Homework1

Zuyao Chen 201728008629002
zychen.uestc@gmail.com

1 Question1

a). **algorithm description:**

Let "query(X, k)" denotes the k^{th} smallest number of A or B and we call two databases $A = \{a_1, a_2, ..., a_i, ..., a_n\}, B = \{b_1, b_2, ..., b_j, ..., b_n\}$ arranged in ascending order (In fact,it does no matter to care the sequence order owing to "query").
query(A, i) can be written as a_i,as well as query(B, j). Making sure $i + j = n$,

- if $a_i < b_j$, the median lies in $\{a_{i+1}, ..., a_n\} \bigcup \{b_1, b_2, ..., b_j\}$;
- if $a_i = b_j$, the median is a_i or b_j;
- if $a_i > b_j$, the median lies in $\{a_1, a_2, ..., a_i\} \bigcup \{b_{j+1}, ..., b_n\}$

We initialize $i = n/2, j = n-i$,that equals to comparing the median of each database. Then the search area can be narrowed down to half length of the last until we just need to find 1^{th} smallest number between two separate arrays.
pseudo-code:

Algorithm 1 finding the median of two separate databases via query

Input: Two separate databases A,B,length n. (initializing $i, j = 0, k = n$)
Output: the median(the n^{th} smallest) of $A \bigcup B$

function FIND_KTH(A, i, B, j, k)
    if $k = 1$ **then**
        return $\min\{\text{query}(A, i+1), \text{query}(B, j+1)\}$
    end if
    if $i = 0$ (initial) **then**
        $i \leftarrow k/2, j \leftarrow k - i$
    end if
    if query(A, i) $<$ query(B, j) **then**
        $k \leftarrow k - k/2$ (each discards $k/2$ numbers)
        $i \leftarrow i + k/2, j \leftarrow j - k/2$
        if $k = 1$ **then**
            $j \leftarrow j - 1$
        end if
        return FIND_KTH(A, i, B, j, k)
    else if query(A, i) $>$ query(B, j) **then**
        $k \leftarrow k - k/2, i \leftarrow i - k/2, j \leftarrow j + k/2$
        if $k = 1$ **then**
            $i \leftarrow i - 1$
        end if
        return FIND_KTH(A, i, B, j, k)
    else
        return query(A, i)
    end if
end function

b). **subproblem reduction graph**

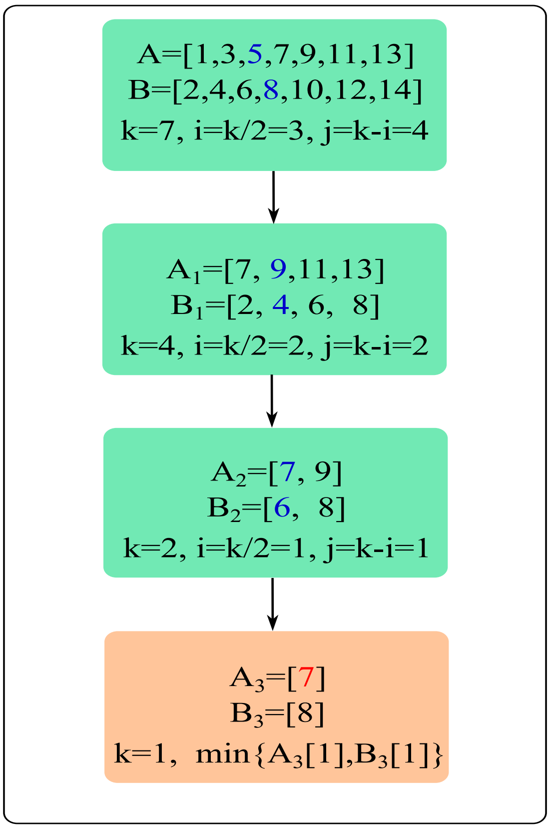

Figure 1: problem instance

c). **proof of the correctness**
Obviously, $k = 1$ means that we just want the 1^{th} smallest number of $A \bigcup B$. In order to find the median of $A \bigcup B$,we initialize $i = n/2, j = n - i, k = n$. Let "$A[i]$" denotes "query$(A, i)$", then compare $A[i]$ with $B[j]$:

i. if $A[i] < B[j]$, we can surely say that $\{A[1], A[2], ..., A[i]\}$ must lies in the left of the median and $\{B[j + 1], ..., B[n]\}$ must lies in the right of the median.For instance, if $B[j + 1]$ is the median,then there has $i + j = n$ numbers smaller than $B[j + 1]$ (each element of $\{A[1], ..., A[i]\} \cup \{B[1], B[2], ..., B[j]\}$ is smaller than $B[j + 1]$).

ii. if $A[i] = B[j]$, the median is $A[i]$ (or $B[j]$).

iii. if $A[i] > B[j]$,it's the opposite of i.

in each iteration, we discard $k/2$ numbers until $k = 1$.

d). **complexity of the algorithm**
The size of original problem is reduced to half at each iteration, and "query" costs $O(1)$,thus

$$T(n) = T(n/2) + cO(1) = O(\log n)$$

2 Question2

a). **algorithm description:**

- first,we randomly choose v from the array A;
- second,we split A into three categories : elements greater than v, those equal to v,and those smaller than v.Call these A_L, A_v, A_R respectively.
- then,we have

$$\text{select}(A,k) = \begin{cases} \text{select}(A_L,k) & \text{if} \quad k \le \text{len}(A_L) \\ v & \text{if} \quad \text{len}(A_L) < k \le \text{len}(A_L) + \text{len}(A_v) \\ \text{select}(A_R, k - \text{len}(A_L) - \text{len}(A_v)) & \text{if} \quad k > \text{len}(A_L) + \text{len}(A_v) \end{cases}$$

here "len" represents the length of an array.

pseudo-code:

Algorithm 2 find the k^{th} largest element in an unsorted array

Input: An unsorted array A and k
Output: the k^{th} largest number of A

```
function SELECT(A, k)
    if k ≤ 0 or k > len(A) then
        return error
    end if
    randomly choose v of A
    A_L = {}, A_v = {}, A_R = {}
    for i = 1 to len(A) do
        if A[i] > v then
            A_L = A_L ⋃ {A[i]}
        else if A[i] = v then
            A_v = A_v ⋃ {A[i]}
        else
            A_R = A_R ⋃ {A[i]}
        end if
    end for
    if k ≤ len(A_L) then
        return SELECT(A_L, k)
    else if k ≤ len(A_L) + len(A_v) then
        return v
    else
        return SELECT(A_R, k − len(A_L) − len(A_v))
    end if
end function
```

b). **subproblem reduction graph**

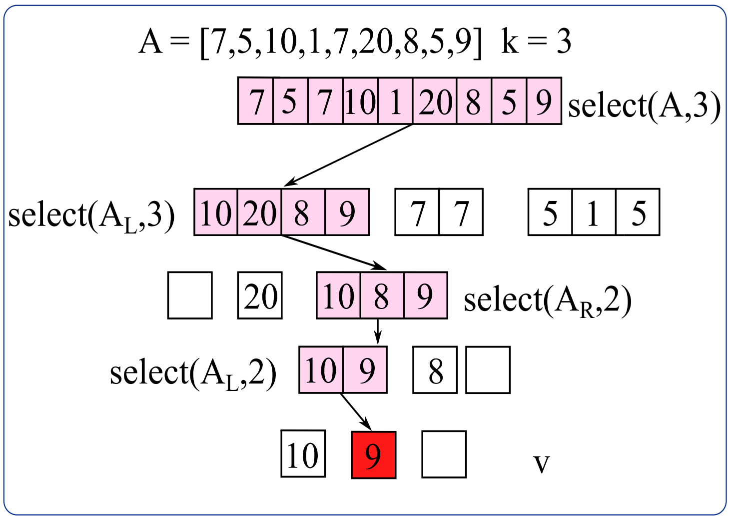

Figure 2: problem instance

c). **proof of the correctness**
In terms of the input constraints, it should throw an exception given $k \leq 0$ or $k > \text{len}(A)$. What we want is finding the k^{th} largest number of array A,there is no need to sort the array. In every recursion, the search can be narrowed down to one of three sublists until we choose the correct one of singletons.

d). **complexity of the algorithm**
Splitting A into three parts costs linear time.

- The most worst situation is that we choose the smallest number every times, then it would force our algorithm to perform

$$T(n) = T(n-1) + O(n)$$

  or $O(n^2)$ operations.

- The best-case scenario is that we select the median at each iteration, thus it would perform

$$T(n) = T(n/2) + O(n)$$

  or $O(n)$ operations.

- good choice: select a nearly-central element , $\text{len}(A_L) \geq \epsilon\text{len}(A)$,$\text{len}(A_R) \geq \epsilon\text{len}(A)$ for a fixed $0 < \epsilon < 1$,

$$\begin{aligned} T(n) \leq& T((1-\epsilon)\text{len}(A)) + O(n) \\ \leq& cn + c(1-\epsilon)n + c(1-\epsilon)^2 n + ... \\ =& O(n) \end{aligned}$$

3 Question3

a). **algorithm description:**

b). **subproblem reduction graph**

c). **proof of the correctness**

d). **complexity of the algorithm**