1. a) Code 1: Assume initially $n > 1$ and eventually the loop takes p steps to terminate.

Therefore, $\frac{n}{2^p} \le 1$ and $\frac{n}{2^{p-1}} > 1$

Now, $\frac{n}{2^p} \le 1 \Rightarrow n \le 2^p \Rightarrow p \ge \log_2 n$

So, $p = \Omega(\log n)$

Also, $\frac{n}{2^{p-1}} > 1 \Rightarrow 2^{p-1} < n \Rightarrow p < \log_2 n + 1$

So, $p = O(\log_2 n)$

Hence, time complexity, $p = T(n) = \Theta(\log n)$

Code 2: Assume outer loop takes p steps.

Then, $2^{p-1} \le n$ and $2^p > n$

Also, $T(n) = \sum_{i=0}^{p-1} 2^i$

$2^{p-1} \le n \Rightarrow p \le \log_2 n + 1$

$2^p > n \Rightarrow p > \log_2 n$

$$\text{Now, } T(n) = \sum_{i=0}^{p-1} 2^i$$

$$\leq \sum_{i=0}^{\log_2 n+1-1} 2^i$$

$$= 2^{\log_2 n+1}$$

$$= 2n \qquad \text{So, } T(n) = O(n)$$

$$T(n) > \sum_{i=0}^{\log_2 n-1} 2^i$$

$$= 2^{\log_2 n}$$

$$= n \qquad \text{So, } T(n) = \Omega(n)$$

$$\text{Hence, } T(n) = \Theta(n)$$

1 b) To solve using substitution method, we first need to deduce time complexity using master theorem or any other means. Then we prove the time complexity using proper substitution.

Master Theorem gives $T(n) = O(n^2)$ here.

We will prove $T(n) \le cn^2 - dn$.

Since base case is not given, we will only prove the inductive step.

Inductive hypothesis: $T(k) \le ck^2 - dk$ for all $1 \le k < n$

Now, $T(n) = 4T\left(\frac{n}{2}\right) + 100n$

$$\le 4\left(c.\frac{n^2}{4} - d.\frac{n}{2}\right) + 100n$$

$$= cn^2 - (2d - 100)n$$

$$\le cn^2 - dn$$

CamScanner

choosing c large enough to avoid boundary conditions and $d \geq 100$ makes the argument above work.

1 c)

i) False. $2n^2$ steps can also be taken.

ii) False. Although $O(n)$ is asymptotically slower, for smaller input, sometimes it can work faster. For example, if we compare two algorithms with n steps and $100 \log_2 n$ steps, for $n = 4$, algorithm with n steps is much faster

iii) True. $c_1 \log_2 n = O(\log_2 n)$

$$c_2 \log_4 n = c_2 \log_4 2 \times \log_2 n$$

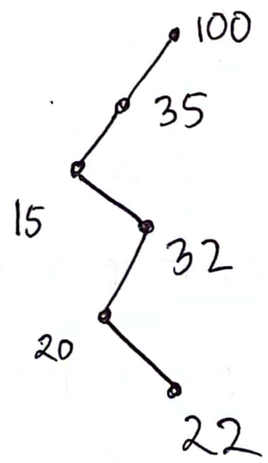
$$= \frac{c_2}{2} \log_2 n = O(\log_2 n)$$

iv) False. Argument similar to ii can be shown.

2.a) Sequence 1: No.

The i-th element $(i \geq 5)$ of the sequence should be less than 30, because after querying 30, we enter the left subtree of 30. But 6th element (35) is > 30.

b) Sequence 2: Yes



Sequence 3: No

After facing 50, search should proceed to left subtree to find 22 or stop if the left subtree is empty. So elements greater than 50 should not be present in the sequence.

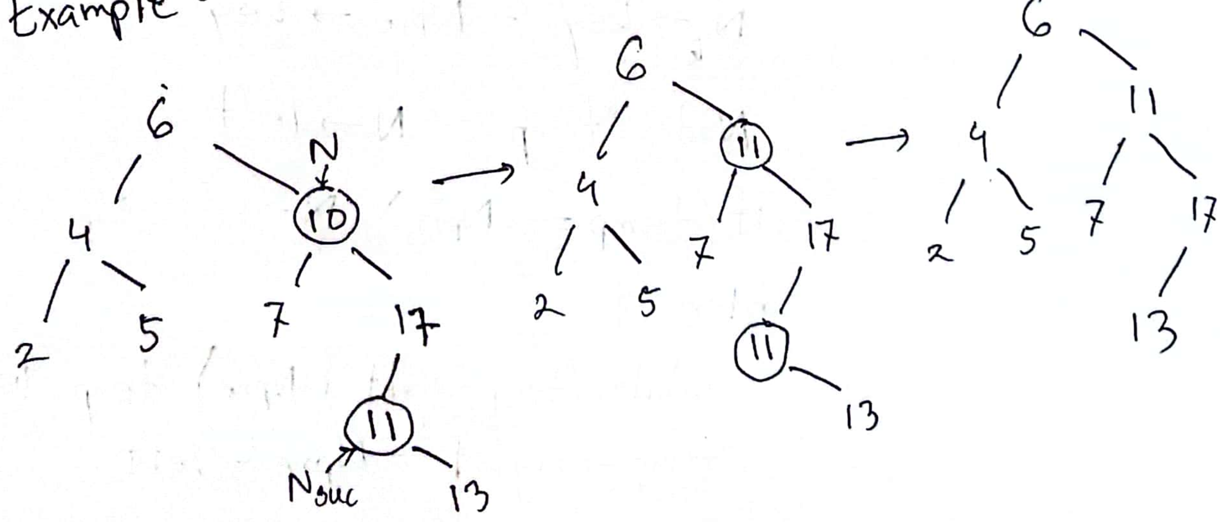
2.b) i) Nodes can be traversed in ascending order through inorder traversal in BST.

```
void traverseAndStore(Node *root) {
        static Node* prev = NULL
        if(root == NULL) return
        traverseAndStore(root->left)
        if(prev != NULL) prev->suc = root->key
        root->pre = prev->key
        prev = root
        traverseAndStore(root->right)
}.
```

ii) Assuming the key in the nodes are distinct.

```
void removeFullNode(N, Nsuc) {
    N → key = Nsuc → key
    Node * temp = N → right
    if (temp == Nsuc) N → right = Nsuc → right
    else {
        while (temp → left != Nsuc) temp = temp → left
        temp → left = Nsuc → right
    }
    delete Nsuc
}
```

Example -



Time complexity depends on the steps taken in while loop in line 6 ; If BST is balanced, this may take O(log n) steps. Otherwise, it can take upto O(n) steps

iii) If the nodes have distinct keys, then the strategy suggested would work. Otherwise, left subtree can have equal keys, which is unwanted.

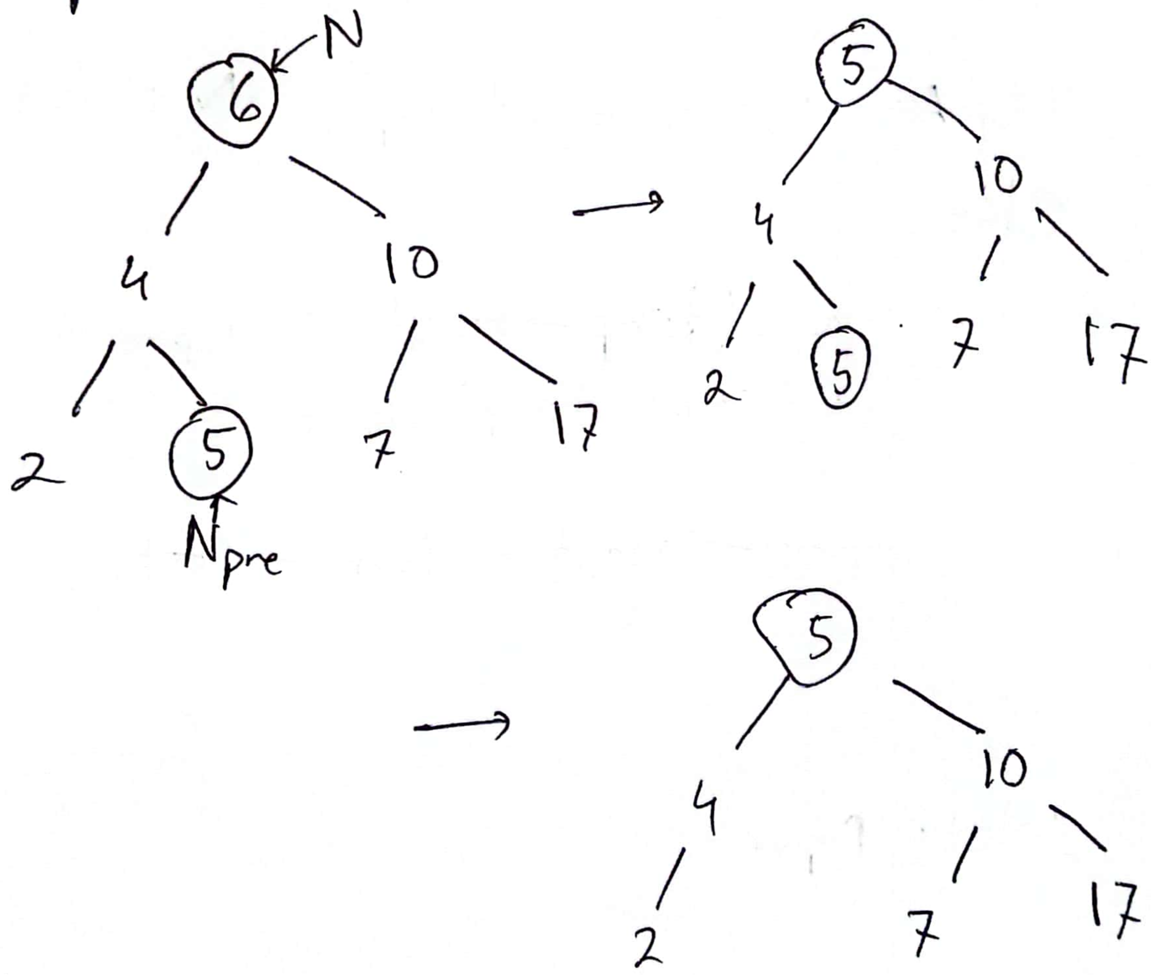
~~void removeFullNode (Node *root) {~~

```
void removeFullNode(N, Npre) {
    N→key = Npre → key
    Node *temp = N → left
    if(temp == Npre) N → left = Npre → left
    else {
        while (temp→right != Npre) temp = temp→right
        temp → right = Npre → left
    }
```

delete Npre

}

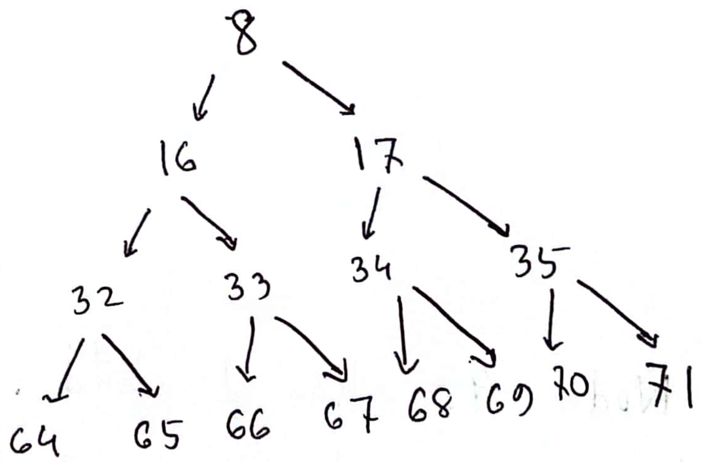
Example -



3.a) i) Node 44's children should have key $2*44 = 88$ and $2*44+1 = 89$. But there are only 86 keys. So it's a leaf.

ii) Node 43's children = 86, 87. But again, there are only 86 keys. So, it has 1 child~~ren~~.

iii)



Since, every node if full, it's a full binary tree.

iv) From the picture of (iii), there are 3 levels.

v)

1
↙
2
↙
4
↙
8
↙
16
↙
32
↙
64

Nodes ~~is~~ shown in left are the leftmost node of each level. Since node 128 is absent, there are only 6 levels

vi) Node 43 has 1 child. So, all the nodes with key greater than 43 has no child.

Number of leaf nodes $= 86-43$

$= 43$

3.b) Cormen 6.3

4 a)

```
class Stack {
    List L;
    void clear() {
        while(L.length != 0) {
            L.moveToEnd();
            L.remove();
        }
    }
    void push(int item) {
        L.append(item);
    }
    int pop() {
        L.moveToEnd();
        ~~L.remove()~~
        return L.remove();
    }
    int length() {
        return L.length();
    }
```

```
int max() {
    L.moveToStart(); int Max = -∞
    for (int i = 0; i < L.length(); ++i) {
        if (max < L.getValue()) {
            Max = L.getValue();
        }
        L.next();
    }
    return Max;
}
```

4 b)
```
int max(Stack &S) {
    if (S.length() == 0) return -∞;
    int u = S.pop();
    int w = max(S) // recursive call
    S.push(u);
    if (u > w) return u;
    else return w;
}
```

```
void addBook(dllNode* head, string title, string publisher)
{

    dllNode* p = head;
    dllNode* q = new dllNode(title, publisher, NULL, NULL);

    while(p->next!= NULL && p->next->publisher != publisher) {
        p = p ->next;
    }

    q->prev = p;
    q->next = p->next;
    if(p->next != NULL){
        p->next->prev = q;
        p->next = q;
    }

}
```