```python
def unimodal_min(array):
##compare the first and last elements since a characteristic of a unimodal array is ascending and then decending values
##if the first element is less than or equal to the last element, return the first element as the minimum
   if array[0] <= array[-1]:
       return array[0]
###otherwise, return the last element as the minimum since it is the smaller of the two values
   else:
       return array[-1]
###################### end of code##########################
###tests####
a = [5, 9, 20, 70, 6, 5, 4, 2, -1]
minimum = unimodal_min(a)
print(minimum)
b = [-4, 9, 25, 14, 6, -4]
minimum = unimodal_min(b)
print(minimum)
c = [0, 3, 5, 2, 0]
minimum = unimodal_min(c)
print(minimum)
d = [0, 0]
minimum = unimodal_min(d)
print(minimum)
e = [2, 12000, -1]
minimum = unimodal_min(e)
print(minimum)
'''
line 2: if statement: O(3) operations: O(1) logical operation (<=) and O(2) array access
line 3: O(2): 1 operation for array access and one operation for return statement
line 5: O(2): 1 operation for array access and one operation for return statement
Time complexity/Big O notation is a constant, O(7) or O(c) where c can be any constant number. Constant running time.
You're performing one operation (comparing the first and last elements of an array) of length n.
This is independent of the length of the list, so no matter how long this list is,
it will take that same amount of time to perform.
'''
```