DSC 190 - Homework 03

Due: Thursday, February 4

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope on Thursday at 11:59 p.m.

Problem 1.

Suppose you initialize an LSH data for storing points in $\mathbb{R}^2$ by choosing $\ell = 1$, $w = 1$, and $k = 2$ random unit vectors (shown below):

$$\vec{u}^{(1)} = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)^T$$
$$\vec{u}^{(2)} = \left(\frac{\sqrt{2}}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right)^T$$

You insert the following points into the LSH data structure one-by-one:

x	y
-1.5	1.5
2.5	-1.5
-0.5	0.5
1.5	-0.5
-1.5	-1.5
0.5	0.5

Upon querying the point (-2, -2), what other point(s) will be in the same cell? Do your calculations by hand and show your work (though you can use code to check your answer, if you wish).

Programming Problem 1.

In a file named `knn_query.py`, implement a function named `knn_query(node, p, k)` which accepts three arguments:

- `node`: A `KDInternalNode` object representing the root of a k-d tree. (See lecture for the implementation of `KDInternalNode`.)
- `p`: A numpy array representing a query point.
- `k`: An integer representing the number of nearest neighbors to find.

Your function should return two things:

- A numpy array of distances to the kth nearest neighbors, in sorted order from smallest to largest.
- A $k \times d$ numpy array of the k nearest neighbors in order of distance to the query point. Each row of this array should represent a point. In the case of a tie (two points at the same distance), break the tie arbitrarily.

In the special case that there are fewer than k points in the tree, simply return all of the points.

Internally, use a heap to store the k closest points. Your heap should never get larger than $k + 1$ elements.

Example:

```
>>> data = np.array([
        [1, 2, 3],
        [4, 2, 1],
        [1, 1, 1],
        [7, 5, 5],
        [3, 2, 0]
    ])
>>> p = np.array([1, 1, 0])
>>> root = build_kd_tree(data) # implemented in lecture
>>> knn_query(root, p, k=3)
(array([1.        , 2.23606798, 3.16227766]),
 array([[1, 1, 1],
        [3, 2, 0],
        [1, 2, 3]]))
```