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

Acknowledgement to Prof. Wang Benyou for preparing midterm exam lecture notes

Overview

- #1 and #2 **Asymptotic** notation (10 * 2)
- #3 linked **list** （**remove** duplicates & find middle node） (20)
- #4 postfix expression (**stack**) (15)
- #5 Traversal of binary **trees** (20)
- #6 BST (Determine **BST &** merge BSTs) (25)

First read these notes

Notes:

First, your bound should be as tight as possible (as if you are using Θ notation) when using use O notation.

Second, you could reuse the following definition for Node, Stack, Queue, and BST in your pseudocodes or codes. You could assume that the data to be stored is only a single integer (for Node, Stack, Queue) for simplicity.

```
class Node {    int value;
                Node next;}
Class Stack{
        int pop();
        void push(int num); }
class Queue{
        int dequeue();
        void enqueue(int num);}
class BSTNode{
        BSTNode left;
        BSTNode right;
        int value;
        BSTNode parent;   // (optional)
        void insert(int key);
        void delete(int key);
}
```

Last, hope that you could enjoy solving these problems.

#1 Asymptotic notation

1.[10 marks] For a log function $f(n) = \log_b n$ with a base b>1, state and prove whether $f(n) = \Theta(\log_2 n)$.

Hint: $\log_a n = \log_b n \,/ \log_b a$; prove both $f(n) = O(\log_2 n)$ and $f(n) = \Omega(\log_2 n)$.

2. [10 marks] check and prove $g(n) = (0.1n^2 + n\log n) \cdot (n\log n + \sqrt{n}) = \Theta(n^3 \cdot \log n)$.

Asymptotic notation

For a log function f(n)= log_bn with a base b>1, state and prove whether f(n)=Θ(log_2n). Hint: log_an =log_bn / log_ba; prove both g(n)=O(n) and g(n)= Ω(n).

Proof:

Step 1: we can find $c = 1/\log_2 b$, $n_0 = 1$ such that

$$\mathrm{f}(n) = \log_b n \le c \cdot \log_2 n \text{ for } n \ge 1$$

Therefore $\mathrm{f}(n) = \log_b n = O(\log_2 n)$

Step 2: we can find $c = 1/\log_2 b$, $n_0 = 1$ such that

$$\mathrm{f}(n) = \log_b n \ge c \cdot \log_2 n \text{ for } n \ge 1$$

Therefore $\mathrm{f}(n) = \log_b n = \Omega(\log_2 n)$

Based step 1 and 2, we conclude that $\mathrm{f}(n) = \Theta(log_2\mathrm{n})$

Asymptotic notation

check and prove $g(n) = (0.1n^2 + n\log n)\cdot(n\log n + \sqrt{n}) = \Theta(n^3 \cdot \log n)$.

Proof:

$g(n) = g_1(n) * g_2(n); g_1(n) = (0.1n^2 + n\log n)\ and\ g_2(n) = (n\log n + \sqrt{n})$

Step 1: $g_1(n) = O(n^2)$, $g_2(n) = O(n\log n)$,

$g(n) = g_1(n) * g_2(n) = O(n^2 \ * n\log n) = O(n^3 \cdot \log n)$

Step 2: $g_1(n) = \Omega(n^2)$, $g_2(n) = \Omega(n\log n)$,

$g(n) = g_1(n) * g_2(n) = \Omega(n^2 \ * n\log n) = \Omega(n^3 \cdot \log n)$

Based step 1 and 2, we conclude that $f(n) = \Theta(log_2 n)$

Asymptotic notation

check and prove $g(n) = (0.1n^2 + n\log n)\cdot(n\log n + \sqrt{n}) = \Theta(n^3 \cdot \log n)$.

Proof:

Step 1: prove $g_1(n) = O(n^2)$

$g_1(n) = 0.1n^2 + n\log n$

$\leq 1.1\, n^2 \quad when\, n > 0 \quad (since\ \log n < n,\ n\log n < n^2)$

we could find $c=1.1$ *and* $n_0=0$ *such that* $g_1(n) \leq cn^2$ *when* $n_0 < n$

prove $g_2(n) = O(cn\log n)$

$g_1(n) = n\log n + \sqrt{n}$

$= \sqrt{n}\,(\sqrt{n}\log n + 1)$

$\leq \sqrt{n}\,(\sqrt{n}\log n + \sqrt{n}\log n)$

$\leq 2n\log n \quad when\, n > e \quad (when\ n>e,\ 1<\sqrt{n}\log n)$

we could find $c=2$ *and* $n_0=e$ *such that* $g_2(n) \leq cn\log n$ *when* $n_0 < n$

Examples of Big-O

- For any linear function $g(n) = a \cdot n + b$, with $a \geq 0, b \geq 0$
  - $g(n) = a \cdot n + b = O(n)$

Proof: We can find $c = (a + b),\ n_0 = 1$ such that

$$g(n) = an + b \leq an + bn = (a + b)n \text{ for } n \geq 1$$

Therefore $g(n) = O(n)$

- For two log functions with different base $a > 1$ and $b > 1$
  - $g(n) = \log_a n = O(\log_b n)$

$\log_a n = \log_b n / \log_b a$

Proof: We can find $c = 1/\log_b a,\ n_0 = 1$ such that

$$g(n) = \log_a n \leq c \cdot \log_b n \text{ for } n \geq 1$$

Therefore $g(n) = \log_a n = O(\log_b n)$

For Big-O, base doesn't matter, so

$$O(\log_b n) = O(\log n)$$

$$O(\log n) = O(\log_{10} n) \text{ or } O(\log n) = O(\log_2 n)$$

Asymptotic notation

check and prove $g(n) = (0.1n^2 + n\log n) \cdot (n\log n + \sqrt{n}) = \Theta(n^3 \cdot \log n)$.

Proof:

Step 2: prove $g_1(n) = \Omega(n^2)$

$$g_1(n) = 0.1n^2 + n\log n$$
$$\geq 0.1\, n^2 \quad when\, n > 0 \quad (minus\ a\ positive\ term)$$

we could find $c=0.1$ *and* $n_0=0$ *such that* $g_1(n) \geq cn^2$ *when* $n_0 < n$

prove $g_2(n) = \Omega(cn\log n)$

$$g_1(n) = n\log n + \sqrt{n}$$
$$\geq n\log n \quad when\, n > 0 \quad (minus\ a\ positive\ term)$$

we could find $c=1$ *and* $n_0=0$ *such that* $g_2(n) \geq cn\log n$ *when* $n_0 < n$

A typical error

题号
Question No.: 2

$g(n) = (0.1n^2 + n\log n)\cdot(n\log n + \sqrt{n})$

$= 0.1n^3\log n + 0.1n^2\cdot\sqrt{n} + n^2(\log n)^2 + n\cdot(\log n)\cdot\sqrt{n}$

$= 0.1n^3\log n + 0.1n^{\frac{5}{2}} + n^2(\log n)^2 + n^{\frac{3}{2}}\log n$

As we know that $(\lg n) = O(n)$

$n^{\frac{1}{2}} = O(n)$

So $0.1n^3\log(n) = \Theta(n^3\log n)$

$0.1n^{\frac{5}{2}} = 0.1n^2\cdot n^{\frac{1}{2}} = O(n^3) = O(n^3\log n)$

$n^2(\log n)^2 = (n^2\cdot\lg n)\cdot\lg n = O(n^3\cdot\log n)$

$n^{\frac{3}{2}}\log n = n^2\cdot n^{\frac{1}{2}}\log n = O(n^2\cdot n\cdot\lg n) = O(n^3\log n)$

So combine them together

$g(n) = \Theta(n^3\log n) + O(n^3\log n) + O(n^3\log n) + O(n^3\cdot\log n)$

$= \Theta(n^3\log n)$

Prove big O only

An example (not that good)

The Chinese University of Hon

题号
Question No.: 2

$g(n) = (0.1n^2 + n\log n)(n\log n + \sqrt{n}) = f(n) \cdot h(n)$

where $f(n) = 0.1n^2 + n\log n = \Theta(n^2)$

$h(n) = n\log n + \sqrt{n} = \Theta(n\log n)$

$\Rightarrow f(n) \cdot h(n) = \Theta(n^2 \cdot n\log n) = \Theta(n^3 \log n)$

$\Rightarrow g(n) = \Theta(n^3 \log n)$

It seems fine but not concrete as expected (-4 marks)

#3 Linked list

3. [20 marks] Linked list

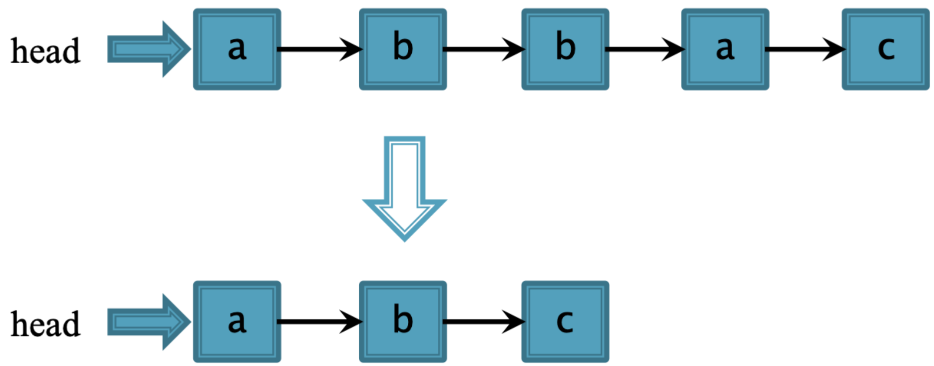
(1) [10 marks] Given an unsorted list of nodes, design an algorithm with pseudocodes or codes to remove duplicates from the list. Keep only the first-appeared node for the nodes with a same value.

For example, for a linked list 12->11->**12**->21->41->43->**21**, it returns a new one: 12->11->21->41->43. Because the second occurrence of 12 and 21 should be removed. We could assume that all numbers stored in nodes are positive integers that are smaller than 1000.

(2) [10 marks] design an algorithm with pseudocodes or codes to find the middle node of a linked list using a **single** loop and O(1) extra space. For example, return node 2 for a linked list 1->2->3; return node 2 or 3 (both are okay, you could return any of them) for a linked list 1->2->3->4.

Problem 2: Duplicate deletion

Given the head of a singly linked list *L*, in which each node's data is a lowercase letter, remove all the nodes with duplicate lowercase letters



Remove duplicates O(n^2)

```
Node ptr1 = null, ptr2 = null, dup = null;
ptr1 = head;
/* Pick elements one by one */
while (ptr1 != null && ptr1.next  != null) {
    ptr2 = ptr1;
    /* Compare the picked element with rest
        of the elements */
    while (ptr2.next  != null) {
        /* If duplicate then delete it */
        if (ptr1.data == ptr2.next.data) {
            ptr2.next  = ptr2.next.next;
    }
        else /* This is tricky */ {
            ptr2 = ptr2.next;
        }
    }
    ptr1 = ptr1.next;
}
```

Remove duplicates O(n)

```
// Hash to store seen values
HashSet<Integer> hs = new HashSet<>();
/* Pick elements one by one */
node current = head;
node prev = null;
while (current != null) {
    int curval = current.val;
    // If current value is seen before
    if (hs.contains(curval)) {
        prev.next = current.next;
    }
    else {
        hs.add(curval);
        prev = current;
    }
    current = current.next;
```

Exercise 1: how to get the middle node?

- Given a linked list with its head node, write the java codes to find the middle node

```java
// find middle element in linkedlist
public Node findMiddleNode(Node head) {
    Node slowPointer, fastPointer;
    slowPointer = fastPointer = head;

    while(fastPointer !=null) {
        fastPointer = fastPointer.next;
        if(fastPointer != null && fastPointer.next != null) {
            slowPointer = slowPointer.next;
            fastPointer = fastPointer.next;
        }
    }

    return slowPointer;
}
```

Find the middle node

```
(2) middle-node (head):
        pointer = head
        fastpointer = head
        while ( fast_pointer != NULL):
            fastpointer = fast_pointer.next
            if ( fast_ pointer. next != NULL):
                pointer = pointer . next
                fast_pointer = fast_ pointer. next

        return pointer.
```

10

Using two loops

~~head = head.next~~ head = head.nıxt;

}

return ret;

}

12). node get_middle (node head) {

int len = 0; node ret = head;

while (head != Null){ head = head.next , len++; } ret = ret.next;

~~for~~ ~~len /= 2;~~ for (int i=1; i≤len/2; i++) ~~head=hea~~

return ret;

}

答题不得写在红线外

Do not write your answer outside the red margins.

#4 STACK

4. [15 scores] Evaluate a postfix expression using a stack.

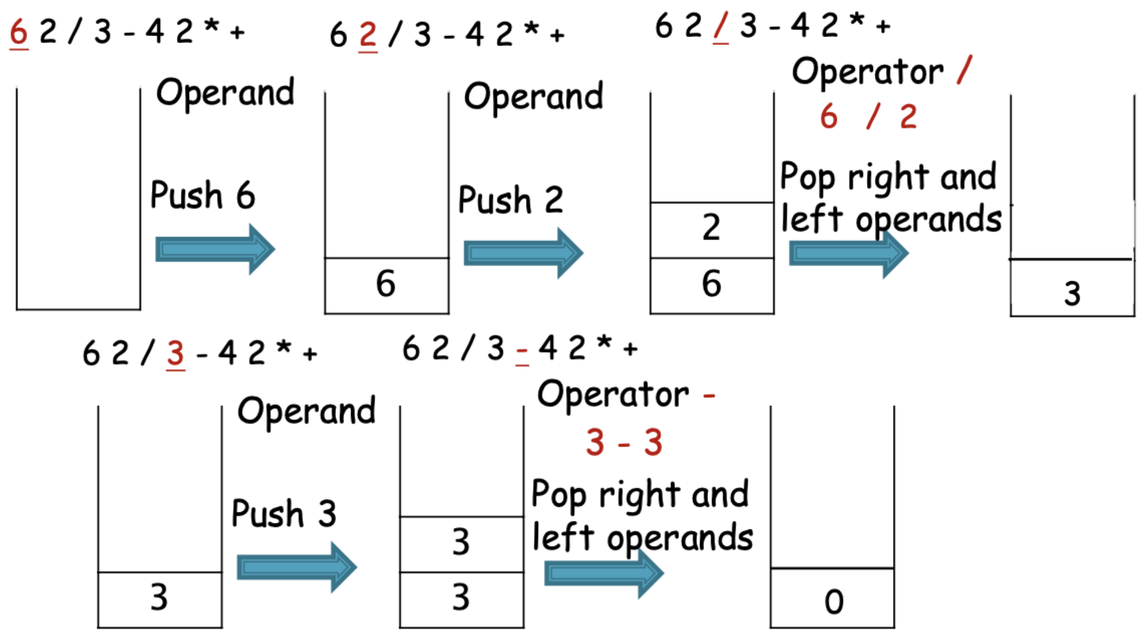
(1) [5 scores] briefly discuss the main idea using natural language. Plus, calculate the result for a given postfix expression: $545 * +5/$.

(2) [10 scores] write pseudocodes or codes to evaluate a given postfix expression (input is a postfix expression and return the evaluated result if it is valid, otherwise return error information). Assume that the postfix expression contains only single-digit numeric operands, without any whitespace. Only binary operators are considered, including four typical operators (+,-,*, and /).

A running example

Infix expression: (6/2-3) + (4*2)

- Evaluate the postfix expression: 6 2 / 3 - 4 2 * +



Evaluate a postfix expression

```
static int evaluatePostfix(String exp){
    Stack<Integer> stack=new Stack<>();
    for(int i=0;i<exp.length();i++){
      char c=exp.charAt(i);
      if(Character.isDigit(c))
        stack.push(c - '0');
      else{          {
        int val1, val2;
        if (stack)
            val1 = stack.pop();
        else
            raise;
        if (stack)
            val2 = stack.pop();
        else
            raise;
        if(c == '+')
          stack.push(val2+val1);
        else if(c == '-')
          stack.push(val2- val1);
        else if(c == '/')
          stack.push(val2/val1);
        else if(c == '*')
          stack.push(val2*val1);
    }  }  }
    return stack.pop();
  }
```

An example answer

). For a given postfix expression, we start ~~frir~~ from it's first charact
If it's a numpber, we push it into the stack and go ~~th~~ to th
next one. ~~Uuntil~~ we meet an operator, we first pop the numb
from the stack as the number right of the operator, then we
pop the left one. We do the calculation and push the result i
the stack. Then repeat the steps until the expression is empty.

⇒ 5 4 5 * + 5 / is (5 + 4 * 5) / 5 = 5

```
Evaluate (postfix expression)     let S be a stack
  let P be the input postfix expression.
  for i in P
    if i is a number
      PUSH (S, i)
    else
      right = POP (S)
      left = POP (S)
      if right is empty or left is empty
        return error
      if i == '+'
        new = left + right
      elif i == '-'
        new = left - right
      elif i == '*'
        new = left * right
      else
        new = left / right
      PUSH (S, new)
  if S is empty
    return error
  result = POP (S)
  return result
```

不得写在红线外
write your answer outside the red margins.

A typical error

```
left = num.pop().
if right and left:
    result = eval(str(right) + i + str(left)).
    num.push(result)
else:
    return error.
return num.pop().
```

-2

#5 Binary tree

5. [20 marks] Binary tree and its traversal sequences.

(1) [4 marks] give the reconstructed binary tree (if it has; give all if it has many) based on the following *preorder* and *inorder* sequences.

Preorder sequence: { 1, 2, 4, 3, 5, 7, 8, 6 } inorder sequence: { 4, 2, 1, 7, 5, 8, 3, 6 }

(2) [6 marks] with which two traversal sequences it could get a unique reconstructed binary tree?

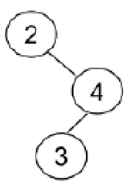
cases	Traversal sequence 1	Traversal sequence 2	unique reconstruction
Case #1	*preorder*	*inorder*	√ or ╳
Case #2	*inorder*	*postorder*	√ or ╳
Case #3	*preorder*	*postorder*	√ or ╳-

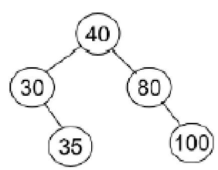
Fill the empty cells (in gray color) with either √ or ╳; the former √ (╳) indicates the reconstructed binary tree is (not) unique according to these two specified traversal sequences.

(3) [5 marks] If there exists one or many of the above three cases (#1, #2, and #3) where it cannot get a unique reconstructed binary tree, give an example to show different reconstructed binary trees under such a case. Otherwise, just specify it is always unique. You do not need to explain the reason.

(4) [5 marks] Given an array of numbers, design an algorithm with pseudocodes or codes to check if the given array can represent **preorder traversal** of an arbitrary **Binary Search Tree**. For example, the first two arrays are legal as a preorder traversal of a BST (as shown), while the last one is illegal as a preorder traversal of any BSTs. The algorithm should return True for the first two arrays and return False for the last array.

#1 {2, 4, 3} *#2 {40, 30, 35, 80, 100}* *#3 {40, 30, 35, 20, 80, 100}*



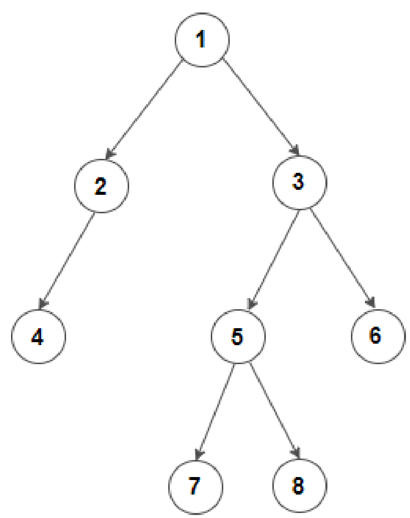


illegal preorder traversal

Reconstruction of binary trees

- **Input:**
  Inorder Traversal : { 4, 2, 1, 7, 5, 8, 3, 6 }
  Preorder Traversal: { 1, 2, 4, 3, 5, 7, 8, 6 }

  Output: Below binary tree



Reconstruction of binary trees

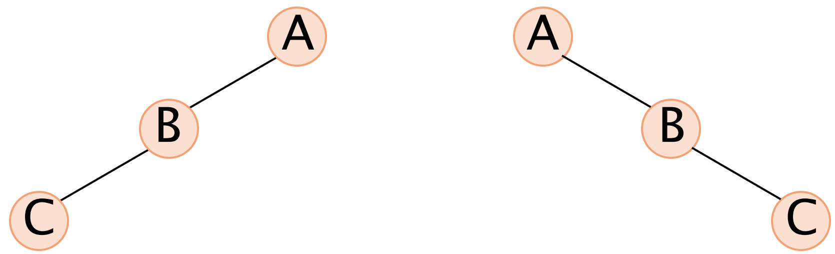
Cases	Traversal sequence 1	Traversal sequence 2	unique reconstruction
Case#1	Preorder	Inorder	√
Case#2	Inorder	Postorder	√
Case#3	preorder	Postorder	×

Reconstruction of binary trees

But: A binary tree may not be uniquely defined by its preorder and postorder sequences.

Example: **Preorder sequence: ABC**
Postorder sequence: CBA

We can construct 2 different binary trees:



legal preorder traversal of BST

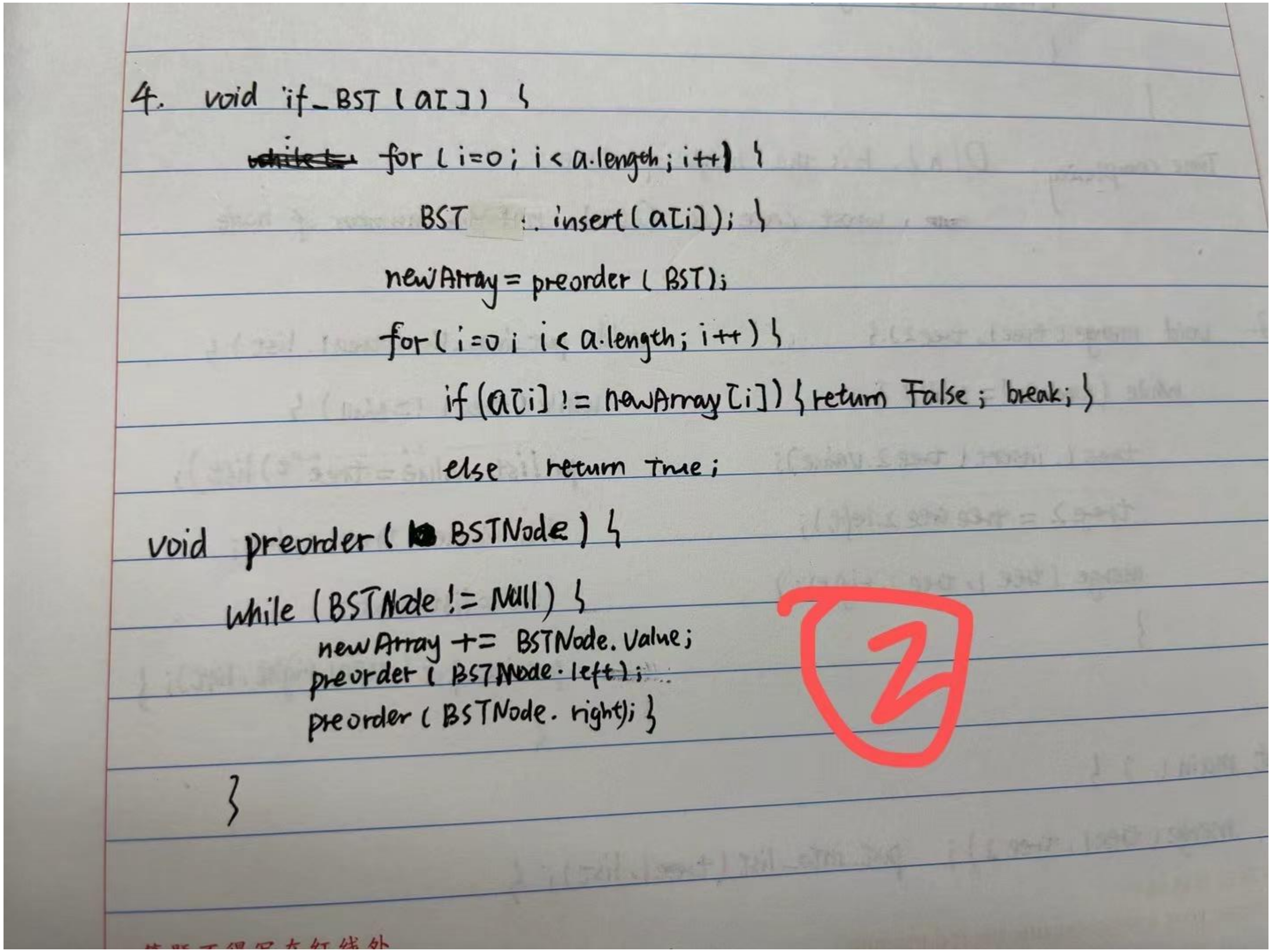
- Simple solution:
  - 1) Find the **first greater value** on right side of current node.
  - Let the index of this node be j. Return true if following
  - conditions hold. Else return false
  - (i) **All values after** the above found greater value are
  - than current node.
  - (ii) Re**greater**cursive calls for the subarrays pre[i+1..j-1] and
  - pre[j+1..n-1] also return true.

https://www.geeksforgeeks.org/check-if-a-given-array-can-represent-preorder-traversal-of-binary-search-tree/

```
public class Main
{
    static int preIndex = 0;

    static void buildBST_helper(int n, int[] pre, int min, int max) //actually not building the tree
    {
        if (preIndex >= n)     return;
        if (min <= pre[preIndex] && pre[preIndex] <= max) {
            int rootData = pre[preIndex];
            preIndex++;
            buildBST_helper(n, pre, min, rootData); // build left subtree
            buildBST_helper(n, pre, rootData, max); // build right subtree
        }
    }
}
```

```
static boolean canRepresentBST(int[] arr, int N)
  {
      // code here
      int min = Integer.MIN_VALUE, max = Integer.MAX_VALUE;

      buildBST_helper(N, arr, min, max);

      return preIndex == N;
  }
```

An example ?



```
4. void if_BST (a[]) {
      while for (i=0; i<a.length; i++) {
            BST   . insert(a[i]); }
         newArray = preorder (BST);
         for (i=0; i<a.length; i++) {
               if (a[i] != newArray[i]) {return False; break;}
               else  return True;

   void preorder ( BSTNode) {
      while (BSTNode != Null) {
            newArray += BSTNode. Value;
            preorder ( BSTNode. left);
            preorder ( BSTNode. right); }

      }
```

2

An example ?

4. void if_BST (a[]) {

~~while~~ for (i=0; i<a.length; i++) {

BST . insert(a[i]); }

newArray = preorder (BST);

for (i=0; i<a.length; i++) {

if (a[i] != newArray[i]) {return False; break;}

else return True;

void preorder (BSTNode) {

if ~~while~~ (BSTNode != Null) {

newArray += BSTNode.Value;

preorder (BSTNode.left);

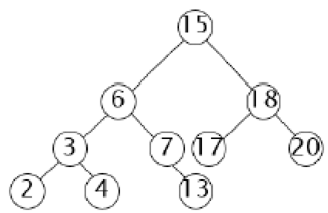
preorder (BSTNode.right); }

}

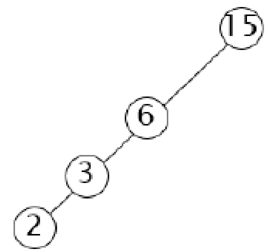
#6 Binary Search tree

6. [25 marks] Binary Search Tree (BST)

(1) [5 marks] explain the basic properties of BST and check if the following two trees are legal BSTs.



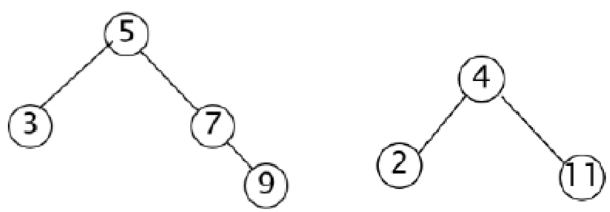
Tree 1



Tree 2

(2) [10 marks] write pseudocodes or codes to determine whether a given binary tree is a BST or not, and analyze the time complexity (e.g. using O).

(3). [10 marks] merge two binary search trees into a sorted linked list. For example, with two binary trees:



We could get the following linked list

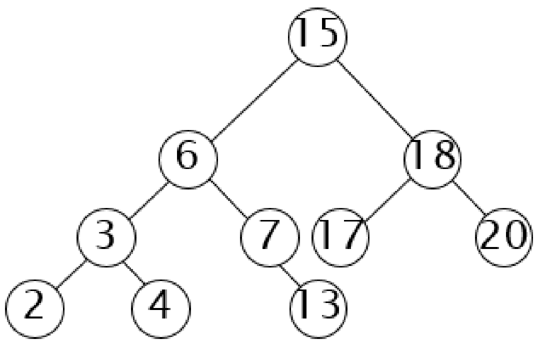
Please write pseudocodes or codes and analyze the time complexity (e.g. using O).

Binary Search Tree

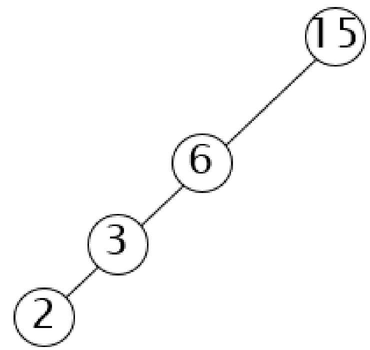
- BST has two properties:

- PRO 1: Binary tree property (same as for BST)
    - Each node has at most 2 children
- PRO 2: Order property (same as for BST)
    - For each node T
        - the key values in its left subtree are *smaller* than the key value of T
        - the key values in its right subtree are *larger than* the key value of T

Binary Search Tree



yes



yes

Binary Search Tree

```
boolean isBST(BSTNode root) {
    if (root == NULL)
        return True
    if (root.left and root.left.val > root.val)
        return False
    if (root.right and root.right.val < root.val)
        return False

    return isBST(root.left) and isBST(root.right))
```

O(n)

An example answer

Tree 1 is legal , Tree 2 is legal

```
(2) Analyze ( Node node)   // First input root.
if ( node == None)  return true.
else
    if ( ! ( (node.left.value < node.value) and (node.right.value > node.value)) return false.
    else  return ((Analyze (node.left) ) and ( Analyze (node.right)))
```

This algorithm checks all nodes (iterate all nodes)
until it encounter a false situation or finish checking
∴ Time complexity is $O(n)$. (n is the ~~number~~ amount of nodes)

(3) Inorder (~~Node~~ Node node, List list)

Merge two binary search trees into a sorted linked list

- A = inorder(bst1) $O(n)$
- B = inorder(bst2) $O(m)$
- C = merge(A,B) $O(m+n)$

```
void inorder(BSTNode root, List<Integer> array) {
    if (node == null)
        return;
    inorder(root.left(), array);
    array.add(root.value);
    inorder(root.right(), array);
}
void inorder(BSTNode root){
     List<Integer> array= new ArrayList<Integer>();
    inorder(root, array)
    return array
}
```

Merge Two Sorted Arrays

```
public static Node merge(int[] arr1, int[] arr2, int n1, int n2) {
    int i = 0, j = 0, k = 0;
    Node head = new Node();
    Node last = head;
    while (i < n1 && j < n2) {
        if (arr1[i] < arr2[j])
            last.next = new Node(arr1[i++]);
        else
            last.next = new Node(arr2[i++]);
        last = last.next;
    }
    while (i < n1)
        last.next = new Node(arr1[i++]); last = last.next;
    while (j < n2)
        last.next = new Node(arr2[i++]); last = last.next;
    return head;
}
```

Other solution

- seq1 = Inorder(bst1)
- For number in seq1:
-             bst2.insert(number)
- seq = Inorder(bst2)   (linked list)

- Thanks