Assignment 5

1.

You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values—so there are $2n$ values total—and you may assume that no two values are the same. You'd like to determine the median of this set of $2n$ values, which we will define here to be the n^{th} smallest value.

However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the kth smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most $O(logn)$ queries.

Suppose A and B are two databases, and $A(i)$, $B(i)$ are the i^{th} smallest element of A and B

Let k be the $\lceil \frac{n}{2} \rceil$, then $A(k)$ and $B(k)$ are the medians of the two databases. Suppose $A(k) < B(k)$, then one see that $B(k)$ is greater than the first k elements of A, also, $B(k)$ is always greater than the first $k-1$ elements of B. Therefore $B(k)$ is at least $2k^{th}$ element in the combine data base.

Since $2k \geq n$, all elements that are greater than $B(k)$ are greater than the median and we can eliminate the second part of the B database. Let B' be the half of B.

Similarly, the first $\lfloor \frac{n}{2} \rfloor$ elements of A are less than $B(k)$, and thus, are less than the last $n-k+1$ elements of B. Also they are less than the last [Inl elements of A. Sa, they are less than at least $n-h+1+\lceil \frac{n}{2} \rceil - n+1$ elements of the combine database. It means that they are less than the median and we can eliminate them as well. Let A' be the remaining parts of A.

Now we eliminate $\lfloor \frac{n}{2} \rfloor$ elements that are less than the median, and the same number of elements that are greater than median. It is clear that the median of the remainins clcmcnts is thc samc as thc mcdian of thc original sct of clcmcnts. Wc can find a mcdiar in the remaining set using recursion for A' and B'. Note that we can't delete elements from

the databases. However, we can access i^{th} smallest elements of A' and B': the i^{th} smallest elements of A' is $i + \lfloor \frac{n}{2} \rfloor^{th}$ smallest elements of A, and the i^{th} smallest elements of B' is i^{th} simallest elements of B.

Algorithm is the following. We write recursive function $median(n, a, b)$ that takes integers n, a and b and find the median of the union of the two segments $A[a+1; a+n]$ and $B[b+1; b+n]$.

```
FindMedian(a,b,n):
    if(n == 1){
        return min(A(a+k),b(b+k);
    }
    k = [n/2](ceil);
    if(A(a+k)<B(b+k)){
        return FindMedian(a+n/2(ceil),b,k);
    }else{
        return FindMedian(a,b+n/2(ceil),k);
    }
```

Suppose $T(n)$ be the number of queries asked by FindMedian to evaluate $median(a, b, n)$, then $T(n) = T(\lceil \frac{n}{2} \rceil) + 2$

Therefore, the time complexity is $T(n) = 2logn = O(log\ n)$.

2.

Recall the problem of finding the number of inversions. As in the text, we are given a sequence of n numbers $a_1, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that $a_i > a_j$. We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a significant inversion if $i < j$ and $a_i > 2a_j$. Give an $O(n\ logn)$ algorithm to count the number of significant inversions between two orderings.

Actually the solution is almost close to the `MergeSort` we have learnt before.

We will still solve it by using divided and conquer.

- When doing merge, we use original right array to merge. However, to calcuate signifiant inversion, we need to use another virtual array which each element in

the right array has been multiply by 2.

See the pseudocode below

```
// T(n) = O(n logn)
SignificantInversion(array[]):
    // base condition
    // O(1)
    if(array.length = 1){
        return 0, array[];
    }

    // Step1: Divide it into 2 part
    // O(n)
    left[] ← left-half of the array;
    right[] ← right-half of the array;

    // Step2: Conquer them separately
    // 2 * T(n/2)
    // The method will return the Significant Inversion it has counted
    Inversion1 = SignificantInversion(left[]);
    Inversion2 = SignificantInversion(right[]);


    // Step3: Merge to get the result
    // O(n)
    Inversion3, returnArray[] = Merge(left[], right[]);
    count = Inversion1 + Inversion2 + Inversion3;
    return count, returnArray[];

```

```
// O(n)
Merge(left[], right[]):
    create returnArr[left.length + right.length] // the Array that finish merged
    Inversion3 ← 0 // Siginificant Inversion
    right2 ← copy the right array, multuiply each element by 2;

    i ← 0;
    j ← 0;

```

```
    // compare left[] and right2[], to get the correct
Siginificant Inversion
    while(bose left[] and right2[] are not null){
        l ← left[i], r ← right2[j]
        if(right2[j] is the smaller value){
            Inversion3 + the elements left in left[];
            j++;
        }else{
            i++;
        }
    }

    // compare left[] and right[], to merge them into one array

    i ← 0;
    j ← 0;
    point ← 0; // index of returnArray[]

    while(bose left and right are not null){
        l ← left[i], r ← right[j]
        if(right[j] is the smaller value){
            returnArr[point++] = right[j++];
            j++;
        }else{
            returnArr[point++] = left[i++];
            i++;
        }
    }

    return Inversion3, returnArr[]

```

Suppose the time cost of `SignificantInversion()` is $T(n)$, then it satisfy

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

Thus, the total time complexity is $O(n\ logn)$