CS 170 HW 02

Jialiang Tang
tangjialiang@berkeley.edu

9/10/2023

1. Study Group

   Jialiang Tang (myself), SID: 3039758308

   Buji Xu, SID: 3039687809

2. Two sorted arrays

Description of algorithm: Use 2 numbers to mark the two element to consider. Assume the arrays are a[k] and b[k]. Firstly make pos_a = 0, pos_b = 0. Then we do such operations k times: compare a[pos_a] with b[pos_b], add the counter by 1 (counter $\overset{=m}{}$ represents it's the m-th smallest element and initial value is 0). If a[pos_a] is smaller, add pos_a by 1, else add pos_b by 1. Whenever the counter == k, return immediately the smaller one of a[pos_a] and b[pos_b].

Proof of correctness: At first a[0] and b[0] are smallest in their array, so the smallest element is the smaller one of them. After we find all the first to the i-th smallest elements, the pos_a and pos_b are marking the smallest element of rest of a and rest of b since a and b are sorted and all elements found are in front of pos_a in a or pos_b in b. So the (i+1)-th smallest element is the smaller one of a[pos_a] and b[pos_b] By limited steps, we find the k-th smallest element.

Runtime analysis: we only need to make at most k comparisons and O(k) additions. So the runtime overall is O(k).

pseudo code

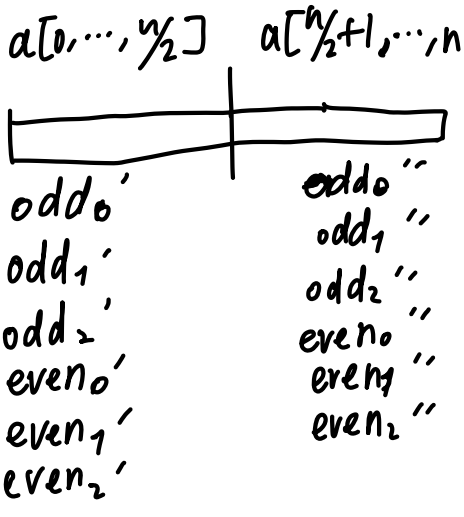
two_sorted_arrays (a[0..k], b[0..k-1])

Input : two sorted arrays a[] and b[].

```
pos_a = 0 , pos_b = 0 , counter = 0
while True:
    if a[pos_a] >= b[pos_b]:
        pos_b += 1
        counter += 1
        if counter == k:
            return b[pos_b - 1]
    else:
        pos_a += 1
        counter += 1
        if counter == k:
            return a[pos_a - 1]
```

3. Counting multiples of 3

Description of algorithm : Divide whole array by half, compute 6 datas for each sub-array : number of odd-sized subsets whose element add up to 0 (mod 3), numbers of odd-sized subsets whose element add up to 1 (mod 3), numbers of odd-sized subsets whose element add up to 2 (mod 3), numbers of even-sized subsets whose element add up to 0 (mod 3), numbers of even-sized subsets whose element add up to 1 (mod 3), numbers of even-sized subsets whose element add up to 2 (mod 3). Then compute the same 6 numbers of current array by multiply some of ones of 6 numbers of first sub-array and some of ones of 6 numbers of second sub-array and add them together.



numbers of whole array are below:

$odd_0 = odd_0' \times even_0'' + even_0' \times odd_0'' + odd_1' \times even_2'' + even_1' \times odd_2'' + odd_2' \times even_1'' + even_2' \times odd_1''$

Use addition principle and multiplication principle in combinatorics

Others are omitted. The principle is to add all the subsets satisfying parity and mod 3 condition, as well as addition principle and multiplication principle in combinatorics.

We can directly assign the 6 numbers of array of length 1, so by the divide-and-conquer, we have 6 numbers of initial array, then we return the odd of it.

Proof of correctness: We only need to prove that after division, the number of subsets are the multiplication and addition of numbers of subsets of sub-arrays. Each selection can be seen as 2 steps: we first select some elements in first sub-array, then select some elements in second array. It's a multi-step decision so the multiplication principle holds. On the other hand, there are different selections in each step, so the addition principle holds.

Also, odd+even=even+odd=odd, even+even=odd+odd=even,
$0+0=1+2=2+1=0 \pmod 3$, $0+1=1+0=1 \pmod 3$, $0+2=2+0=1+1=2 \pmod 3$
($1+0 = 2+2$)

As a result, we can properly computer the number of subsets, the algorithm is correct.

Runtime analysis: Assume the runtime for n-length array is $T(n)$, since n-bit can represent 2^n numbers and number of subsets of n elements is at most 2^n, we can say that it can be stored in an n-bit integer.

So the multiplication of k-bit integer takes $O(k \log k)$, while addition takes $O(k)$. Then we have

$$
\begin{aligned}
T(n) &= 2T(n/2) + O(n \log n) \\
&= 2(2T(n/4) + O(n/2 \log n/2)) + O(n \log n) \\
&= \cdots = 2^{\log_2 n} T(1) + O(n \log n + n \log \tfrac{n}{2} + \cdots + n \log 1) \\
&= n + O(n \log^2 n - (\log 2 + \log 4 + \cdots + \log n)) \\
&= O(n \log^2 n)
\end{aligned}
$$

4. The Resistance

Strategy: Divide players into s groups, each of which includes n/s players. For each player, we act as below:

① Let the whole group go on a mission, if succeed, then finish.

②Else, divide players into 2 groups, each of which has half of people, let the 2 groups go on a mission, there must be at least one of them fails. Then divide by half the failed group and let 2 groups go on a mission again. Repeat the strategy until there is a one-person group failing, then identify the person as a spy. Erase him from the whole $\frac{n}{s}$-people group, go back to ①.

Missions needed: For each n/s-people group, each time we are certain to find a spy after $O(\log \frac{n}{s})$

missions (if the group has spy) or directly say the group has no spy in $O(1)$ missions.

As a result, each spy needs $O(\log \frac{n}{s})$ missions to be identified, so we need $O(s \log \frac{n}{s})$ missions

5. (a)

Description of Algorithm: Query every one (except x) with x.

If at least $n/2$ people say x is human. then return True.
Otherwise return False.

Explanation: Assume there are h humans and w wolves, then $h+w=n$, $h>w$.

If x is human, then there are $h-1$ humans and w wolves in the rest. So at least $h-1 \geq n/2$ people will say x is human (human always tell the truth).

If x is werewolf, there are h humans and $w-1$ werewolves in the rest, So at most $w-1 < n/2$ people will say x is human.

(b)

Description of Algorithm:

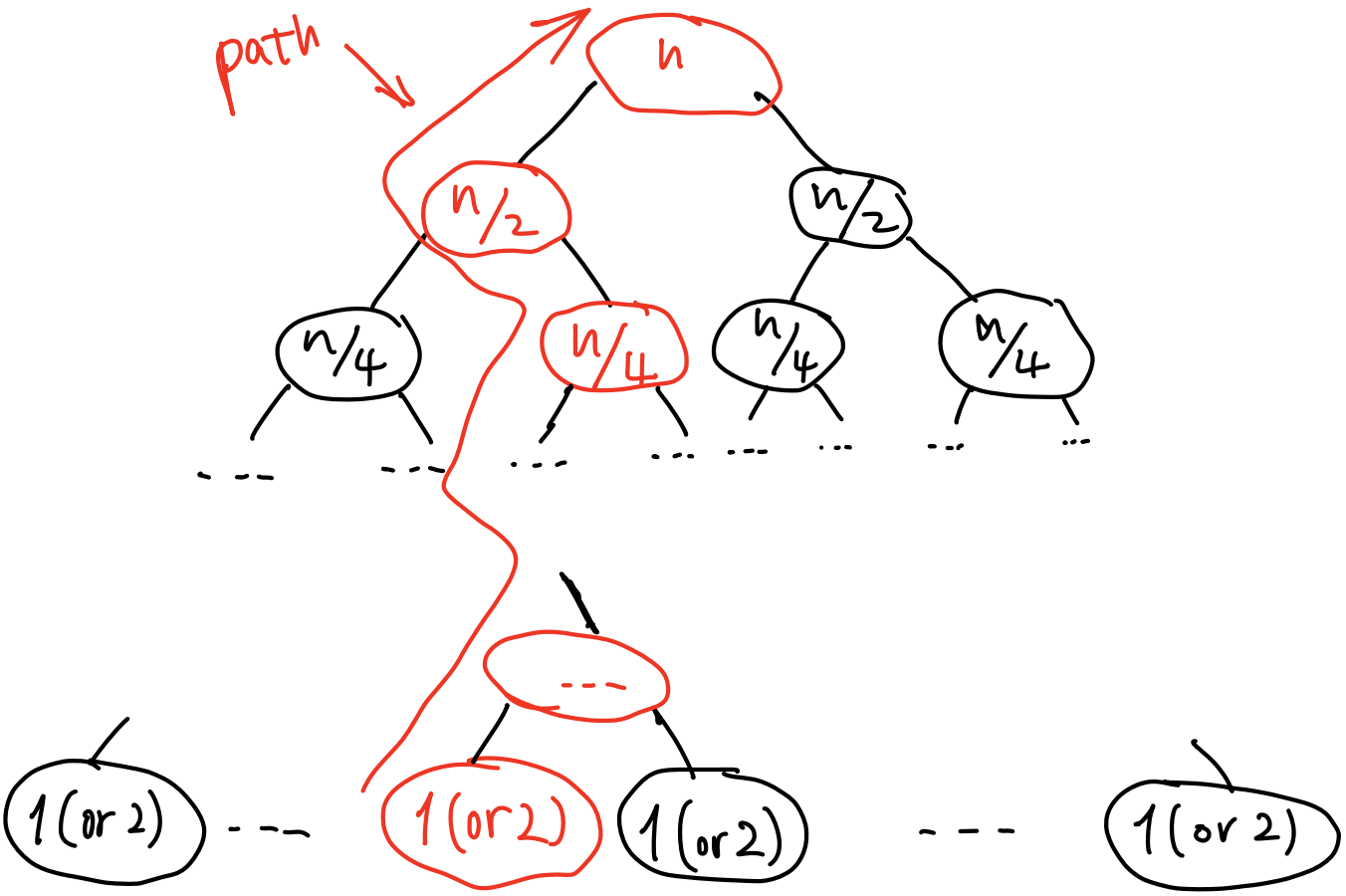
If $n=1$, then return the only people.

If $n=2$, then return either of them.

Divide all people by half into 2 groups,

then recursively run the algorithm on each group. For each friend returned by the algorithm, run algorithm in (a) at range of all people to identify whether he/she is human. Return the human one.

Proof of correctness: It's obvious that after a division, at least one of the 2 groups satisfy the condition: human is more than werewolves. So there exist a path in the recursion tree that all nodes (groups) in it satisfy the condition.



In the simplest case, if there are only 1 or 2 people and the group satisfy that human is more than werewolves,

all the people must be human. By our algorithm, it returns a human certainly. Since its father node also satisfy human is more than weremolves, we will identify correctly the two friends returned by 2 groups. So we return a human certainly in the case just 'above' the simplest case. By induction, the cases on the path will all return a human correctly. So the top case (n-scale) returns a human correctly.

Runtime analysis: We divide problem into 2 sub-problem whose scale is $n/2$ and recursively run algorithm on all of them. Also we take $2O(n) = O(n)$ queries to identify the human one of 2 friends returned by algorithm. Assume that it takes $T(n)$ queries to solve the n-scale problem, we have

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

So $T(n) = O(n \log n)$ by master theorem.