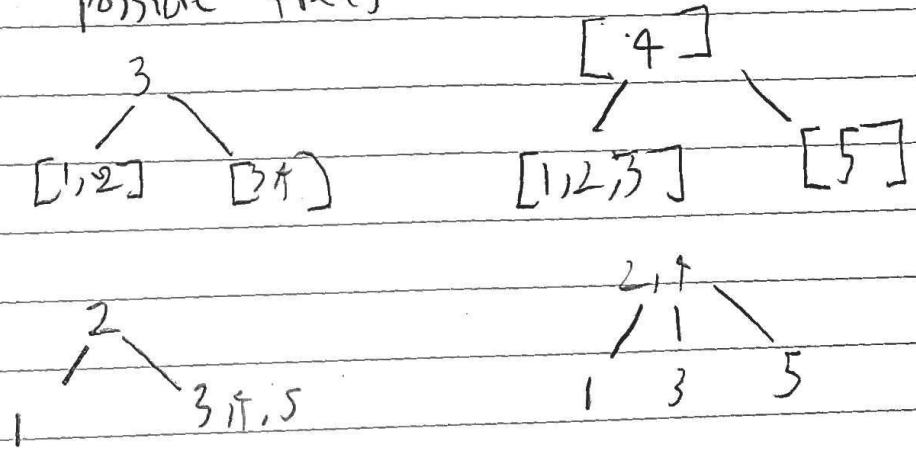
313 Assignment

1) All legal trees of degree 2 [1,2,3,4,5]

$M-1=4$, $Max\ 2M-1=3$

4 possible trees



2) a) To find Minimum node start from root then traverse to leftmost child / leaf, take leftmost element of the node

Take a node of the tree as a parameter, if value is NULL then return NIL

If left and right child is NULL (so its a leaf node) then return the minimum tree

Else call B-Tree recursively

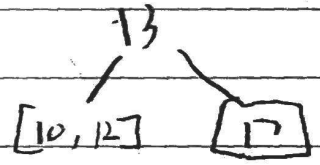
b) To find the Predecessor find if the node is a leaf or not if it's not then return largest element of child that Precedes the key which has been found, if its a leaf then return Predecessor of given node

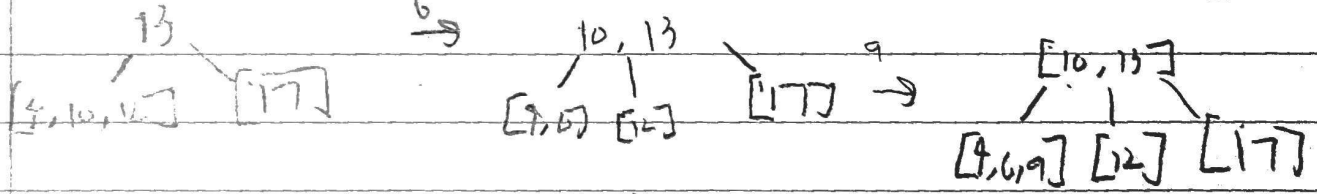
Case 1: If node is not leaf then return Max of left subtree

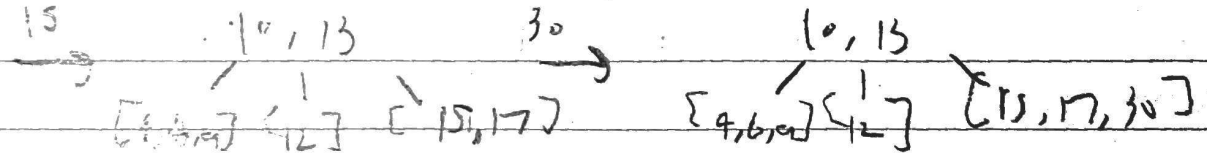
Case 2: If the node is the leaf and i>1 then return i-1th key or null

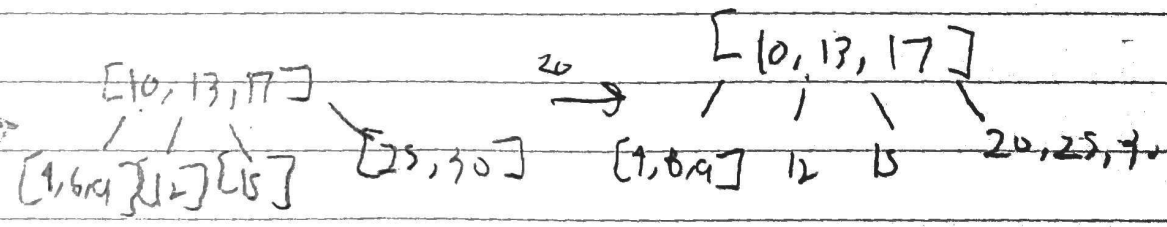
case 3: If the node has no left child, then find predecessor of node by traversing bottom to top until i>0

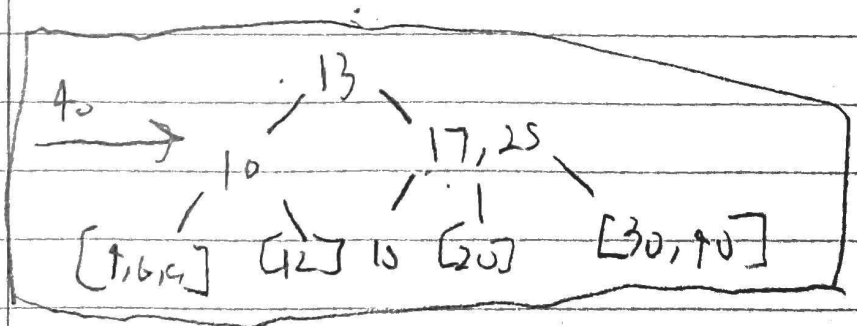
Use a Temp variable to store prev. & then find predecessor

3) [12, 13] → [12, 13, 17] 17 → 









4) We will want to use either Heap-Sort or Merge-Sort which is $O(\lg n)$ then we count the most appears search which will be $O(n)$ time.

```
Election (S, n)
    Sort (S, S+n)
    Most = S[0]
    Current = 1
    Current most = 1
for i=1 to n-1
    if S[i] == S[i-1]
        {
            Counter += 1
        }
    else {
        Counter = 1
if Counter > Current most
        Current most = Counter
        Most = S[i]
        }
}
return Most
```

5) 1. Store an array A with size K in sorted order, the A[x] with most votes is winner.

2. for each vote done do a binary-search on Array A to find candidate with the particular voter ID and increment by 1

3. once vote record is over, declare candidate with highest vote count in array A.

$O(k \log k)$ to sort the list,

Binary search will take $O(\log k)$ time

$O(k \log k) + O(n \log k) = O(n \log k)$

```
Election (S, k)
    Array = k = candidates
    Array S = Votes
    for i in len(S)
        BinarySearch into k
            Count += 1
    for i in k
        if Count > MaxCount
            Candidate = Array[i]
    return candidate
```

6) O(n) for $\left(\left\lceil \frac{n+1}{2} \right\rceil\right)$

Majority element

keep count of every students in a map and see if any candidate exceed it.

```
Majority (A) :
    Maj_index = 0
    count = 1
    for i in len(A)
        if A[Maj_index] == A[i]:
            count += 1
        else:
            count -= 1
        if (count == 0):
            Maj_index = i
            count = 1
    count = 0
    for i in len(A)
        if A[i] == A[Maj_index]
            count += 1
    if count > len(A)/2:
        return True
    else
        return false
```