Sreeram Danda
CS577 Homework #1
Problem #4

**#4 Find Median**
```
int getMedium(int[] A, int[] B, int n){
      int m1 = A[⌊(n / 2)⌋]
      int m2 = B[⌊(n / 2)⌋]

      if(n == 2){
            return min(max(A[0],B[0]), min(A[1],B[1]))
      }

      if(m1 > m2){
            return getMedium(A[0...⌊n-1/2⌋], B[⌊n/2⌋...n], n/2)
      }else if(m2 > m1){
            return getMedium(A[⌊n/2⌋...n], B[0...⌊n-1/2⌋], n/2)
      }
}
```

**Program Correctness:**
*Base Case: The size of both arrays (n) = 2*. The algorithm outputs the minimum of the max of the first value in each array and the minimum of the second value in both arrays. This is the correct output because in an array of size four. The median will be at the second index if the two arrays of size two are merged in sorted order. Taking the maximum of the first two elements guarantees the third element is correct and the minimum of the second values guarantees the second value is correct. The minimum of these values will then correctly return the median of the two arrays.
*Inductive Step: n > 2*. The algorithm executes one of two recursive calls. If the medium of array A is larger than the medium of array B, it means that the median of the two arrays lies between the first half of A and the second half of B. This is because all values of B less than the median are guaranteed to be smaller than the median of A, so they do not matter. Similarly, the values greater than the median of A are all greater than the median of B so the median must exist before them. Thus they do not matter. Then by IH, a recursive call with half of array A and half of array B is valid. Similarly, if the median of B is greater than median of A, one can imagine swapping the arrays and using the same method mentioned above to reduce the size of both arrays. Once again by IH, the recursive call is valid.

**#4 kth Smallest Element**
```
int kth(int[] A, int[] B, int k){
      if(A.length == 0)
            return B[k]
      else if (B.length == 0)
            return A[k]

      int m1 = A.length/2;
      int m2 = B.length/2;
```

```
        if(m1+m2 < k)
              if(A[m1] > B[m2])
                    return kth(A, B[m2+1,...,B.length], k - m2 - 1);
              else
                    return kth(A[m1+1,...,A.length], B, k - m1 - 1);
        else
              if(A[m1] > B[m2])
                    return kth(A[0,...,m1], B, k)
              else
                    return kth(A, B[0,...,m2], k)
  }
```

**Program Correctness:**
*Base Case: The size of either array A or B is 0*. The algorithm correctly returns the k^th^ value of the other array. If one array is empty the kth smallest element will be the kth element of the other array.
*Inductive Case:* If the sum of the middle index of each array is less than k, the algorithm determines which of the values at the indices is larger and shortens that array by half and recursively calls with a new k that is decremented. This is correct because if k is greater than the sum of the median indices one half of an array cannot have the kth element. At this point by comparing the two values at these indices we can determine which first half of the arrays can be eliminated. It turns out the smaller median is the array whose first half we can eliminate safely because those values are strictly smaller than both medians and k exists at or after both medians. Alternatively, if k is less than the sum of these indices then it exists in the first half of either array. If this is the case we can once again eliminate half of one of the arrays. Looking at the medians the median which is larger is the array we can reduce in size, k cannot exist past the larger median of the two arrays if the the sum of the indices of the medians are greater than k. Thus in both recursive calls we successfully reduce the size of one array by half. Then by IH, the recursive call is valid and returns the correct value.