CS208 · Algorithm Design

Assignment 5

刘乐奇 / 12011327

May 3, 2022

1 Chapter 5

1.1 Exercise 1

Suppose the two databases are A and B, and $A^{(i)}$ and $B^{(i)}$ are the i^{th} smallest elements of A and B, respectively.
At the first, we can try taking k as $\lceil \frac{n}{2} \rceil$. Then $A^{(k)}$ and $B^{(k)}$ are the medians of the two databases, respectively. Suppose that $A^{(k)} > B^{(k)}$ as below (or else we can exchange the role of A and B). the elements under the

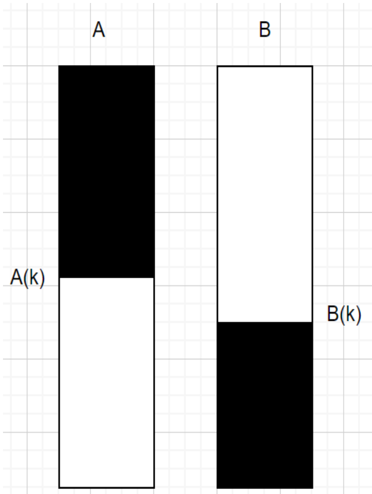

Figure 1: $A^{(k)} > B^{(k)}$

line of $A^{(k)}$ or $B^{(k)}$ are smaller than $A^{(k)}$ or $B^{(k)}$ in each database; above are greater.
Since $2k \geq n$, then all the elements greater than $A^{(k)}$ in A are also greater than the median of the two databases. So we can ignore the greater part of A, which has size of $n - k$. Similarly, all the elements smaller than $B^{(k)}$ in B are also smaller than the median of the two databases. So we can ignore the smaller part of B but still withhold $B^{(k)}$, which has size of $k - 1$.
After all, we only need to consider the median in k smaller elements in A and $n - k + 1$ greater elements in B, totally $n + 1$ elements, leading a recursive algorithm as below.

```
fn median(r: integer, a: integer, b: integer) -> value {
    if r==1 {                       // O(1)
        return min(A(a+k),B(b+k));
    }
    k: integer = (r+1)/2;           // O(1)
    if A(a+k)>B(b+k) {              // T((r+1)/2)
        return median(k, a, b+r/2);
    } else {
        return median(k, a+r/2, b);
    }
}
```

where a and b are the searching boundaries of A and B, respectively, r is the number of elements we want to search. Initially the parameters are $(n, 0, 0)$.

Obviously, we have the time complexity as

$$T(n) = T(\lceil \frac{n}{2} \rceil) + 2O(1);$$
$$\implies T(n) = 2\lceil \log n \rceil = O(\log n)$$

1.2 Exercise 2

We can count the significant inversions during the process of merge sort. Suppose that the origin sequence is $a_1, a_2, ..., a_n$ and after sorted it will be $a^{(1)}, a^{(2)}, ..., a^{(n)}$. The algorithm is as following

```
fn merge({a1,a2,...,an}: integer, n: integer) -> number: integer, {a(1),a(2),...,a(n)}:
     integer {
    if n==1 {                  // O(1)
        return (0,{a1});
    }
    k: integer = n/2;          // O(1)
    (n1, {a(1),a(2),...,a(k)}) = merge({a1,a2,...,ak},k); // O(nlogn)
    (n2, {a(k+1),a(k+2),...,a(n)}) = merge({a_{k+1},a_{k+2},...,an},n-k); // O(nlogn)
    i: integer = k, j: integer = n, N: integer = 0;
    while true {               // O(n)
        if a(i) <= 2a(j) {
            if j > (k+1) {
                j = j - 1;
            } else if j == (k+1) {
                break;
            }
        } else {
            N = N + j - k;
            if i > 1 {
                i = i - 1;
            } else if i == 1 {
                break;
            }
        }
    }
    {a(1),a(2),...,a(n)} = sort({a(1),a(2),...,a(k)},{a(k+1),a(k+2),...,a(n)}); // O(n)
    return (n1+n2+N, {a(1),a(2),...,a(n)});
}
```

The time complexity is

$$T(n) = 2O(1) + 2O(nlogn) + 2O(n)$$
$$\implies T(n) = O(nlogn)$$