## DSC 40B - Homework 04
Due: Tuesday, February 3

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 at 11:59 p.m.

> This homework is a little shorter than usual, due to the midterm. Because of that, it will be worth fewer points than the other homeworks. **Keep in mind that slip days cannot be used on this homework.**

**Problem 1.**

For each of the sequences of numbers shown below, insert the numbers into a binary search tree (BST) in the exact order given. Draw the binary tree that results (each node in the tree should be labeled by the number it contains).

**a)** $50, 30, 70, 20, 40, 60, 80, 10, 35, 45, 55, 65, 75, 85, 5$

**b)** $5, 10, 20, 30, 35, 40, 50, 55, 45, 60, 70, 80, 85, 75, 65$

**Problem 2.**

Suppose a binary search tree has been augmented so that each node contains an additional attribute called `size` which contains the number of nodes in the subtree rooted at that node. Complete the following code so that it computes the value of the $k$th smallest key in the subtree rooted at `node`, where $k = 1$ is the minimum.

```python
def order_statistic(node, k):
    if node.left is None:
        left_size = 0
    else:
        left_size = node.left.size

    order = left_size + 1

    if order == k:
        return node.key
    elif order < k:
        return order_statistic(...)
    else:
        return order_statistic(...)
```

**Programming Problem 1.**

Suppose you are trying to remove outliers from a data set consisting of points in $\mathbb{R}^d$. One of the simplest approaches is to remove points that are in "sparse" regions – that is, points that don't have many other points close by. To do this, we might calculate the distance from a point to it's $k$th closest neighbor. If this distance is above some threshold, we consider the point an outlier.

More generally, the task of finding the distance from a query point to its $k$th closest "neighbor" is a common one in data science and machine learning. Here, we'll consider the 1-dimensional version of the

problem of finding $k$th neighbor distance. In a file named `knn_distance.py`, write a function named `knn_distance(arr, q, k)` that returns a pair of two things:

- the distance between `q` and the $k$th closest point to `q` in `arr`;

- the $k$th closest point to `q` in `arr` itself

The query point `q` does not need to be in `arr`. For simplicity, `arr` will be a Python list of numbers, and `q` will be a number. `k` should start counting at one, so that `knn_distance(arr, q, 1)` returns the distance between `q` and the point in `arr` closest to `q`. Your approach should have an expected time of $\Theta(n)$, where $n$ is the size of the input list. Your function may modify `arr`. In cases of a tie, the point you return is arbitrary (though the distance is not). Your code can assume that $k$ will be $\leq$ `len(arr)`.

Example:

```
>>> knn_distance([3, 10, 52, 15], 19, 1)
(4, 15)
>>> knn_distance([3, 10, 52, 15,], 19, 2)
(9, 10)
>>> knn_distance([3, 10, 52, 15], 19, 3)
(16, 3)
```

As this is a programming problem, submit your code to the Gradescope autograder.