Exercise Session n. 5 (21 March 2023)

Algorithms and Data Structures

Tests are available here: **Tests and Solutions**

1. Exercise 243 (m20)

A left-rotation of an array `A` is defined as a permutation of `A` such that every element is shifted by one position to the left except for the first element that is moved to the last position.

Example:
A = [1, 2, 3, 4, 5, 6, 7, 8, 9] A left-rotation would change A into
A = [2, 3, 4, 5, 6, 7, 8, 9, 1]

Question 1: Write an algorithm `rotate(A,k)` that takes an array `A` and performs `k` left-rotations on `A`. The complexity of your algorithm must be `O(n)`, which means that the complexity must not depend on `k`

Examples

```
>>> rotate( [1, 2, 3, 4, 5], 1 )
[2, 3, 4, 5, 1]
>>> rotate( [1, 2, 3, 4, 5], 2 )
[3, 4, 5, 1, 2]
>>> rotate( [1, 2, 3, 4, 5], 3 )
[4, 5, 1, 2, 3]
```

Question 2: Write a function `rotate_in_place(A,k)` that takes an array `A` and, in `O(n)` steps, performs `k` left-rotations in-place. In-place means that `rotate_in_place(A,k)` may not use more than a constant amount of extra memory. If your implementation of `rotate(A,k)` is already in-place, then you may use it directly to implement `rotate_in_place(A,k)`.

2. Exercise 244 (m20)

Write a function `is_sorted(A)` that returns `True` if `A` is sorted in either ascending or descending order. Analyze the complexity of `is_sorted(A)`.

Examples

```
>>> is_sorted( [2, 1, 3, 4, 5] )
False
>>> is_sorted( [1, 2, 3, 4, 5] )
True
>>> is_sorted( [5, 4, 3, 2, 1] )
True
```

3. Exercise 255 (m21)

Given a sequence of `2n` numbers `A = x1, y1, x2, y2, . . . . , xn, yn` representing the Cartesian coordinates of `n` points in the plane, `p1 = (x1,y1), p2 = (x2, y2), . . . pn = (xn, yn)`, consider the line segments `pi–pj` defined by pairs of distinct points in `A`. You may assume that no two points in `A` are identical.

Question 1: Write two Python functions, `count_vertical(A)` and `count_horizontal(A)`, that given the sequence `A` structured as above, return the number of vertical and horizontal segments in `A`, respectively. Also, write an analysis of the complexity of your solution.

Question 2: Write a Python function `intersection(A)` that returns `True`if `A` contains at least one vertical segment that intersects at least one horizontal segment, or `False` otherwise. Also, write an analysis of the complexity of your solution, in particular describing a worst-case input.

Examples

```
>>> count_vertical( [1,2,1,3] )
1
>>> count_horizontal( [1,1,3,1] )
1
>>> intersect( [1,1,3,1,2,0,2,4] )
True
>>> intersect( [1,2,1,3,2,1,2,2] )
False
```