DSC 190 - Discussion 03

Problem 1.

When performing a search for the k nearest neighbors to a query point, we need to keep track of the k smallest distances found so far. We can do so using a heap.

Fill in the class below so that it keeps track of the k smallest numbers inserted while maintaining a heap whose size is never larger than $k + 1$.

```
class KSmallest:

    def __init__(self, k):
        ...

    def insert(self, number):
        """Insert a number."""
        ...

    def max(self):
        """Return the largest of the k numbers stored."""

    def as_list(self):
        """Return the k elements as a list."""
        ...
```

Solution:

```
class KSmallest:

    def __init__(self, k):
        self.k = k
        self.heap = MaxHeap()

    def insert(self, key):
        if len(self.heap.keys) < self.k or key < self.heap.max():
            self.heap.insert(key)

        if len(self.heap.keys) > self.k:
            self.heap.pop_max()

    def as_list(self):
        return list(self.heap.keys)

    def max(self):
        return self.heap.max()
```

Problem 2.

kNN search requires that we find the k nearest neighbours when we reach a leaf node in our search.

Fill in the brute force search function below to find k nearest neighbours to a point for a given leaf node.

```
def brute_force_knn_search(data, p, k):
    """
    Find nearest neighbour
    Parameters:
    data : np.ndarray
        An n X d array of points
    p : np.ndarray
        A d-array representing the query point
    k : int
        The number of neighbours to find

    Returns:
    knn : np.ndarray
        The k X d array of form [distance, point]
        where point is a d-array and distance is a float value
        represent distance to query point p
    """
```

Solution:

```
import numpy as np

def brute_force_nn_search(data, p, k):
    """Perform a brute force NN search.

    Parameters
    ----------
    data : ndarray
        An n x d array of n points in d dimensions.
    p : ndarray
        A d-array representing a query point.
    k : int
        The number of nearest neighbors to return. Default: 1

    Returns
    -------
    ndarray
        The k nearest neighbors of p as a k-by-d array.
    ndarray
        The distance to the k nearest neighbor.

    """
    all_distances = np.sqrt(np.sum((data - p)**2, axis=1))

    if k > len(data):
        k = len(data)

    # which distances are <= k?
    closest_ix = np.argpartition(all_distances, k-1)[:k]

    # extract the k closest points and their distances
    k_points = data[closest_ix]
    k_distances = all_distances[closest_ix]
    # lastly, we need to sort each
    sort_ix = np.argsort(k_distances)
    k_distances = k_distances[sort_ix]
    k_points = k_points[sort_ix]

    return (k_points, k_distances)
```