```
                [6, 3, 7, 4, 6, 7],
                [7, 6, 7, 4, 9, 9]])
```

```
In[11]: # sort each row of X
        np.sort(X, axis=1)
```

```
Out[11]: array([[3, 4, 6, 6, 7, 9],
                [2, 3, 4, 6, 7, 7],
                [1, 2, 4, 5, 7, 7],
                [0, 1, 4, 5, 5, 9]])
```

Keep in mind that this treats each row or column as an independent array, and any relationships between the row or column values will be lost!

Partial Sorts: Partitioning

Sometimes we're not interested in sorting the entire array, but simply want to find the *K* smallest values in the array. NumPy provides this in the `np.partition` function. `np.partition` takes an array and a number *K*; the result is a new array with the smallest *K* values to the left of the partition, and the remaining values to the right, in arbitrary order:

```
In[12]: x = np.array([7, 2, 3, 1, 6, 5, 4])
        np.partition(x, 3)
```

```
Out[12]: array([2, 1, 3, 4, 6, 5, 7])
```

Note that the first three values in the resulting array are the three smallest in the array, and the remaining array positions contain the remaining values. Within the two partitions, the elements have arbitrary order.

Similarly to sorting, we can partition along an arbitrary axis of a multidimensional array:

```
In[13]: np.partition(X, 2, axis=1)
```

```
Out[13]: array([[3, 4, 6, 7, 6, 9],
                [2, 3, 4, 7, 6, 7],
                [1, 2, 4, 5, 7, 7],
                [0, 1, 4, 5, 9, 5]])
```

The result is an array where the first two slots in each row contain the smallest values from that row, with the remaining values filling the remaining slots.

Finally, just as there is a `np.argsort` that computes indices of the sort, there is a `np.argpartition` that computes indices of the partition. We'll see this in action in the following section.

Example: k-Nearest Neighbors

Let's quickly see how we might use this `argsort` function along multiple axes to find the nearest neighbors of each point in a set. We'll start by creating a random set of 10