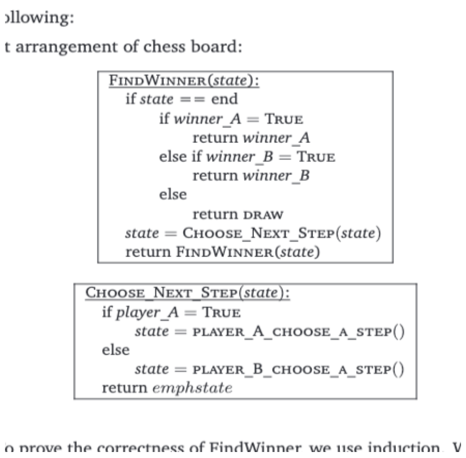
Problem 1:

At first, I was thinking about how each player would make decisions and moves to achieve a win. I want to apply max-min algorithm which is usually used in game theory to help solute the problem. However, I later noticed that each player is playing perfectly, so we do not need to consider players' decisions.

So I apply an algorithm that take a state from current chess board and it returns winner when the game is end. Also, each move made by the players would change the whole chess board. And I apply the algorithm recursively, take the changing state and continue exam whether the game is over.



ɔllowing:

t arrangement of chess board:

```
FINDWINNER(state):
  if state == end
      if winner_A = TRUE
          return winner_A
      else if winner_B = TRUE
          return winner_B
      else
          return DRAW
  state = CHOOSE_NEXT_STEP(state)
  return FINDWINNER(state)
```

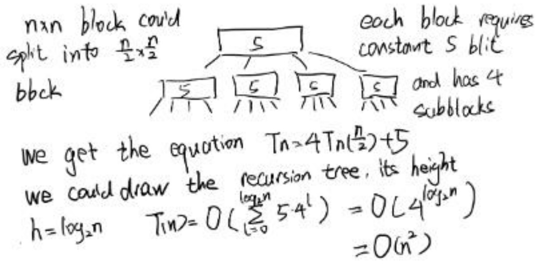
```
CHOOSE_NEXT_STEP(state):
  if player_A = TRUE
      state = PLAYER_A_CHOOSE_A_STEP()
  else
      state = PLAYER_B_CHOOSE_A_STEP()
  return emphstate
```

o prove the correctness of FindWinner, we use induction. V

Problem 2:

a) Since it is a problem that contains a big problem, which could be recursively split into small pieces of problems, I firstly think about we could use induction to prove the correctness of algorithms. For the base case, when there is only pixel, it only needs to be rotated. And I noticed that the n x n block (where n is power of 2), could be divided into 4 same n/2 x n/2 blocks. In that case, we only need to use induction, assuming that it is correct on n/2 x n/2 block, then prove it works on n x n block.
   And I thought for two both algorithms, whether first blit or recurse, they could have same basic case and induction hypothesis to prove it.
b) To get the total numbers of blit, we could easily draw a recursion tree, each node has four nodes, which means each block could split into four small blocks. And each block needs exact 5 blits, so by calculating all nodes and the blits in it, we can find it is exponential, the total blits count is 5^n. And I use the tree and obtain the equation for run time, Tn = 4T(n/2) + 5

It is easy to get the run time complexity which is about O(n^2)



n×n block could split into $\frac{n}{2}\times\frac{n}{2}$ block

each block requires constant 5 blit and has 4 subblocks

we get the equation $T_n = 4T_n(\frac{n}{2})+5$

we could draw the recursion tree, its height $h=\log_2 n$

$T(n) = O(\sum_{i=0}^{\log_2 n} 5\cdot 4^i) = O(4^{\log_2 n})$

$= O(n^2)$

c) At first, we want to apply a "L" block to solve the problem, but we found it is hard to blit an no place to place the block.
And I thought about an algorithm blit like a ring, each time we blit a block contains n-1 pixels (like a bar), clockwisely to another sides. And recursively doing the ring (or bar) blit from outside to inside.
And for each time reduing "one ring", it does 5 blits. And For total, for even number, does n/2*5 blits, for odd, does (n-1)/2*5 blits

Problem 3

To design an algorithm, we notice that this question has exactly two parts.
The first one is counting the depth of the completed tree, and the second job is to find where is the root located (return a pointer or object)
So we could design two algorithms. Since we are solving binary tree problems, both of two methods would be recursive. The first method is used to calculate the depth of the completed tree from the current node. By the definition of the completed tree, two sides of the trees must be full and balance, so we only need to consider which side has lower depth, and count the parent itself as additional depth by adding 1.

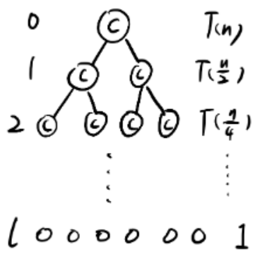
To find the root, we can also compare the depth of current root and with its children, if the children get larger depth, just keep children as root, else if update the current root as root.

To prove the correctness, since both of algorithms are recursive, so we can apply induction. The basic case for counting depth is when have no left or on right, and for finding root is the root is null. It is easy to have hypothesis that their children is correct and then prove whole algorithms is correct.

For run time complexity, we could apply recursion tree again. For each node, it has two sub cases, so we can get T(n) = 2T(n/2)+c

Complexity:

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + C$$
$$= 2T(\frac{n}{2}) + C$$
$$T(n) = O(n)$$


0
1
2
l
T(n)
T(n/2)
T(n/4)
1