The algorithm:
Assumption:
1> The unsorted array A contains finite number of integers, since array A contains n distinct numbers, and n is a certain number.
2> Assume there is a method for array called len() such that len(A) returns the length of the array A. Moreover, this method runs in O(1) in most languages, so we assume len() runs in O(1) here.
3> An integer which is smaller than the integers in the adjacent locations in an unsorted array of n $\geqslant$ 1 distinct integers called "wanted integer", the location of that "wanted integer" in the array called "wanted location". To be more specific, if array A contains only one integer, then the integer in A is automatically "wanted integer" and 1 is automatically "wanted location". Because, if A has only one integer, then there are no locations adjacent to the single location in the array, so the proposition: "the only number in A is smaller than the integers in the adjacent locations" is vacuously true.

Lemma: If A is a unsorted array of n $\geqslant$ 1 distinct integers, then we can find a "wanted integer" or "wanted location" in A.
Proof:
According to the property of set, finite set always has a maximum and a minimum.
So if all elements in A contains a set, this set has a minimum.
So, there exist a smallest integer in A.
In that all integers in A are distinct, other integers in the adjacent locations of that smallest integer (if exist) are greater than that smallest integer. Moreover ,when n=1, we consider that single number is smaller than the integer in the adjacent location according to the assumption.
So, we can always find a "wanted integer" or "wanted location" in A.

Question 1
```
def GetLocation(A, head ← 1, tail ← len(A)):    ### two assignments here
# In this function, we can call GetLocation(A), then head =1 and $\mathrm{tail} \leftarrow \mathrm{len}(A)$ by
#default. But we can also call GetLocation(A, h, t), then head = h and tail = t at that
#circumstance.
    n ← tail – head + 1          # here n is the length of the array we are going to loop
                                 # through no matter we call GetLocation(A) or
                                 # GetLocation(A, h, t)
                                 ###one assignment and one arithmetic calculation here
    if n=1:     # there is the first if statement.
                ### one comparison here
        return head
    i ← $\left\lfloor\frac{\mathrm{tail}+\mathrm{head}}{2}\right\rfloor$         # i is greater or equal to 1 at this time.

                            ### one assignment and one arithmetic calculation here
    if A[head]<A[head + 1]:   # there is the first if-else statement.
                              ###one assignment and one arithmetic calculation here
        return  head
    else if A[tail]<A[tail - 1]:    ###one assignment and one arithmetic calculation here
        return  tail
    else:     # n is greater than 2 at this time.
        if A[i]<A[i+1]:        # there is the second if-else statement.
                               ### one assignment and one arithmetic calculation here
                return GetLocation(A, head , i)   # this GetLocation(A, head , i) has
                                                  #three variables, A inherits from
                                                  #previous body A, head inherits
                                                  #from previous body head and i is
                                                  #equal to i
            else:     # A[i]>A[i+1]
                return GetLocation(A, i , tail)   # this GetLocation(A, i , tail) has
                                                  #three variables, A inherits from
                                                  #previous body A, i is equal to i
                                                  #and tail inherits from previous
                                                  #body tail.
```

Question 2

When computing the time complexity of GetLocation(A, head, tail), we just compute how many times the following operations: arithmetic calculations, comparisons and assignments proceeded , hence the time complexity of those and other operations depends on what programming language we are going to use. Moreover, I don't consider find A[i] or return something as a operation.

Let t(A, head, tail) denote the number of arithmetic calculations, comparisons and assignments during the execution of GetLocation(A, head, tail). According to the "###" comments in the algorithm,

$$\mathrm{t(A, head, tail)} = \begin{cases} 5 \;\; if\ head - tail + 1 = 1 \\ 8 \;\; if\ \mathrm{A[head]} < \mathrm{A[head+1]}: \\ 10 \;\; if\ \mathrm{A[tail]} < \mathrm{A[tail - 1]} \\ 11 + \mathrm{t}\left(\mathrm{A, head}, \left\lfloor \frac{\mathrm{tail + head}}{2} \right\rfloor\right) \;\; if\ \mathrm{A}\left[\left\lfloor \frac{\mathrm{tail+head}}{2} \right\rfloor\right] < A[\left\lfloor \frac{\mathrm{tail+head}}{2} \right\rfloor + 1] \\ 11 + \mathrm{t}\left(\mathrm{A}, \left\lfloor \frac{\mathrm{tail+head}}{2} \right\rfloor, \mathrm{tail}\right) \;\; if\ \mathrm{A}\left[\left\lfloor \frac{\mathrm{tail+head}}{2} \right\rfloor\right] > A[\left\lfloor \frac{\mathrm{tail+head}}{2} \right\rfloor + 1] \end{cases}$$

Let T(A, n) = max{t(A, head, tail) | n = tail – head + 1}

From $\mathrm{t(A, head, tail)}$, *we have*: $\mathrm{T(A,n)} = 5$ if $\mathrm{n} = 1$, $\mathrm{T(A,n)} \leq 10$ if $\mathrm{n} = 2$ and

$$\mathrm{T(A,n)} \leq 11 + \max\left(\mathrm{T}\left(\mathrm{A}, \left\lfloor \frac{\mathrm{n}}{2} \right\rfloor - 1\right), \mathrm{T}\left(\mathrm{A}, \left\lfloor \frac{\mathrm{n}}{2} \right\rfloor\right)\right) \text{ if such } \mathrm{T}\left(\mathrm{A}, \left\lfloor \frac{\mathrm{n}}{2} \right\rfloor - 1\right)\ and$$

$\mathrm{T}\left(\mathrm{A}, \left\lfloor \frac{\mathrm{n}}{2} \right\rfloor\right)$ meaningful.

Proposition: $\mathrm{b} = 2, \mathrm{c} = 11, \forall\ \mathrm{n} \in \mathbb{N}, (\mathrm{n} \geq \mathrm{b}\ \text{IMPLIES}\ \mathrm{T}(\mathrm{A}, \mathrm{n}) \leq \mathrm{c}(log_2 \mathrm{n}))$

Proof:

Let p(n)= $\mathrm{T}(\mathrm{A}, \mathrm{n}) \leq 11(log_2 \mathrm{n})$

Base case:

1.when n=2, $\mathrm{T}(\mathrm{A}, \mathrm{n}) \leq 11 \leq 11(log_2 2) = 11$

2.p(n) is True

Constructor cases:

3.let n be arbitrary and n≥ 2

4. $\left\lfloor \frac{\mathrm{n}}{2} \right\rfloor \leq \frac{\mathrm{n}}{2} < n$ AND $\left\lfloor \frac{\mathrm{n}}{2} \right\rfloor \leq n - 1$ (from 3 and the property of floor function)

5. $log_2 \left\lfloor \frac{\mathrm{n}}{2} \right\rfloor \leq log_2 \frac{\mathrm{n}}{2}$ (from 4 and the property of logarithm)

6.assume $\forall \mathrm{n} \in \mathbb{N}, (\mathrm{k} \leq \mathrm{n}\ \text{ IMPLIES }\ \mathrm{p}(\mathrm{k}))$

7. for k=n+1, $\mathrm{T}(\mathrm{A}, \mathrm{k}) \leq 11 + \max\left(\mathrm{T}\left(\mathrm{A}, \left\lfloor \frac{\mathrm{k}}{2} \right\rfloor - 1\right), \mathrm{T}\left(\mathrm{A}, \left\lfloor \frac{\mathrm{k}}{2} \right\rfloor\right)\right)$

(from the property of T(A, n))

8. $11 + \max\left(\mathrm{T}\left(\mathrm{A}, \left\lfloor \frac{\mathrm{k}}{2} \right\rfloor - 1\right), \mathrm{T}\left(\mathrm{A}, \left\lfloor \frac{\mathrm{k}}{2} \right\rfloor\right)\right) \leq 11 + \max\ (11\left(log_2 \left\lfloor \frac{\mathrm{k}}{2} \right\rfloor - 1\right), 11\left(log_2 \left\lfloor \frac{\mathrm{k}}{2} \right\rfloor\right))$ (direct prove from 4, 5, 6)

9. $\max\left(11\left(log_2 \left\lfloor \frac{\mathrm{k}}{2} \right\rfloor - 1\right), 11\left(log_2 \left\lfloor \frac{\mathrm{k}}{2} \right\rfloor\right)\right) \leq \max\ (11\left(log_2 \left(\frac{\mathrm{k}}{2}\right) - 1\right), 11\left(log_2 (\frac{\mathrm{k}}{2})\right))$ (direct prove from 5)

10. $\max\left(11\left(log_2 \left(\frac{\mathrm{k}}{2}\right) - 1\right), 11\left(log_2 (\frac{\mathrm{k}}{2})\right)\right) = 11\left(log_2 (\frac{\mathrm{k}}{2})\right) = 11(log_2 k - 1)$

(by the property of logarithm)

11. $\mathrm{T}(\mathrm{A},\mathrm{k}) \leq 11 + 11(log_2 k - 1) = 11 log_2 k$ (direct prove from 7,8,9,10)

So by induction, $\mathrm{b} = 1, \mathrm{c} = 11, \forall\ \mathrm{n} \in \mathbb{N}, (\mathrm{n} \geq \mathrm{b}\ \mathrm{IMPLIES}\ \mathrm{T}(\mathrm{A},\mathrm{n}) \leq \mathrm{c}(log_2 \mathrm{n}))$

So $\mathrm{T}(\mathrm{A},\mathrm{n}) \in \mathrm{O}(\log n)$ (by the definition of big O and 1, 2, 3, 6, 11)

$\mathrm{t}(\mathrm{A},\mathrm{head},\mathrm{tail}) \leq \mathrm{T}(\mathrm{A},\mathrm{n}), so\ \mathrm{t}(\mathrm{A},\mathrm{head},\mathrm{tail}) \in \mathrm{O}(\log n)$ (by the property of big O)

So, my algorithm runs in $\mathrm{O}(\log n)$ time.

Question 3

As a precondition, A is an unsorted array of finite number of distinct integers, and the length of A is greater or equal to one, which means A is not empty, so A contains a smallest integer, which is smaller than the integers in the adjacent locations. So, the algorithm we want must return if the precondition is satisfied.

My function GetLocation(A) has three cases:

1) When the unsorted array has length one, we consider that single number is smaller than the integer in the adjacent location. So, we directly return head, which is what the first if statement doing.
2) When the unsorted array has length two, then either A[head] or A[tail] is the smallest number. So we just return the smallest integer, which exactly smaller than the integers in the adjacent locations, which is what the first if-else statement doing.
3) When the unsorted array has length more than two, the algorithm checks whether the first and the last number in A smaller than the integers in the adjacent locations. If so, program returns and halts. If not, it goes to the second if-else statement, which is a recursive part. A[head .. tail] is an unsorted array of finite number of distinct integers, and the length of A is greater or equal to one, which means A is not empty, so A must contains a smallest integer. So, there must exist at least one integer in A[head .. tail] which is smaller than the integers in the adjacent locations. So we divide A[head .. tail] into two part, A, A[head .. i] and A[i .. tail]. Both A[head .. i] and A[i .. tail] are unsorted arrays of finite number of distinct integers, and the length of them are greater or equal to one, which means they are not empty, so Both A[head .. i] and A[i .. tail] must contain a "wanted integer" according to lemma. If other element in A[head .. i] is a "wanted integer" other than A[i], then it is automatically a "wanted integer" in A[head .. tail]. Similarly, if other element in A[i .. tail] is a "wanted integer" other than A[i+1], then it is automatically a "wanted integer" in A[head .. tail]. So There exist two possibilities here.

   Case 1: A[i]<A[i+1]. When A[i] is a "wanted integer" in A[head .. i], then A[i] is a "wanted integer" in A[head .. tail], because A[i]<A[i+1].

   Case 2: A[i]>A[i+1]. When A[i+1] is a "wanted integer" in A[i.. tail], then A[i+1] is a "wanted integer" in A[head .. tail], because A[i]>A[i+1].

So, the second if-else statement always can return a “wanted location” if the the sub GetLocation return a “wanted location”.

So, the recursive part work correctly if previous recursive part working correctly.

So, GetLocation(A) always returns the “wanted location”, if A satisfied the recondition.