CS208 Theory Assignment 5

12312110 李轩然

DDL: May.6

Chapter 5 Exercise 1

Description

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 k-th 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.

Analysis

First, we ask for the smallest and the n-th smallest value of both databases, in order to set the low and high bounds.

Second, we have to choose n (not n-th) smallest values step by step. We ask for the median value ($k = \lceil \frac{n}{2} \rceil$) of both databases, define as m_1 and m_2. If $m_1 = m_2 = m$, then m is the n-th smallest value and we are done. Without lose of generality, we suppose $m_1 < m_2$, then we can say that the values in database 1 which is **not greater than m_1 MUST be choosen** (including m_1).

Now we have choosen $\lceil \frac{n}{2} \rceil$ values, and database 1 has $n - \lceil \frac{n}{2} \rceil$ values left. **Notice: Only $2n - \lceil \frac{n}{2} \rceil$ values left, so k must be changed to $k' = n - \frac{\lceil \frac{n}{2} \rceil}{2}$!** We keep choosing the k'-th smallest value of database 1 left as well as database 2, and then compare as the second step does.

Keep doing these steps until k becomes less than 3, and we are done. ($k = 1$, choose the smaller one we compare; $k = 2$, choose the larger one we compare)

We keep doing binary (quartery actually) search for a sequence whose total length is $2n$,

thus the complexity is $O(logn)$.

Chapter 5 Exercise 2

Description

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 $O(nlogn)$ algorithm to count the number of significant inversions between two orderings.

Analysis

Use **merge sort**, and count the number of significant inversion before merging. The complexities of counting and merging are the same, so the total time complexities is still $O(nlogn)$. Here is the new method of merge sort:

```
void merge(int* a, int left, int right, int mid) {
    int* b = new int [right - left + 1];
    int i = left, j = mid + 1, k = 0;

    // count, we do not change the array a
    int l = left, r = mid + 1;
    while (l <= mid and r <= right) {
        if (a[l] > 2 * a[r]) {
            cnt += mid - l; // a[l],a[l+1],...,a[mid] with a[r] are all valid
            r ++;
        } else l ++;
    }

    while (i <= mid and j <= right)
    {
        if (a[i] <= a[j]) {
            b[k] = a[i];
            k ++; i ++;
        }
        else {
            b[k] = a[j];
            k ++; j ++;
        }
```

```
    }

    while (i <= mid) {
        b[k] = a[i];
        k ++; i ++;
    }

    while (j <= right) {
        b[k] = a[j];
        k ++; j ++;
    }

    k = 0;
    for (int i = left; i <= right; i++)
    {
        a[i] = b[k];
        k ++;
    }

    delete[] b;
}

void mergesort(int* a, int left, int right) {
    if (left < right) {
        int mid = (right - left) / 2 + left;
        mergesort(a, left, mid);
        mergesort(a, mid + 1, right);
        merge(a, left, right, mid);
    }
}
```

Explanation: if $a[l] > 2 * a[r]$, then all of elements in the left-half array which are larger than $a[l]$ $(a[l+1], \ldots, a[mid])$ can also form a significant inversion with $a[r]$, so the number of inversions is $mid - l$. Count the total number of it and we are done.