CS 260 Fundamentals of the Design and Analysis of Algorithms **Fall 2016**

HomeWork 4

Name: *Liangjian Chen*
ID*: #52006933*

November 1, 2016

Problem 5.1 **Solution:**

Assume that $A(i)$ means the i^{th} smallest number in Database A, and $B(i)$ means the i^{th} smallest number in Database B.

function FIND_MEDIAN(n, L_A, L_B)
  if $N == 1$ **then**
    return Min($A(L_A + 1), B(L_B + 1)$)
  end if
  $mid = \lceil \frac{n}{2} \rceil$
  if $A(L_A + mid) \leq B(L_B + mid)$ **then**
    FIND_MEDIAN($mid, L_A + \lfloor \frac{n}{2} \rfloor, L_B$)
  else
    FIND_MEDIAN($mid, L_A, L_B + \lfloor \frac{n}{2} \rfloor$)
  end if
end function

Initially, call function FIND_MEDIAN($n, 0, 0$)

Every time, we call two median values in both database($mid = \lceil \frac{n}{2} \rceil$), we can see if $A(mid) \leq B(mid)$, then $A(1)..A(\lfloor \frac{n}{2} \rfloor)$ would not be the answer, $B(\lceil \frac{n}{2} \rceil)..B(N)$ would not be the answer. So, every time we eliminate half of the possible answer which leads the recurrence $T(n) = T(n/2) + O(1)$, solve this recurrence,we get $T(n) = O(log(n))$

Problem 5.3 **Solution:**

There is a linear time algorithm.

function FIND_MAJORITY_CARD($Cards$)
  $Cur = \emptyset$
  $Count = 0$
  for all $card \in Crads$ **do**
    if $Count == 0$ **then**
      $Current = card$
    else
      if EQUIVALENCE_TESTER(Current, card) $= True$ **then**
        $Count = Count + 1$
      else
        $Count = Count - 1$
      end if
    end if
  end for

```
    Count = 0
    for all card ∈ Crads do
        if EQUIVALENCE_TESTER(Current, card) = True then
            Count = Count + 1
        end if
    end for
    if Count ≥ ⌈n/2⌉ then
        return Current is a majority card.
    else
        return no such card.
    end if
end function
```

The idea is that if there is a card that equivalent to more than $\lceil \frac{n}{2} \rceil$ other cards, then after first for loop, the current must be that card. So in second for loop we check whether it is the answer. We iterate all cards twice, so time complexity is $O(n)$

Problem 5.5 **Solution:**

Consider every line $y = a_i x + b_i$ is a half-plane $y \geq a_i x + b_i$. Finding upper bound is equivalent to finding their half-plane intersection. There are several different approaches that is able to achieve $O(n \log n)$ time. One of them is base on the Dived and Conquer(Actually it is a such classical algorithm that you can find it in every computational geometry text book).

The main idea is split all the half-planes into two equal size subset and recursively calculate these two half-plane intersections, then merge them together to one. The divide part is easy and the key point is how to merge two intersections.

First, it is easy to realize that all half-plane intersection is a convex polygon(Maybe unbounded) and also in this problem, all intersections would be a lower polygon shell. Another crucial point is that, from left to right, the slope of visible line should be monotonically increasing. This feature make sure we can do it by two points.

After that, Let's assume $Segment$ is the segment contains all segment in half-plane intersection. Pointer p_1 point to current segment in $Segment_1$. Pointer p_2 point to current Segment in $Segment_2$.

Then between two segment $Segment_1[p_1]$ and $Segment_2[p_2]$. First, we need check whether the left point of one segment is on the right of another segment's right point. If so that means one segment is entirely on the left of another one. In this case, we put the left segment into answer list and move the left segment's pointer to next and go back to beginning again.

Second determined that in the very left, which Segment is on the upper. Then calculate the intersection point of these two segment. If there is not intersection point of two segments, which mean the upper one cover the lower one, move the pointer of lower segment to next and go beginning. If a intersection point reported, which means the upper segment with intersection point as right point is a new segment, put the new segment into our answer list. Finally return answer list as half-plane intersection.

Obviously, we iterator all the segment once, so merge time is $O(n)$, overall time complexity is $T(n) = 2T(\frac{n}{2}) + O(n)$, which leads to $T(n) = O(n \log n)$

Extra Problem 1 **Solution:**

```
function FIND_MISSING_NUMBER(k)
    S = {1, 2, 3..2^k − 1}
    Ans = 0
    for i ← k − 1, k − 2, ..., 0 do
        S_0 = ∅ and S_1 = ∅
        for all s ∈ S do
            if BITVALUE(s, i + 1) == 0 then
                S_0 ← S_0 + {s}
            else
                S_1 ← S_1 + {s}
            end if
        end for
        if S_1's size equal to 2^i − 1 then
            S ← S_1
            Ans = Ans + 2^{k−i−1}
        else
            S ← S_0
        end if
    end for
    return ans
end function
```

In each iteration, count the how many number whose i^th bit is 1, and how many is 0. So the size of corresponding set to missing number's i^th bit is $2^i - 1$. Then we can construct the answer step by step.

After each iteration, the size of S becomes a half of previous one. So $T(n) = n + \frac{n}{2} + \frac{n}{4} ... + 1 = O(n)$

Extra Problem 2 **Solution:**

(a) $\log_4(6) > 1$. So, $T(n) = \Theta(n^{\log_4(6)})$

(b) $\log_4(2) = \frac{1}{2}$. So, $T(n) = \Theta(\sqrt{n} \log n)$

(c) $\log_3(6) < 2$. Also choose $k = \frac{3}{4}$, $6(\frac{n}{3})^2 = \frac{2n^2}{3} \leq kn^2$. Thus, $T(n) = \Theta(n^2)$

(d) Assume that $n = a^{2b}$, Thus

$$T(a^{2b}) = a^b T(a^b) + a^{2b}$$
$$\frac{T(a^{2b})}{a^{2b}} = \frac{T(a^b)}{a^b} + 1$$

Easily conclude that

$$\frac{T(a^{2b})}{a^{2b}} = \Theta(\log b)$$
$$T(n) = n * \Theta(\log \frac{\log_a(n)}{2})$$
$$T(n) = \Theta(n \log \log n)$$