```
def divide(dist,k,X,Y):
    ans_p=[np.sort(dist(p[0]-X[i],p[1]-Y[i]))for i in range(len(X))]
    ans_n=[np.sort(dist(n[0]-X[i],n[1]-Y[i]))for i in range(len(X))]
    t=[ans_p[i][int((k-1)/2)]>ans_n[i][int((k-1)/2)]for i in
range(len(ans_p))]
    return np.array(t)
```

这里对参数做几个假设

`dist`为一函数，接受两个参数为大小相同的数组(np.array)，同一位置的两个数组成一个向量坐标。返回值为存有所有向量的长度的数组(np.array)。

`k`为KNN的参数，并且默认是奇数。

`X`存有若干向量的横坐标，类型为np.array

`Y`存有若干向量的纵坐标，类型为np.array，大小与`X`相同。

即`(X[0],Y[0])`就是一个向量坐标。`dist`接受的两个参数实际上就是类似于`X`,`Y`这种。

2个trick

1. 可以注意到`p[0]`和`X[i]`的大小不一致，他们做减法用到了numpy中广播(boradcast)的trick。`p[0]-X[i]`得到的结果是`p[0]`中每一个元素都减去`X[i]`。其余同理。
2. 列表推导式。其格式为`[f(i) for i in [1,2,3]]`这样会得到`[f(1),f(2),f(3)]`。所以`ans_p`和`ans_n`都是二维数组。`t`是一个一维数组，他的每一个元素都是`bool`值。指示了是否属于负类。

关于如何求到如何属于哪一类。其核心语句为`ans_p[i][int((k-1)/2)]>ans_n[i][int((k-1)/2)]` 这就是看第i个点，是哪一类的。

注意在`ans_p[i]`和`ans_n[i]`中我们对距离进行了排序。那么想象一下，如果我们要找所有数据点中，距离`(X[i],Y[i])`最近的k个如何做。一个显然的做法就是在`ans_p[i]`和`ans_n[i]`中抽出前k小的，然后看一下从哪个数组中抽了更多元素。

那么问题就来了。如果其中一个数组中抽了更多元素，那么这个数组的第`int((k-1)/2)`个元素一定会被抽到。（注意这里是从0开始数的，如果是从1开始数就是`int((k+1)/2)`）那么如果是奇数，我们就知道在另外一个数组里面的第`int((k-1)/2)`个元素一定不会被抽到。

这就意味着，我们只需要比较第`int((k-1)/2)`个元素就能知道把`(X[i],Y[i])`分到哪一类了。

最后考虑一下k为偶数的情况。如果k为偶数，KNN算法就会出现某一些点无法分类的情况。通常的解决方案是随便（随机）放入某一类。在这里出现无法分类的情况就是两个数组中第`int((k-1)/2)`个元素都被抽到。排除这种情况，那仍然是一个被抽到，另外一个不被抽到。