香港中文大學（深圳）
The Chinese University of Hong Kong

CSC3100 Data Structures
Mid-term Exam

School of Data Science (SDS)
The Chinese University of Hong Kong, Shenzhen

Overview

- #1 Multiple choice questions (27)
- #2 Asymptotic notation (12)
- #3 linked list (delete nodes & find integers) (15)
- #4 STACK (2 stacks in one array) (15)
- #5 Tree reconstruction (16)
- #6 BST (insert, delete key & build complete BST) (15)

	Q1	Q2	Q3	Q4	Q5	Q6	Total
Average	22.47	10.24	11.25	8.31	11.90	9.57	73.73
Median	24	12	12	10	13	10	79
Max	27	12	15	15	16	15	100
Min	0	0	0	0	0	0	0

6 students

Multiple choice questions

1. [9×3 marks] Choose **ONE** solution that best suits each question. (3 marks each)

(1) The time complexity of the following program is ( )

```
for (i = 0; i < m; i ++)
  for (j = 0; j < n; j ++)
A[i][j] = i + j;
```

A. $O(m^2)$ B. $O(n^2)$ C. $O(mn)$ D. $O(m+n)$

(2) What is the time complexity to count the number of elements in a singly linked list that contains *n* elements? ( )

A. $O(1)$ B. $O(n)$ C. $O(\log n)$ D. $O(n^2)$

(3) Given a sequence of 5 integers *1 2 3 4 5* that are sequentially pushed into a stack, it is impossible to get a popped sequence of integers in ( )

A. *2 3 4 1 5* B. *5 4 1 3 2* C. *2 3 1 4 5* D. *1 5 4 3 2*

(4) A linear queue follows ( )

A. LIFO principle B. FIFO principle C. Linear tree D. Ordered array

Multiple choice questions

(5) Given a binary tree, how many children does each node have ( )

A. *2* B. Any number of children C. *0* or *1* or *2* D. *0* or *1*

(6) Given a binary tree with *n* nodes, its height is ( )

A. $log_2 n$ B. *n* C. *n/2* D. Uncertain

(7) What kind of traversal does the following piece of code use? ( )

```
public void func(Tree root){
    System.out.println(root.data());
    func(root.left());
    func(root.right());
}
```

A. Pre-order B. In-order C. Post-order D. None of the above

(8) Given an array of *n* integers in the range [0, 100] which are sorted in ascending order, how many comparisons does CountingSort need to sort it in descending order? ( )

A. *0* B. $\Theta(n)$ C. $\Theta(nlogn)$ D. $\Theta(n^2)$

(9) What are the best-case and worst-case time complexities of MergeSort? ( )

A. $O(n)$ and $O(nlogn)$ B. $O(n)$ and $O(n^2)$

C. $O(nlogn)$ and $O(nlogn)$ D. $O(nlogn)$ and $O(n^2)$

#2 Asymptotic notation

2. [12 marks] State and prove whether the following two statements are correct or not:
(1) [6 marks] $3n^3 2^n + 2000n^2 3^n = O(n^3 3^n)$;
(2) [6 marks] $4n^3 + 5n^2 = \Omega(n^3)$;

Asymptotic notation

$3n^3 2^n + 200n^2 3^n = O(n^3 3^n)$

Proof:

We need to find c, n_0 satisfying that

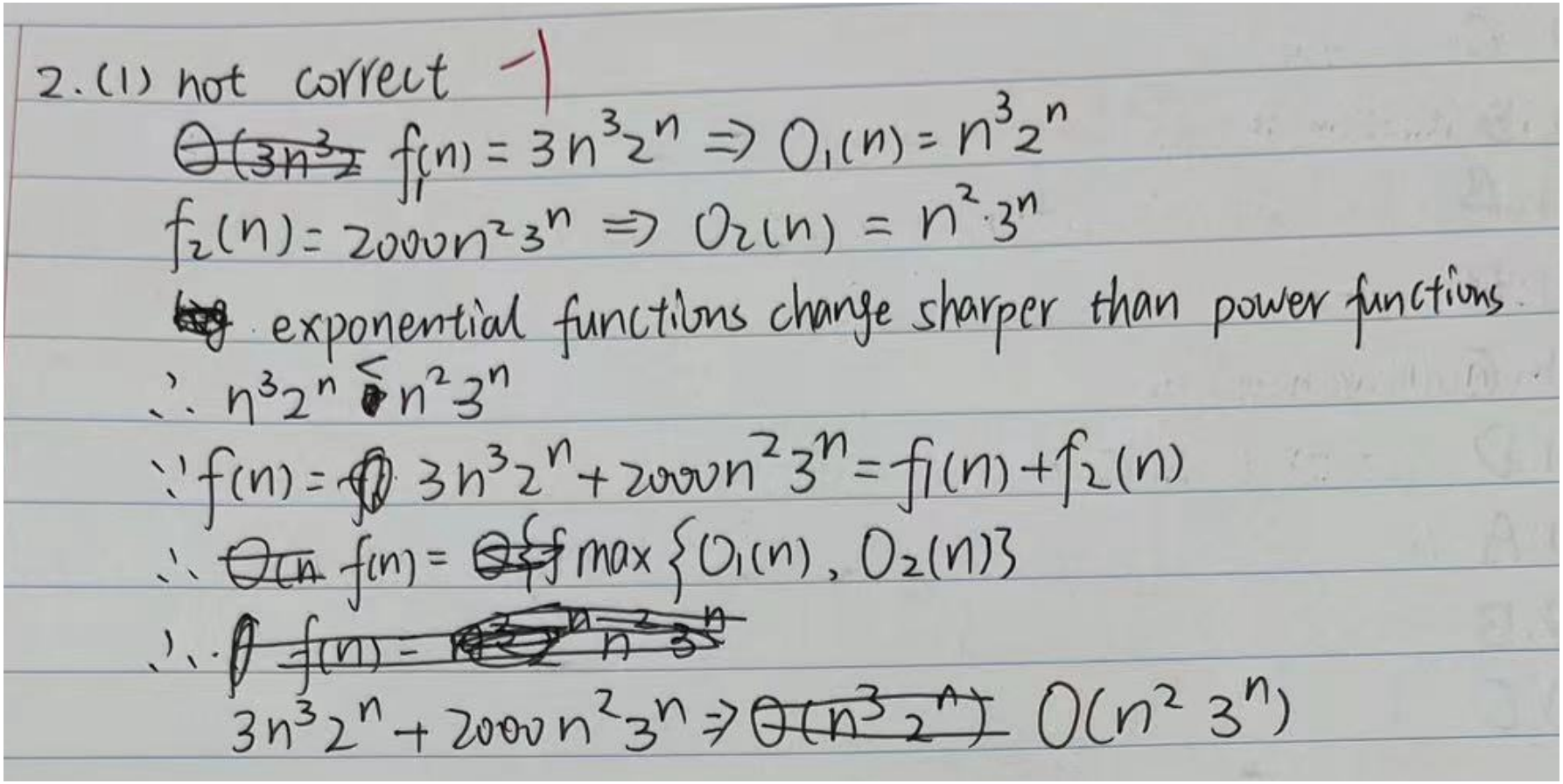
$3n^3 2^n + 200n^2 3^n \geq c \cdot n^3\, 3^n$ for all $n > n_0$

$\Rightarrow c \geq \frac{1}{3} \cdot \frac{2^n}{3^n} + \frac{2000}{n}$ for all $n > n_0$

When $c = 2, n_0 = 2000$, this is right.

$\therefore 3n^3 2^n + 200n^2 3^n = O(n^3 3^n)$

A typical error



Big-Oh is an **upper bound**

Asymptotic notation

$4n^3 + 5n^2 = \Omega(n^3)$

Proof:

We need to find c, n_0 satisfying that

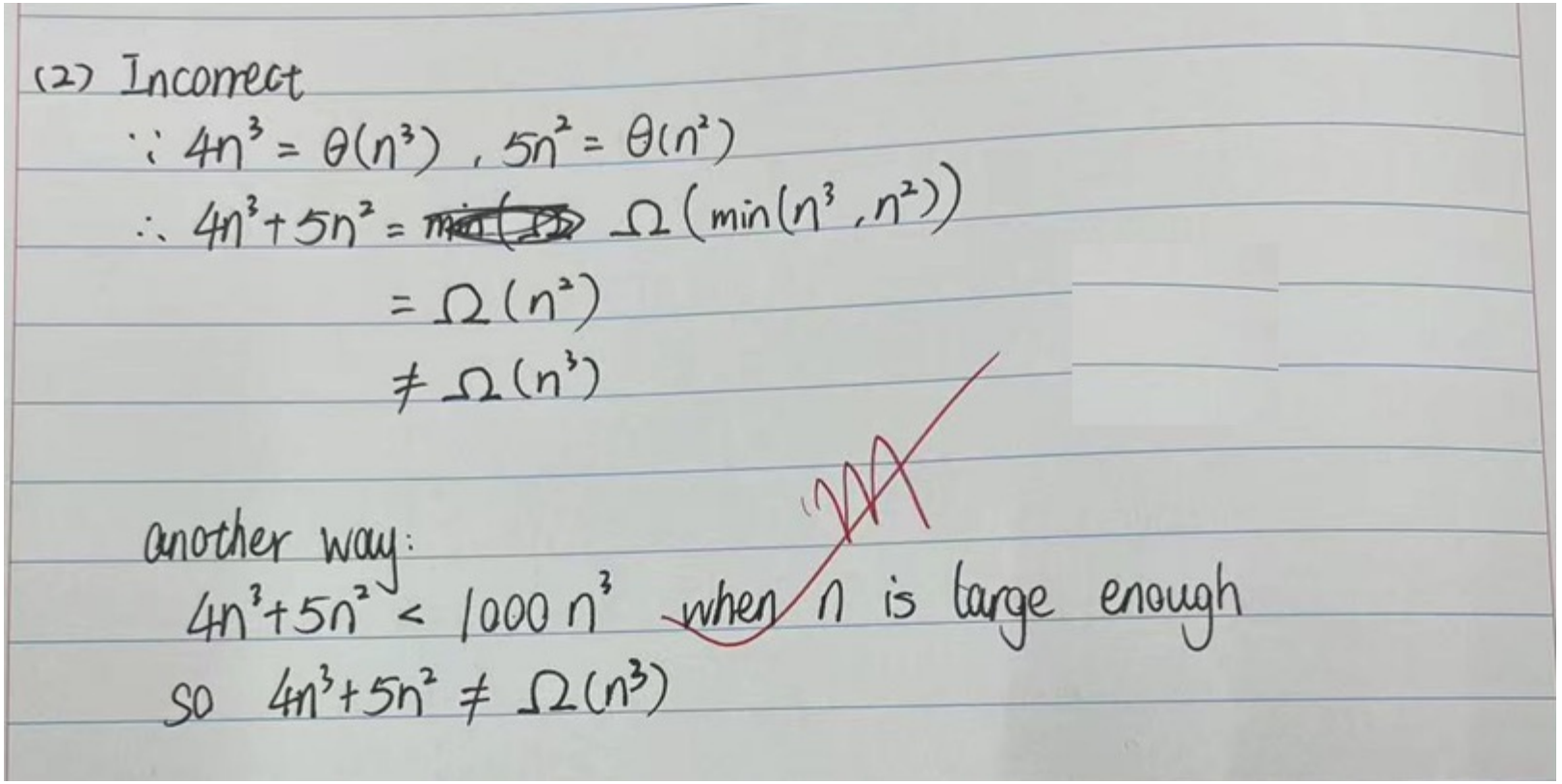
$4n^3 + 5n^2 \leq c \cdot n^3$ for all $n > n_0$

$\Rightarrow c \leq 4 + \frac{5}{n}$ for all $n > n_0$

When $c = 4, n_0 = 1$, this is right.

Hence, $4n^3 + 5n^2 = \Omega(n^3)$

A typical error



1. The biggest eats all
2. The second way misuses the definition

#3 Linked list

3. [15] Consider a singly linked list, where the head node does not contain data, and each remaining node *p* has a next pointer and a data element, namely *p.next* and *p.data*, where *p.data* is an integer value in [0, 100].
(1) [10 marks] Given a specific integer *x*, design an algorithm with pseudocodes to delete all the nodes whose data elements equal to *x*.
(2) [5 marks] Assume that there are *n* nodes ($n > 1$) in the above linked list. Given *n* arbitrary integer values, can we check which integer values of them appear in the above linked list using just $O(n)$ time cost in total? Please explain the reason.

Delete nodes with data "x"

```
Node p = null;
p = head;
/* Pick elements one by one */
while (p != null && p.next != null) {
    /* Check the data of node p.next */
    while (p.next != null && p.next.data == x) {
        /* Delete node p.next */
        p.next = p.next.next;
    }
    p = p.next;
}
```

Delete nodes (correct)

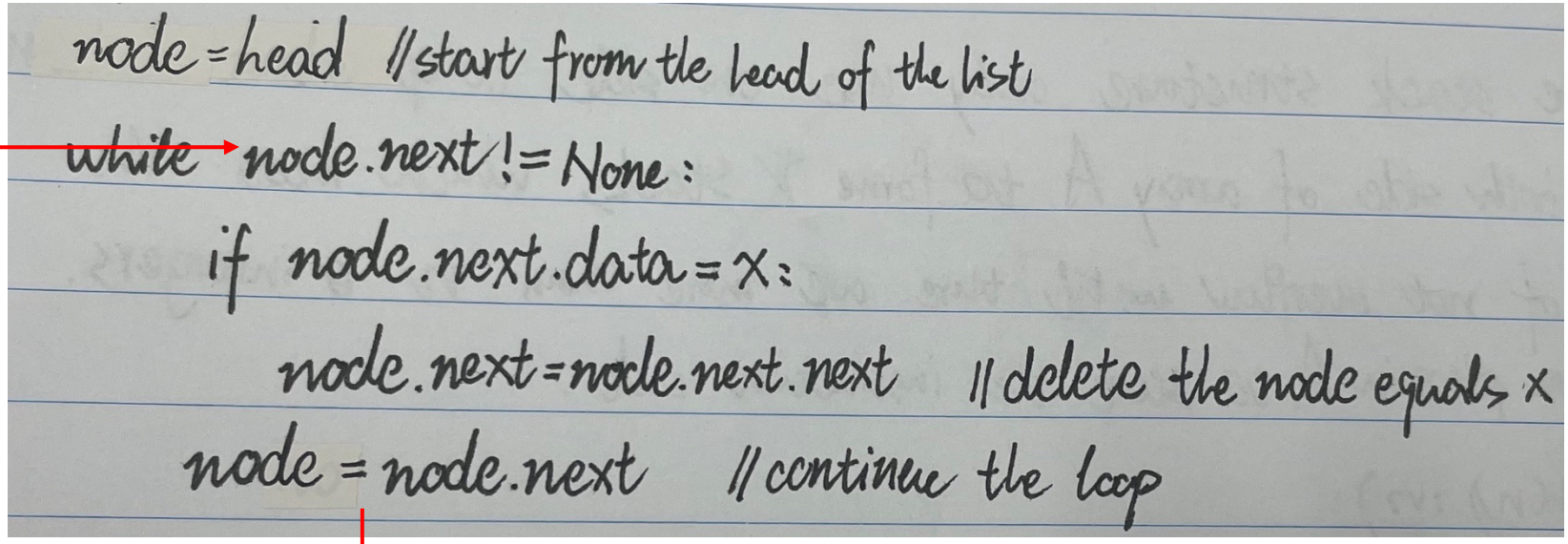
```
delete(x){
  y = head.next , pre = head
  while y ≠ NIL:
    if y.data = x:
      do pre.next = y.next
         y.next = NIL
         y = pre.next
    else
      pre = y
      y = y.next
```

```
(1) void deletenode(int x) {
      node tmp = head;
      while (tmp != NULL && tmp.next != NULL) {
        while (tmp.next != NULL && tmp.next.data == x)
          tmp.next = tmp.next.next;
        tmp = tmp.next;
      }
    }
```

Delete one node each step

Use an inner loop to delete all consecutive nodes with data x

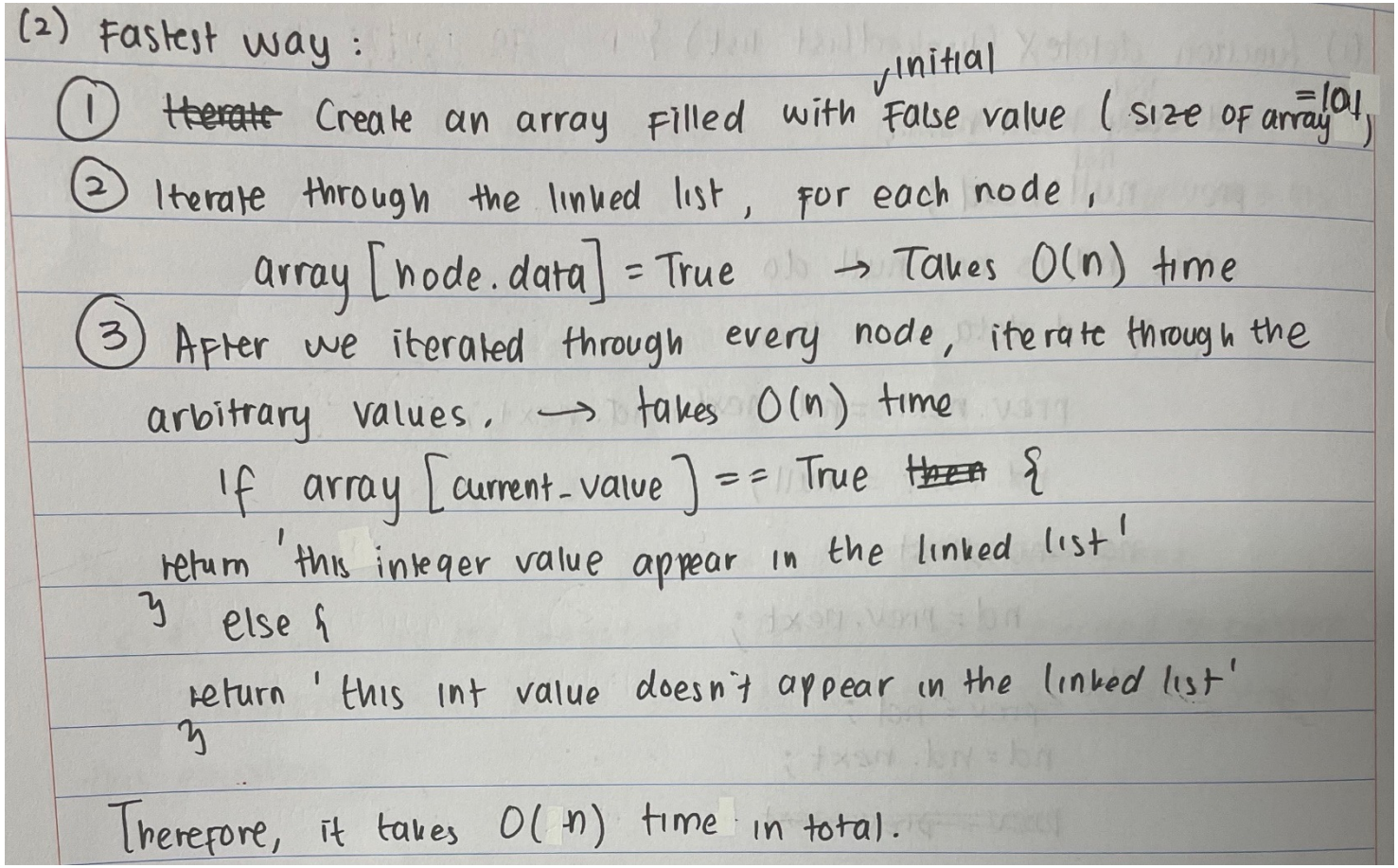
Delete nodes (wrong)



```
node = head  // start from the head of the list
while node.next != None:
    if node.next.data = x:
        node.next = node.next.next   // delete the node equals x
    node = node.next   // continue the loop
```

node can be "None" here, then "None.next" will raise an error

Check integer values (correct)



(2) Fastest way:

① ~~Iterate~~ Create an array filled with initial False value (size of array = |a|)

② Iterate through the linked list, for each node,

array[node.data] = True → Takes O(n) time

③ After we iterated through every node, iterate through the arbitrary values. → takes O(n) time

if array[current_value] == True ~~then~~ {

return 'this integer value appear in the linked list'

} else {

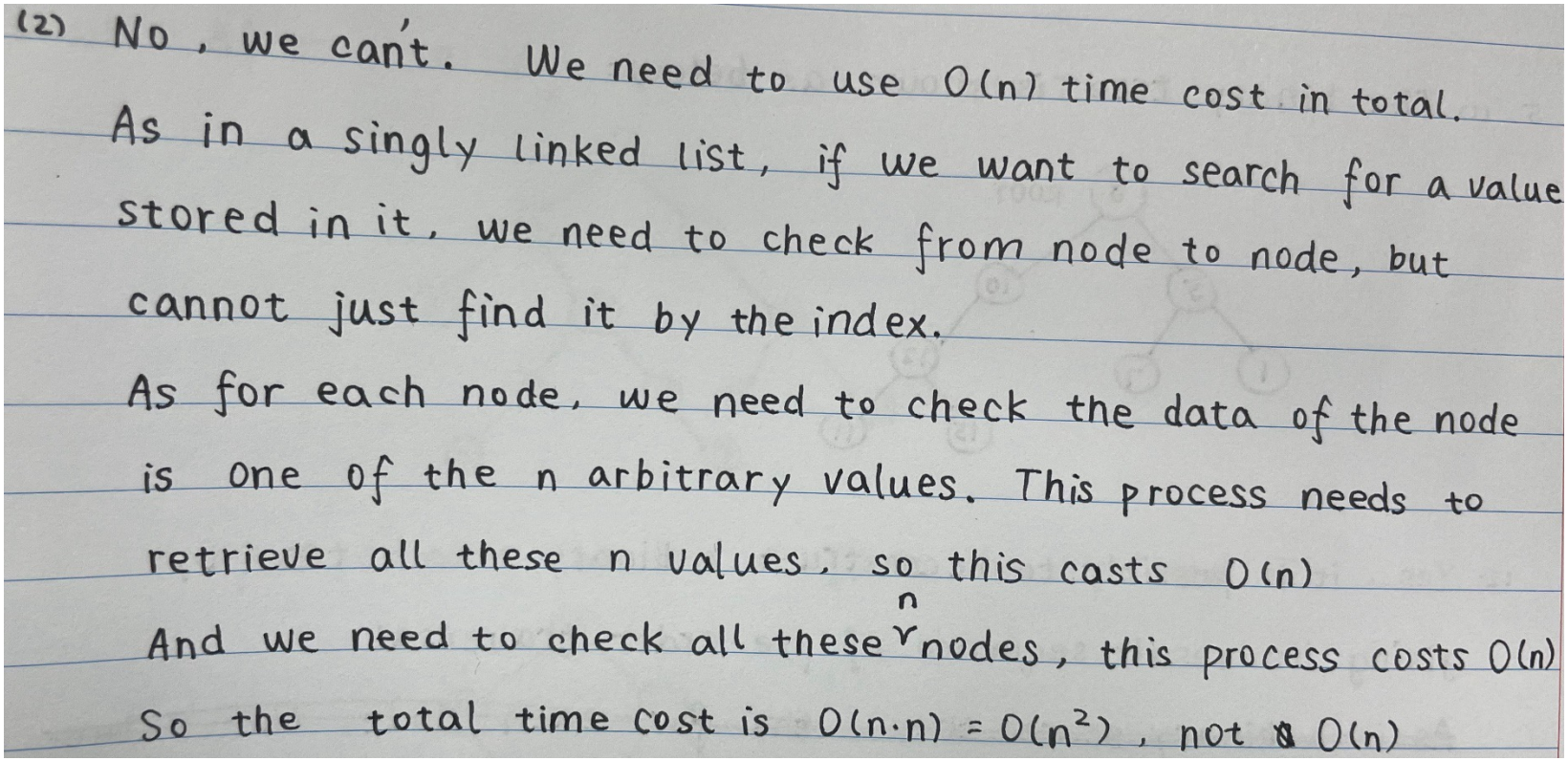
return 'this int value doesn't appear in the linked list'

}

Therefore, it takes O(n) time in total.

Use a Boolean array to indicate which integers appear in the linked list

Check integer values (wrong)



(2) No, we can't. We need to use O(n) time cost in total.
As in a singly linked list, if we want to search for a value stored in it, we need to check from node to node, but cannot just find it by the index.
As for each node, we need to check the data of the node is one of the n arbitrary values. This process needs to retrieve all these n values, so this casts O(n)
And we need to check all these n nodes, this process costs O(n)
So the total time cost is $O(n \cdot n) = O(n^2)$, not O(n)

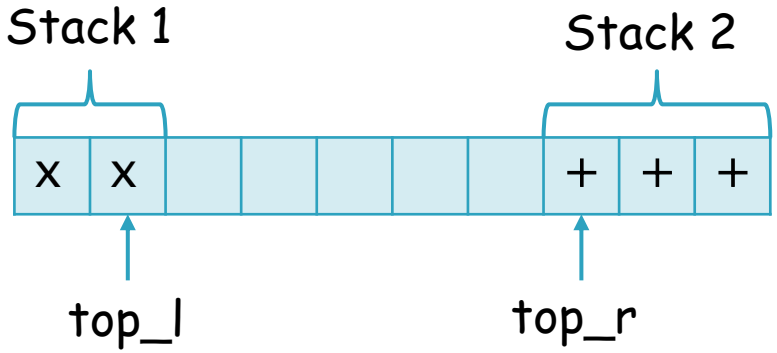
It takes $O(n^2)$ time if we check each value separately

#4 STACK

4. [15 marks] Design two stacks in one array A[*1*...*n*] (*n* > *1*) in such a way that neither stack overflows unless the total number of elements in both stacks together is *n*. The PUSH and POP operations should run in *O(1)* time. Please briefly explain how to implement the two stacks, and then use pseudocodes to explain the steps of PUSH and POP for them.

Implement two stacks in one array

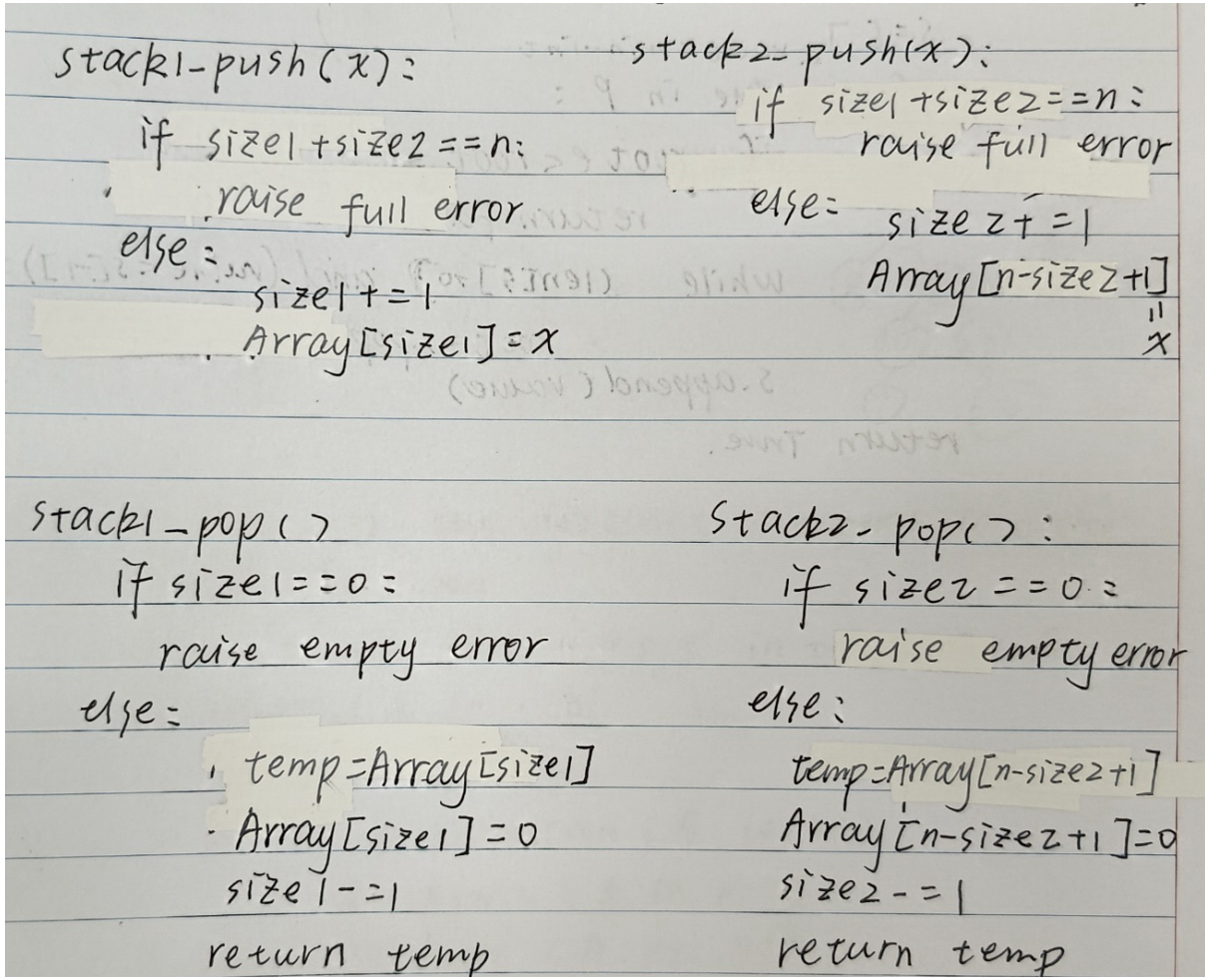
```
1.  Array A[1…n]
2.  top_l = 0, top_r = n+1

3.  def isFull():
4.      return top_r - top_l < 2

5.  def isEmptyS1():
6.      return top_l <= 0

7.  def pushS1(data):
8.      if(isFull()){
9.          return false;
10.     }
11.     A[++top_l] = data;
12.     return true;
```




```
1.  def popS1():
2.      if(isEmptyS1()){
3.          return null;
4.      }
5.      return A[top_l--];
```

We can implement Stack 2 similarly using top_r

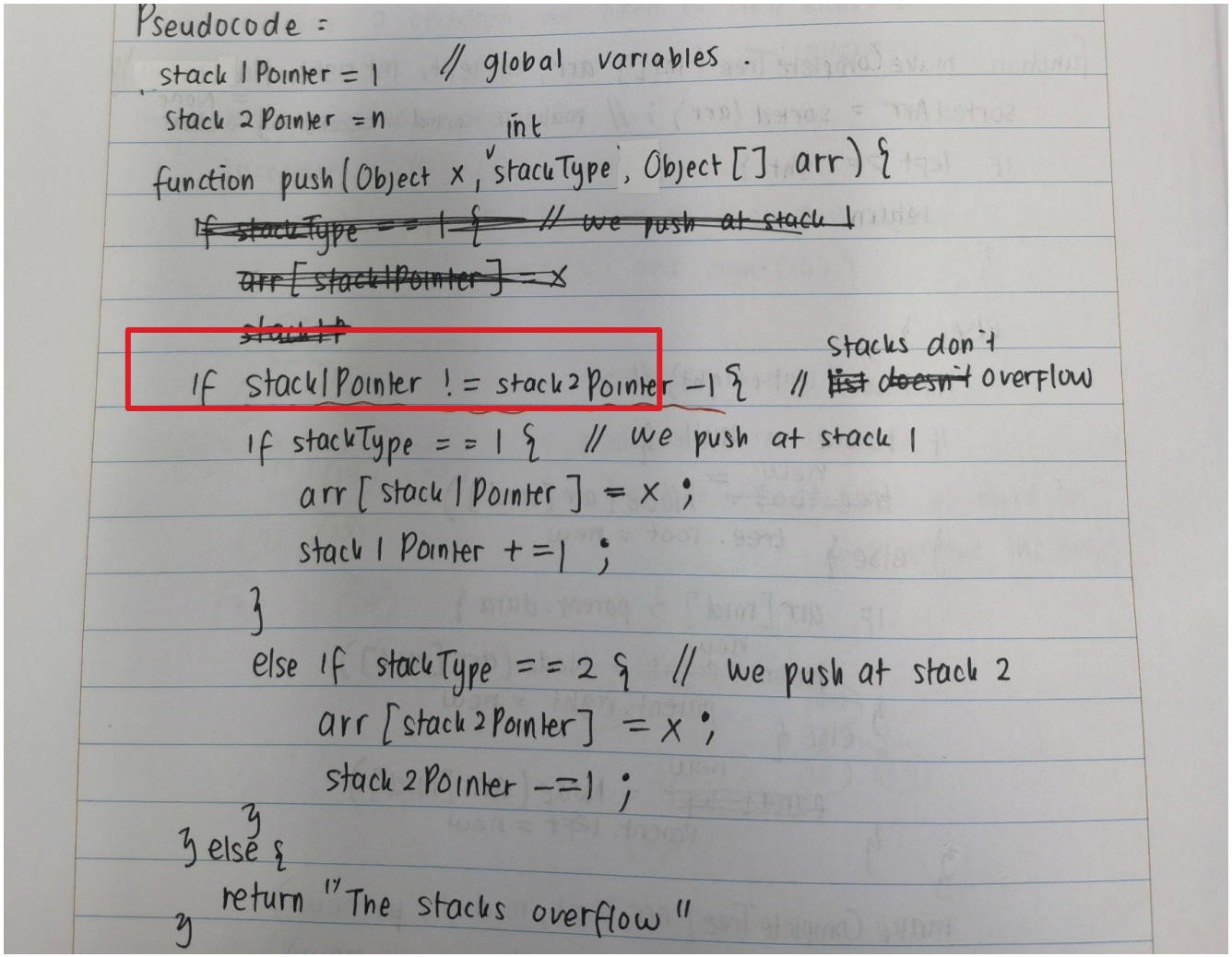
An example answer



```
stack1_push(x):
    if size1+size2==n:
        raise full error
    else:
        size1+=1
        Array[size1]=x

stack2_push(x):
    if size1+size2==n:
        raise full error
    else:
        size2+=1
        Array[n-size2+1]=x

stack1_pop():
    if size1==0:
        raise empty error
    else:
        temp=Array[size1]
        Array[size1]=0
        size1-=1
        return temp

stack2_pop():
    if size2==0:
        raise empty error
    else:
        temp=Array[n-size2+1]
        Array[n-size2+1]=0
        size2-=1
        return temp
```

Store the size of each stack instead of a pointer

A typical error



```
Pseudocode :
stack1Pointer = 1        // global variables.
stack2Pointer = n
                                    int
function push(Object x, stackType, Object[] arr) {
  if stackType == 1 {  // we push at stack 1
    arr[stack1Pointer] = x
    stack1++
  if stack1Pointer != stack2Pointer - 1 {   // list doesn't overflow
                                              Stacks don't overflow
    if stackType == 1 {  // we push at stack 1
      arr[stack1Pointer] = x;
      stack1Pointer += 1;
    }
    else if stackType == 2 {  // we push at stack 2
      arr[stack2Pointer] = x;
      stack2Pointer -= 1;
    }
  } else {
    return "The stacks overflow"
  }
```

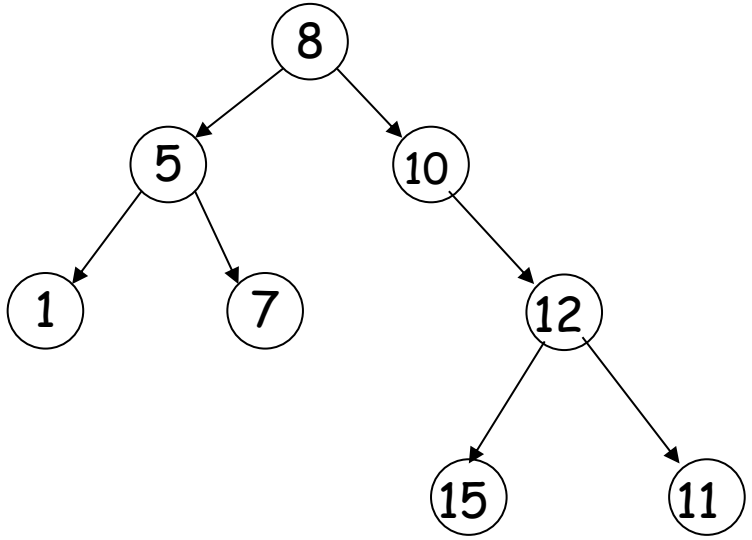
Incorrect overflow condition

#5 Binary tree

5. [16 marks] Tree reconstruction.
(1) [12 marks] Construct a binary tree such that its pre-order = [8,5,1,7,10,12,15,11] and in-order = [1,5,7,8,10,15,12,11].
(2) [4 marks] Given a binary search tree *T* and its result sequence of pre-order traversal, is it possible to reconstruct *T*? Please justify your answer using a few sentences.

Reconstruction of binary trees

In-order : { 1,5,7,8,10,15,12,11 }
Pre-order: { 8,5,1,7,10,12,15,11 }

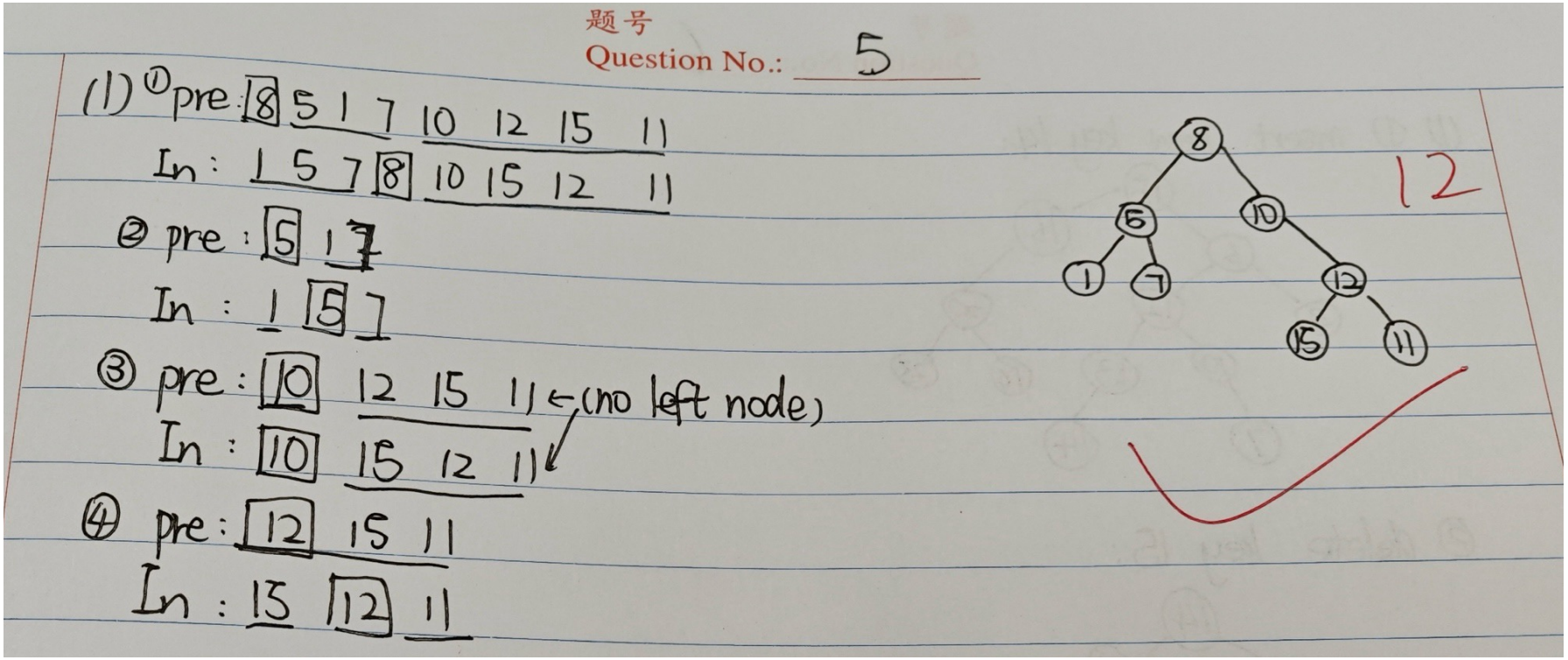


Yes. Given a binary search tree, its in-order is the result of sorting all the keys in ascending order, so we can reconstruct the tree as follows:

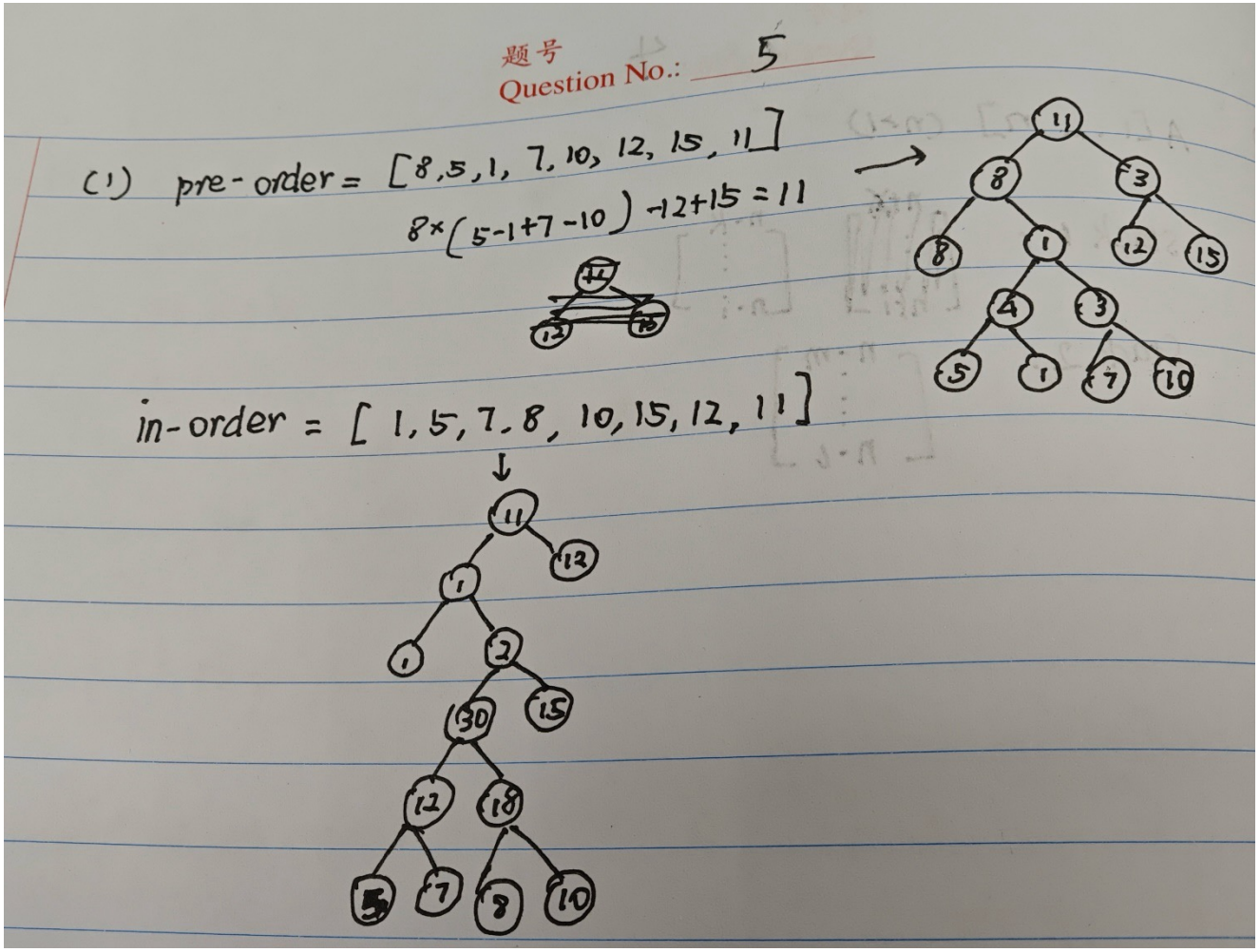
(1) Sorting all the keys in the pre-traversal;

(2) Given the pre-order and in-order results, we can use the recursion method in the lecture notes to reconstruct the tree.

Reconstruct binary tree (correct)

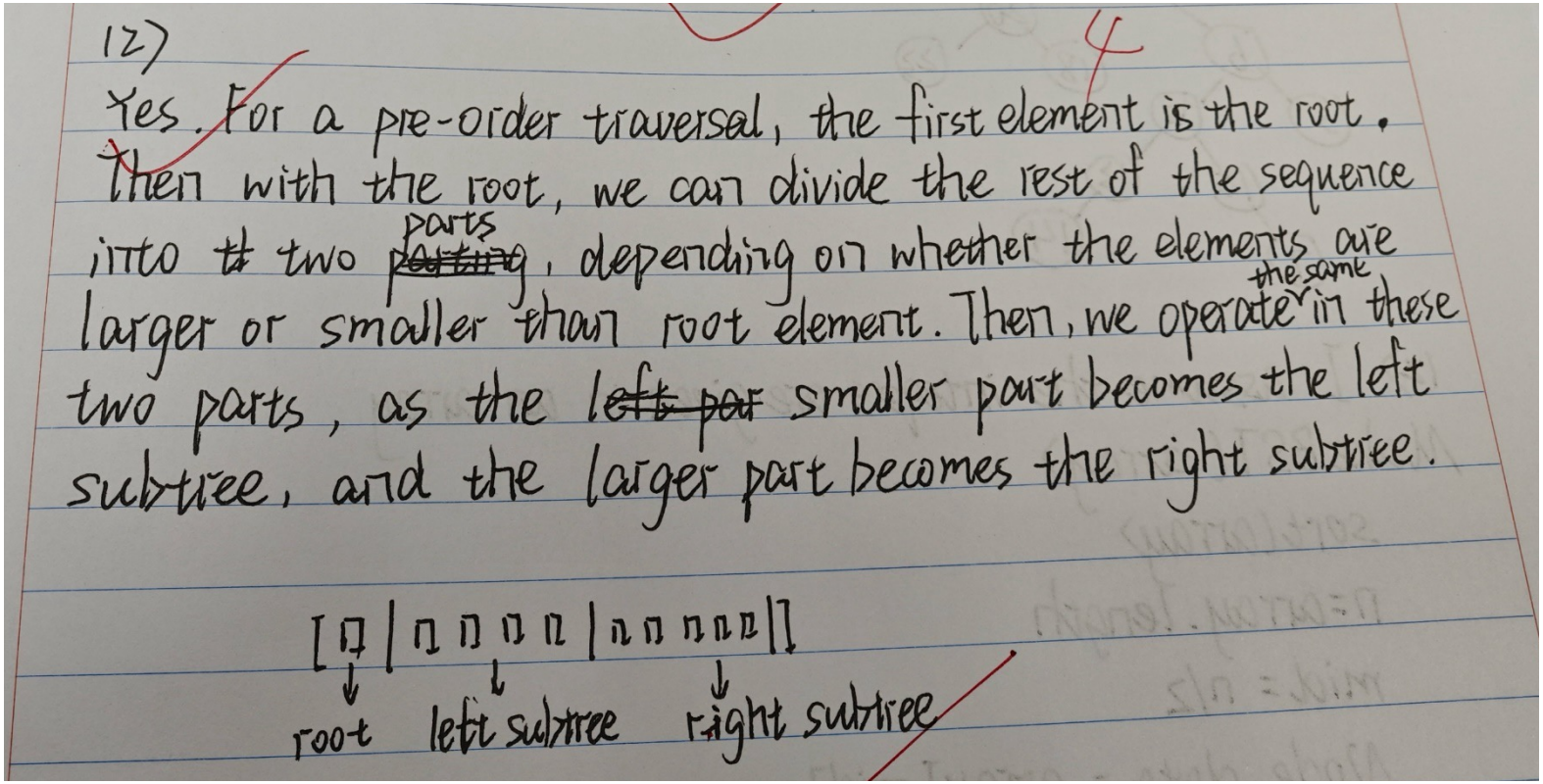


Reconstruct binary tree (wrong)



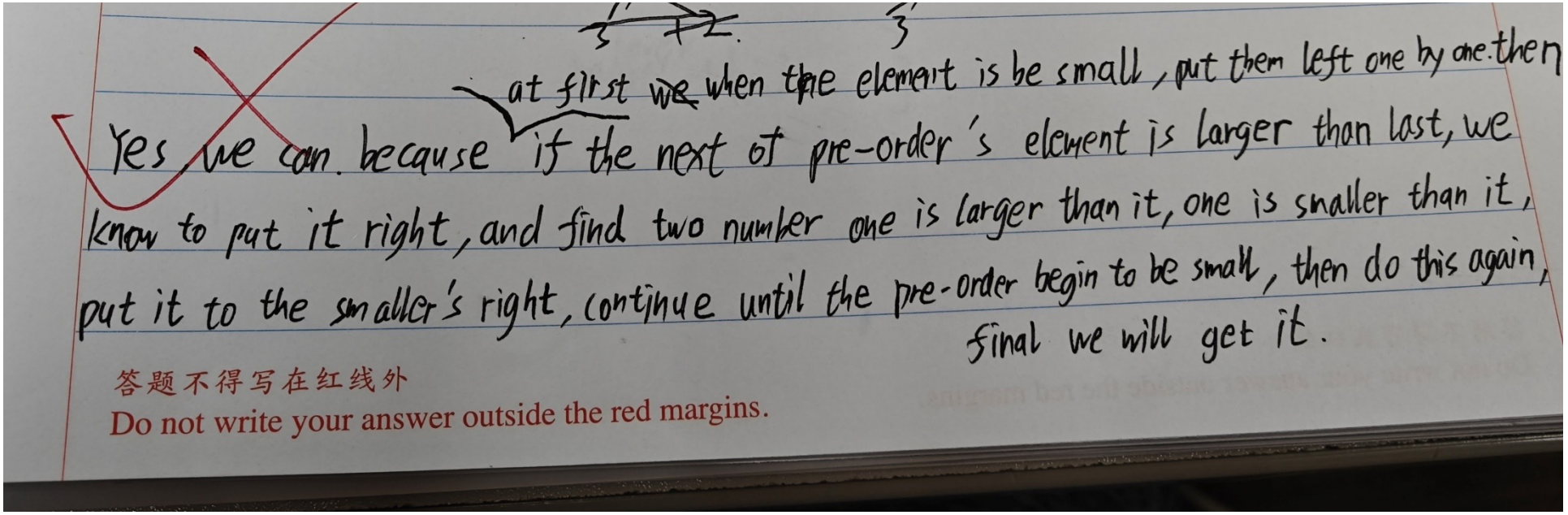
The first node in pre-order should be root.

Reconstruct BST (correct)



The element value in BST is ordered

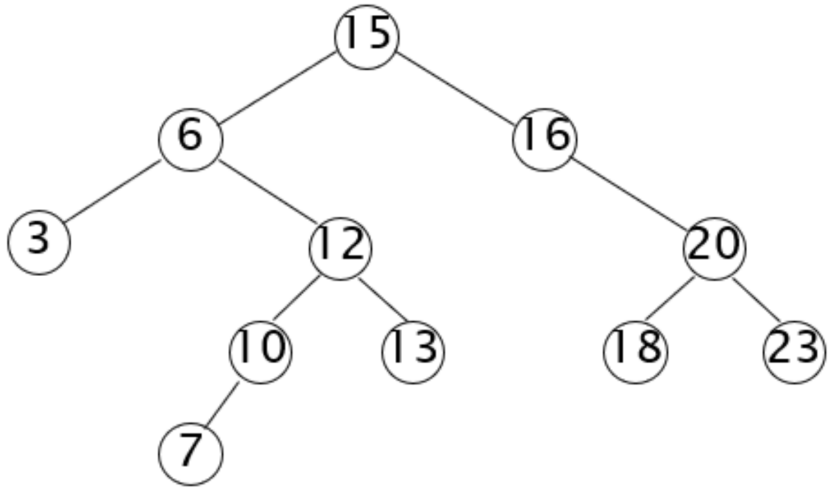
Reconstruct BST (wrong)



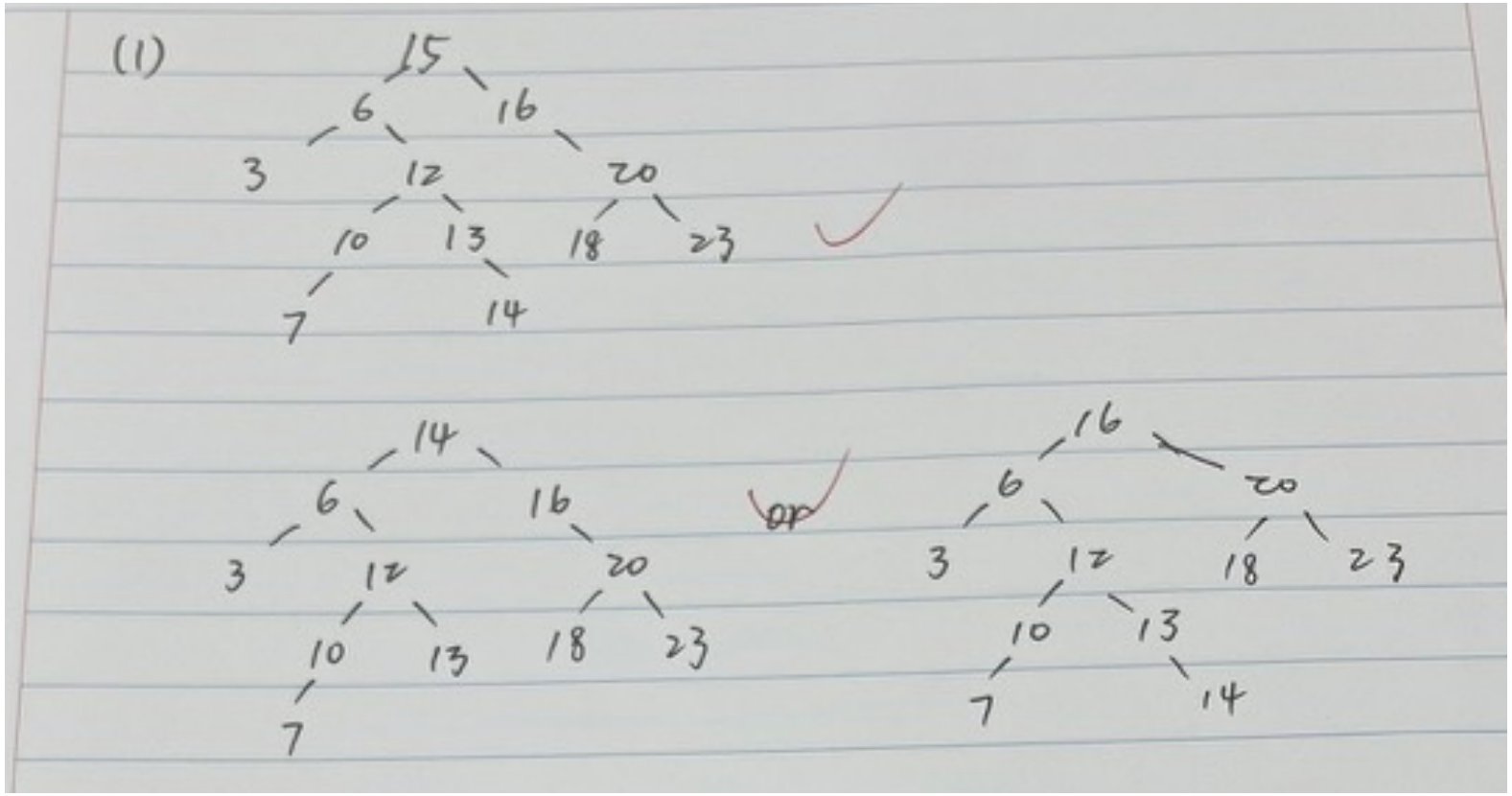
Misunderstand the pre-order of BST

#6 Binary Search tree

6. [15 marks] Binary search tree.
(1) [10 marks] Consider the binary search tree T in the following figure. Perform two consecutive operations on T: first insert a new key 14 into T, and then delete the key 15 from the updated T. Please draw the two updated trees after these operations respectively.
(2) [5 marks] Given n distinct integer keys with $n = (2^d - 1)$ where d is a positive integer, how to build a complete binary search tree (where all the levels are completely filled)? Please show your pseudocodes.

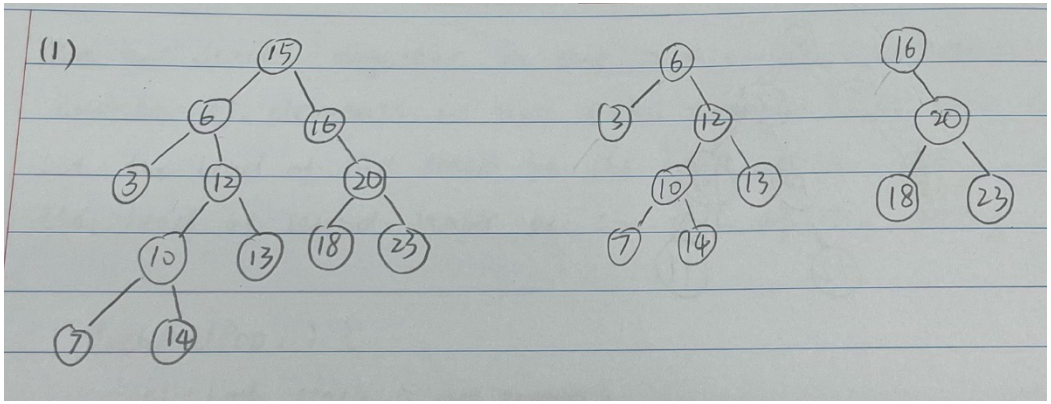


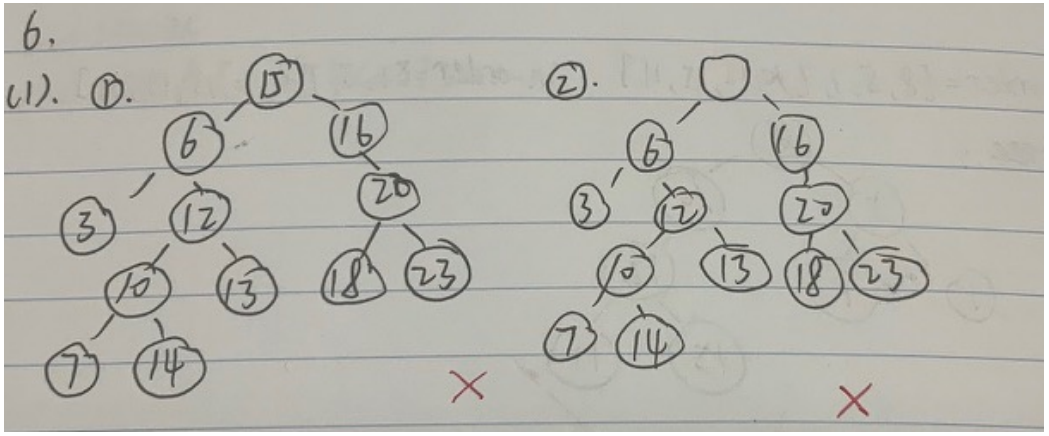
Insert and delete key (correct)



2 different answers for "delete key 15"

Insert and delete key (wrong)

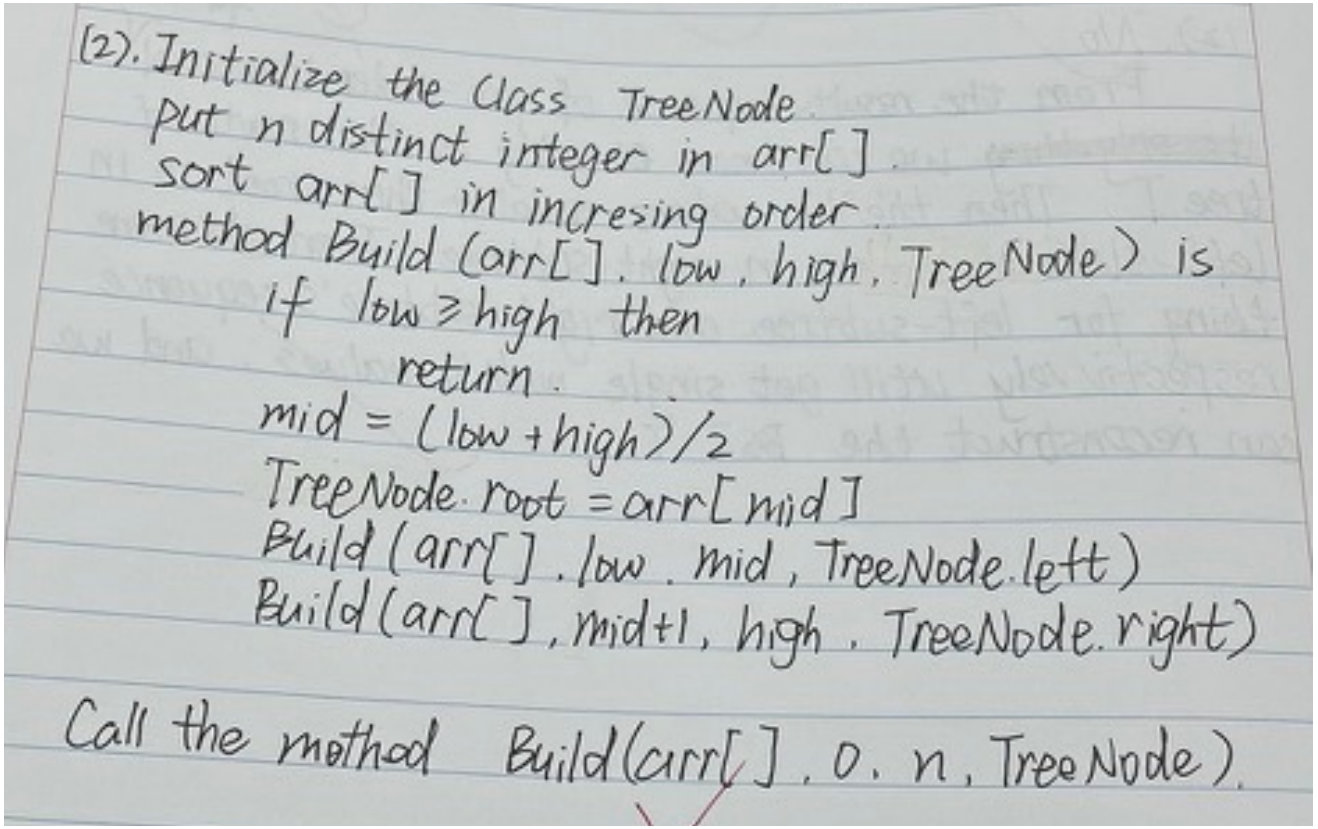




Build complete BST

```
1.  keys.sort()
2.  def build_tree(keys, start, end):
3.      if start > end:
4.          return None
5.
6.      mid = (start + end) // find the middle point
7.      root = Node(keys[mid])
8.      root.left = build_tree(keys, start, mid - 1)
9.      root.right = build_tree(keys, mid + 1, end)
10.
11.     return root
12. root = build_tree(keys, 0, len(keys)-1)
```

Build complete BST (correct)

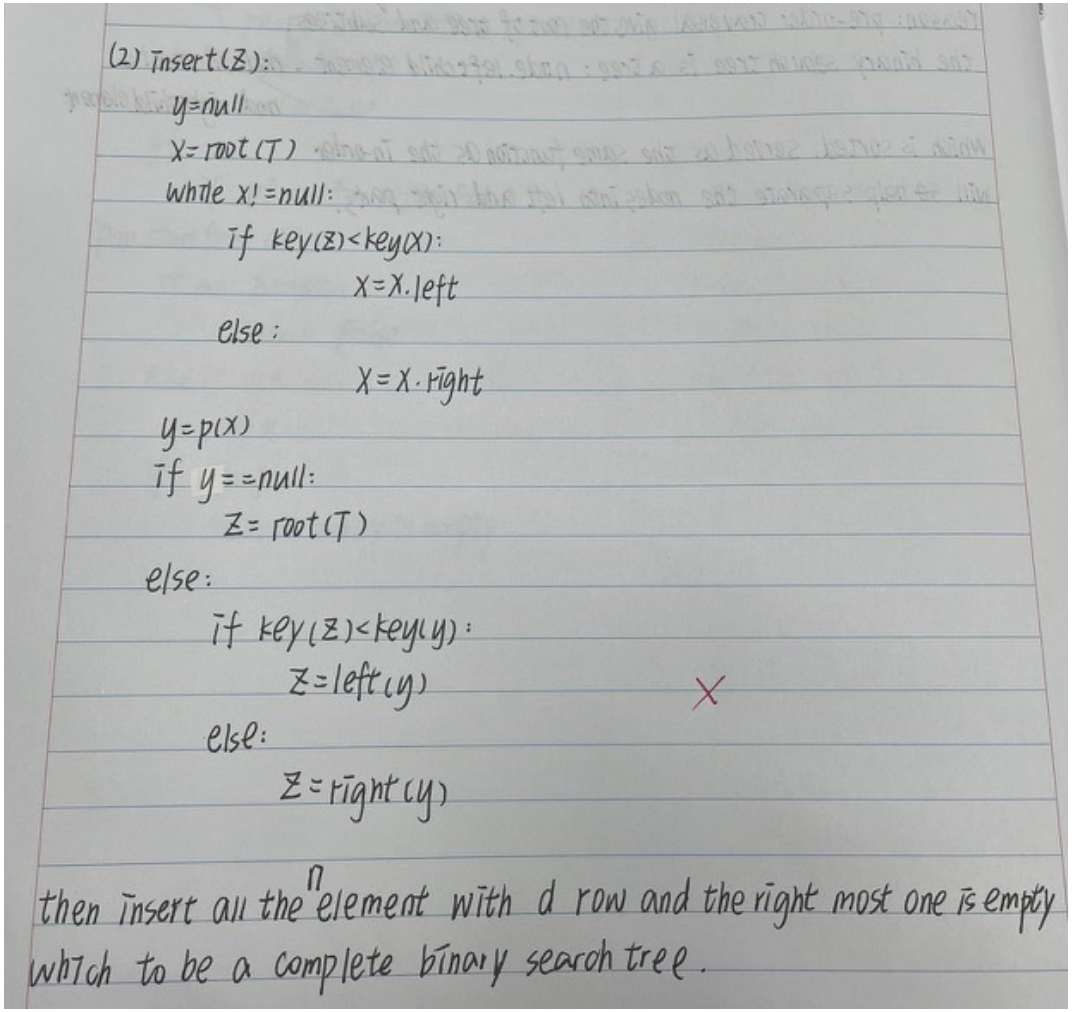


(2). Initialize the class TreeNode.
Put n distinct integer in arr[].
Sort arr[] in increasing order.
method Build (arr[], low, high, TreeNode) is
if low ≥ high then
return.
mid = (low + high)/2
TreeNode.root = arr[mid]
Build (arr[], low, mid, TreeNode.left)
Build (arr[], mid+1, high, TreeNode.right)

Call the method Build(arr[], 0, n, TreeNode).

Use recursion

Build complete BST (wrong)



```
(2) Insert(Z):
    y=null
    X=root(T)
    while X!=null:
        if key(Z)<key(X):
            X=X.left
        else:
            X=X.right
    y=p(X)
    if y==null:
        Z=root(T)
    else:
        if key(Z)<key(y):
            Z=left(y)
        else:
            Z=right(y)
```

then insert all the n element with d row and the right most one is empty which to be a complete binary search tree.

It does not guarantee "**complete**"