid] = c **then**
            return mid
        if c > A[mid] **then**
            return first(A[.], mid + 1, hi, c, n)
        if c < A[mid] **then**
            return first(A[.], lo, mid - 1, c, n)
    return -1

function last(A[.], lo, hi, c, n)
    if hi >= lo **then** mid = (hi + lo) / 2
        if (mid == n-1 or c < A[mid + 1]) **and** A[mid] = c **then**
            return mid
        if c < A[mid] **then**
            return last(A[.], lo, mid - 1, c, n)
        if c > A[mid] **then**
            return last(A[.], mid + 1, hi, c, n)
    return -1

Mark

3.0 / 5.0

Comment

This solution suggests that the student does have clear clue of applying binary search. Problems / errors include (1) Incorrect indentation (2) Special case not correctly handled, the function returns 1 when there is no c in A (3) Case where $A[mid]=c$ AND $A[mid-1]=c$ is not handled.

Solution 5 (slightly modified from a student's submission)

Function BinSearch_Occurrence(*A*[.], *n*, *c*)

    $lo \leftarrow 0$

    $hi \leftarrow n - 1$

    $mid \leftarrow (lo + hi) / 2$

    $right \leftarrow 0$

    $left \leftarrow 0$

    while $A[mid] = c$ **do**

        $right \leftarrow mid + 1$

        $left \leftarrow mid - 1$

        if A[*right*] != *c* and *A*[*left*] != *c* **then**

            break

    return $right - left + 1$

Mark

2.0 / 5.0

Comment

This solution shows evidence of processing some elements, though it will return an incorrect value for most of the inputs.