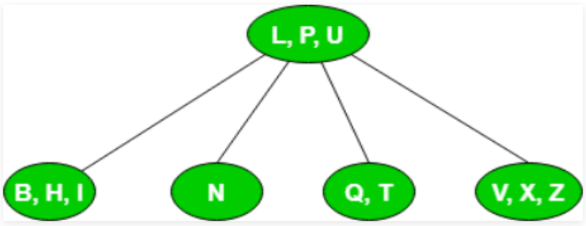
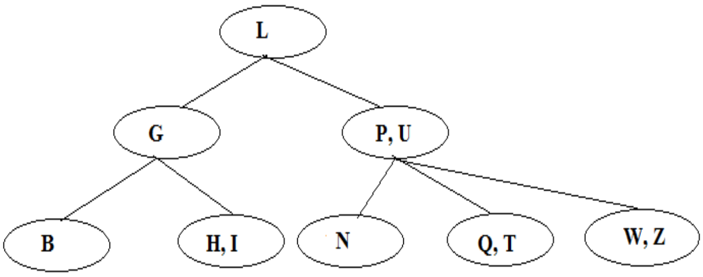
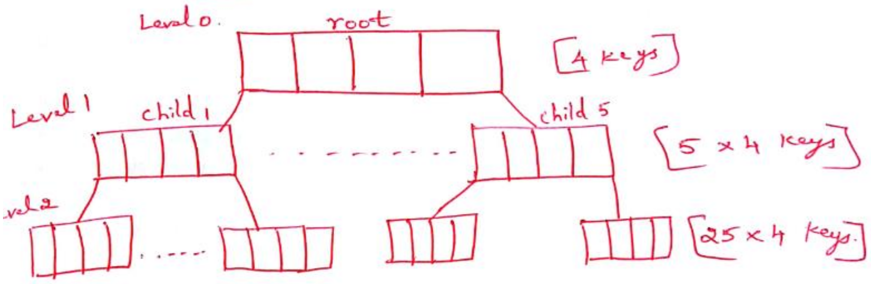
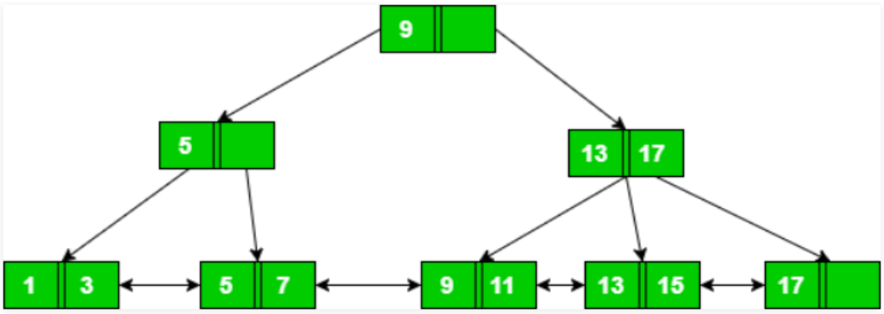
<table>
<tr><td colspan="5">ROLL NO:</td></tr>
<tr><td colspan="5">KARPAGAM COLLEGE OF ENGINEERING, COIMBATORE-641032</td></tr>
<tr><td colspan="3">B.E (CSE/ECE/EEE/EIE/ETE/MECH/AUTO)/B.TECH IT</td><td colspan="2">SEMESTER : III</td></tr>
<tr><td colspan="5">17PE01/17LE01/17EE01/17NE01/17TE01/17FE01 - ADVANCED DATA STRUCTURES</td></tr>
<tr><td colspan="5">CONTINUOUS INTERNAL ASSESSMENT: II ANSWER KEY</td></tr>
<tr><td colspan="3">DURATION : 3 HOURS</td><td colspan="2">DATE : 28.08.2018</td></tr>
<tr><td colspan="3">MAXIMUM : 100 MARKS</td><td colspan="2">SESSION : FN</td></tr>
<tr><td colspan="2">Answer All Questions</td><td colspan="3">PART- A (10 x 2 = 20 Marks)</td></tr>
<tr><td>1

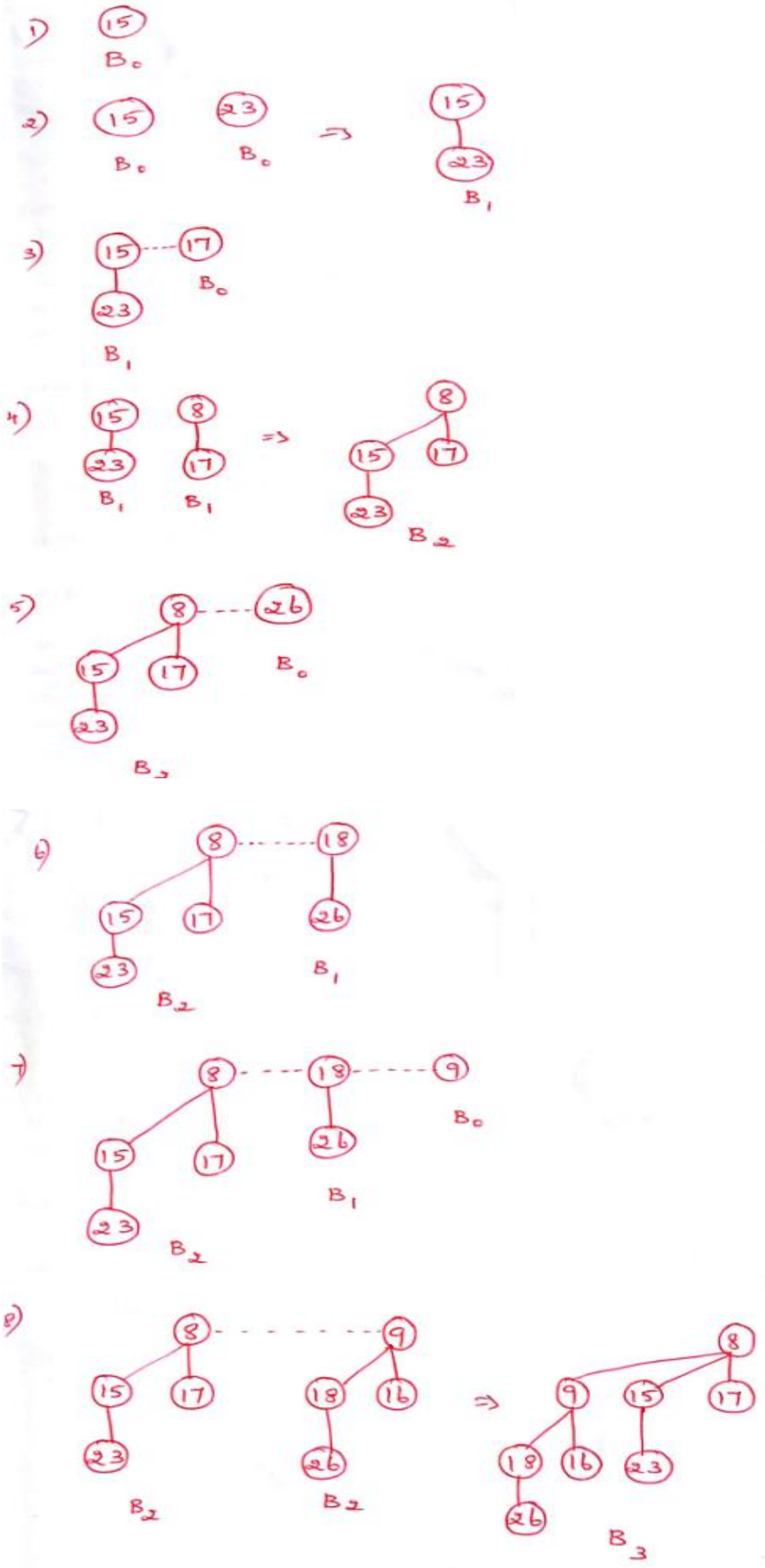
ANS</td><td>Fill in the blanks. The following code fragment is used to check whether two binary trees are identical or not.
Input: r1 and r2 are roots of two Binary Trees.
int isIdentical(Node *r1, Node *r2)
{
if(r1==NULL && ___________)
return 1;
if(_____________ || r2==NULL)
return 0;
if(r1->data==r2->data)
return (isIdentical(_____ , ______), isIdentical(_____,_____)
return 0;
}
int isIdentical(Node *r1, Node *r2)
{
if(r1==NULL && r2==NULL)
return 1;
if(r1==NULL || r2==NULL)
return 0;
if(r1->data==r2->data)
return (isIdentical(r1->left,r2->left), isIdentical(r1->right,r2->right)
return 0;
}</td><td>02</td><td>CO2</td><td>K1</td></tr>
<tr><td>2

ANS</td><td>Say true or false and justify.
a) The subtree of the root of a red-black tree is always itself a red-black tree.
b) The worst case time complexity of the insert operation into an AVL tree is not always O(logn), where n is the number of nodes in the tree.
a) False
b) False</td><td>02</td><td>CO2</td><td>K2</td></tr>
<tr><td>3</td><td>Consider the following 2-3-4 tree (i.e., B-tree with order 4) in which each data item is a letter. The usual alphabetical ordering of letters is used in constructing the tree. What is the result of inserting G into the following tree?

 </td><td>02</td><td>CO3</td><td>K4</td></tr>
</table>

]

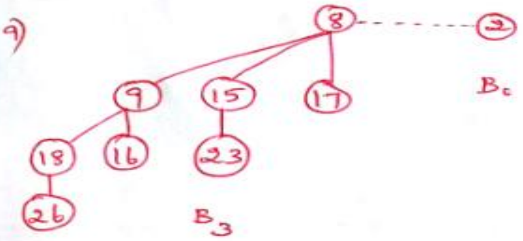
ANS				
4 ANS	State the differences and similarities between red black and AVL tree. DIFFERENCE 1. AVL trees provide faster lookups than Red Black Trees because they are more strictly balanced. 2. Red Black Trees provide faster insertion and removal operations than AVL trees as fewer rotations are done due to relatively relaxed balancing. 3. AVL trees store balance factors or heights for each node, thus requires O(N) extra space whereas Red Black Tree requires only 1 bit of information per node, thus require O(1) extra space. 4. Red Black Trees are used in most of the language libraries like map, multimap, multiset in C++ whereas AVL trees are used in databases where faster retrievals are required. SIMILARITIES Both are used for balancing Binary search tree using rotations.	02	CO2	K4
5 ANS	Which of the following statement is true about inorder successor needed in delete operation of a node with non-empty left and right child in a BST? Justify the answer with an example tree structure. a) Inorder Successor is always a leaf node. b) Inorder successor is always either a leaf node or a node with empty left child. c) Inorder successor may be an ancestor of the node. d) Inorder successor is always either a leaf node or a node with empty right child. b) Inorder successor is always either a leaf node or a node with empty left child.	02	CO2	K3
6 ANS	Find the maximum number of keys that can be stored in a B-tree of height two (the root plus two additional levels) with a minimum branching factor (order) of m=5? Justify the solution with its diagrammatic representation. B-Tree of order 5 with Height 2  Level 0. root [4 keys] Level 1 child 1 child 5 [5 × 4 keys] Level 2 [25 × 4 keys] Level 0 → 4 keys Level 1 → 20 keys Level 2 → 100 keys Total → 124 keys	02	CO2	K3

7 ANS	Write the properties of Binomial Trees. a) It has exactly 2^k nodes. b) It has depth as k. c) There are exactly ^kC_i nodes at depth i for i = 0, 1, . . . , k. d) The root has degree k and children of root are themselves Binomial Trees with order k-1, k-2,.. 0 from left to right.	02	CO3	K3
8 ANS	With reference to the B+ tree shown below, the minimum number of nodes (including the root node) that must be fetched in order to satisfy the following query: “Get all records with a search key greater than or equal to 7 and less than 15” is ______________. Show the sequence of node visit.  Output->5-7-9-11-13 Nodes visited->9-5-5-7-9-11-13-15	02	CO3	K4
9 ANS	Give the worst case running time for each of the following in terms of ‘n’. Assume that all keys are distinct. (a) Insert in an AVL tree of size n. (b) Insert in a Red Black tree of size n. (c) Search in a BST of size ‘n’. (c) DeleteMin in a binary min heap of size n. a) O(log n) b) O(log n) c) O(n) d) O(1)	02	CO3	K5
10 ANS	Government of India is developing a computer simulation of air-traffic control which handles events such as landings and take-offs. Each event has a time-stamp that denotes when the event occurs. The simulator needs to perform efficiently the following two operations: • Insert( ): a new event with a given time-stamp (i.e. add a future event). • Extract(): the event with the smallest time-stamp (i.e. determine the next event to process). Which data structure is suitable for implementing the above operations efficiently? Justify your answer. Insert and extract operations takes place based upon timestamp which is on priority basis. Hence, Heap is the data structure used to support those operations which can be binary heap or binomial heap.	02	CO3	K6
Answer All Questions PART- B (5 x 16 = 80 Marks)				
11)i)	Construct a Min Ordered Binomial Heap for the Data: {15, 23, 17, 8, 26, 18, 9, 16, 2, 28, 10, 7, 53, 21, 39 and 12}. Perform 4 consecutive DeleteMin( ) operations over the final Binomial Tree Constructed.	10	CO2	K3

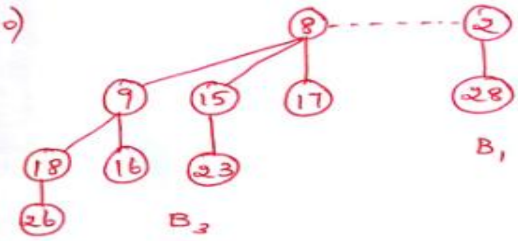
ANS				

1) 15 B0
2) 15 B0 23 B0 ⇒ 15 23 B1
3) 15 23 B1 --- 17 B0
4) 15 23 B1 8 17 B1 ⇒ 8 15 17 23 B2
5) 8 15 17 23 B2 --- 26 B0
6) 8 15 17 23 B2 --- 18 26 B1
7) 8 15 17 23 B2 --- 18 26 B1 --- 9 B0
8) 8 15 17 23 B2 --- 9 18 16 26 B2 ⇒ 8 9 15 17 18 16 23 26 B3

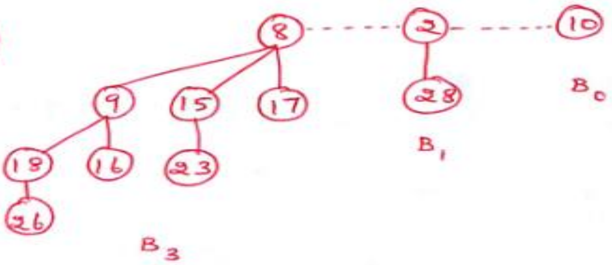
9)



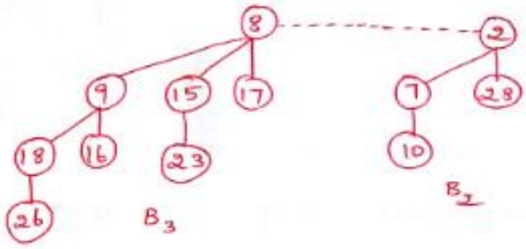
10)



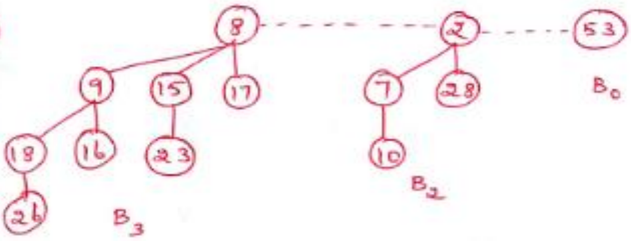
11)



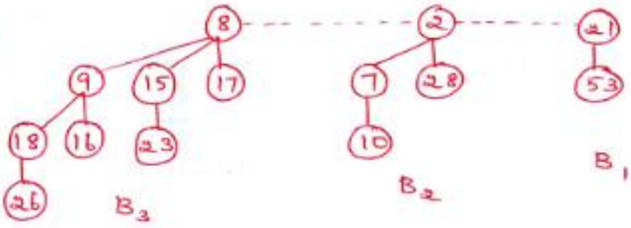
12)



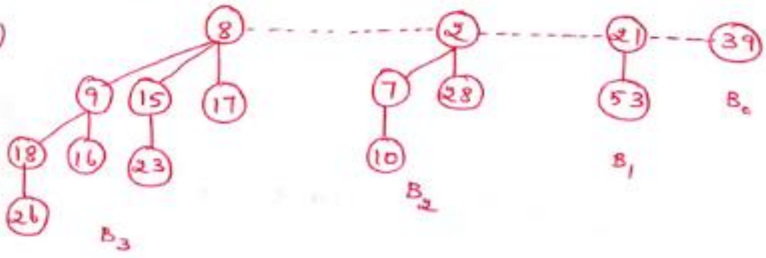
13)



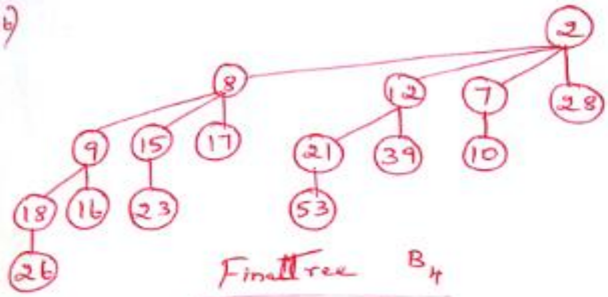
14)



15)



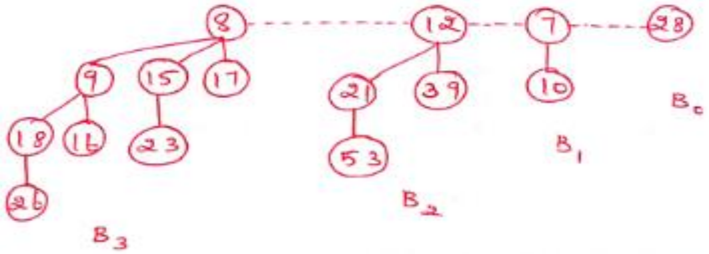
16)



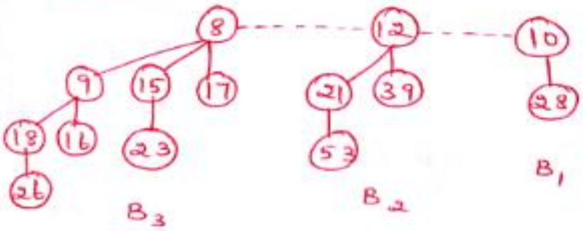
Final tree B4

4 Consecutive Deletions :-

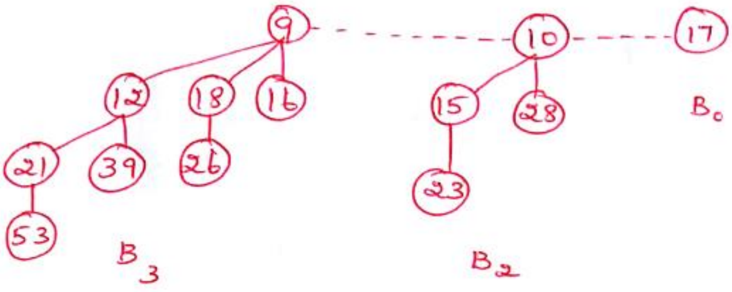
1) Delete (2)



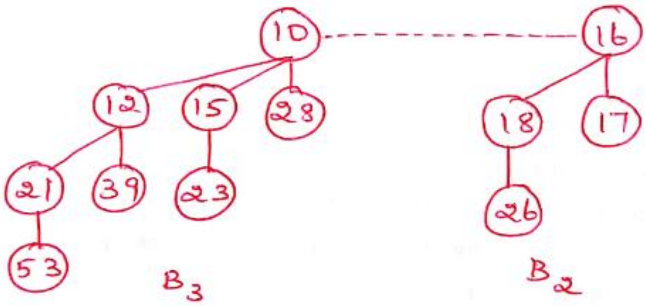
2) Delete (7)



3) Delete (8)



4) Delete (9)

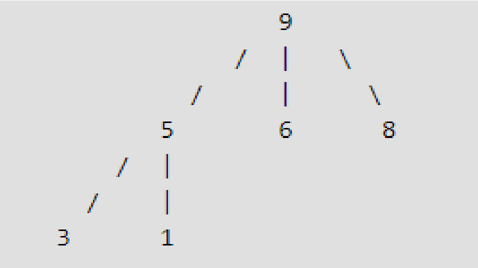
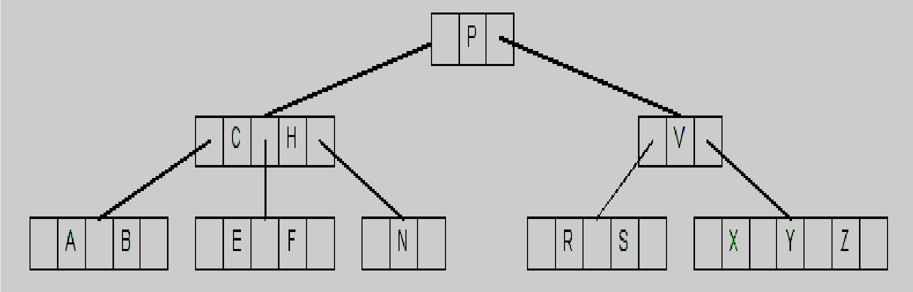


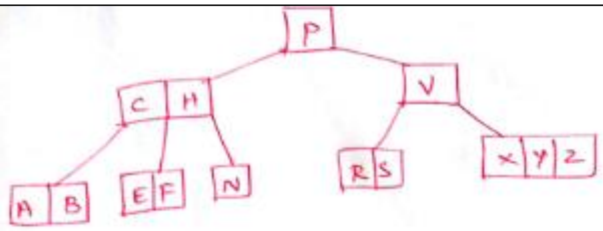
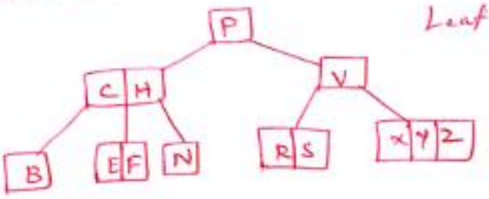
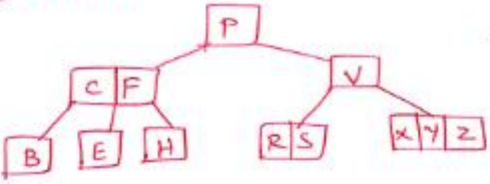
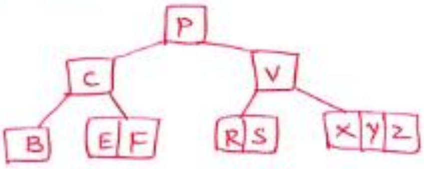
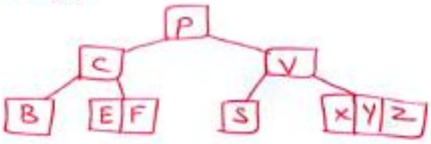
11)ii)	Complete the following functions.
	LeftRotation()
	{
	_____________
	}
	RightRotation()
	{
	_____________
	}
		06	CO2	K4

ANS	struct Node *leftRotate(struct Node *x)
	{
	struct Node *y = x->right;
	struct Node *T2 = y->left;
	y->left = x;
	x->right = T2;
	x->height = max(height(x->left), height(x->right))+1;
	y->height = max(height(y->left), height(y->right))+1;
	return y;
	}
	struct Node *rightRotate(struct Node *y)
	{
	struct Node *x = y->left;
	struct Node *T2 = x->right;
	x->right = y;
	y->left = T2;
	y->height = max(height(y->left), height(y->right))+1;
	x->height = max(height(x->left), height(x->right))+1;
	return x;
	}			

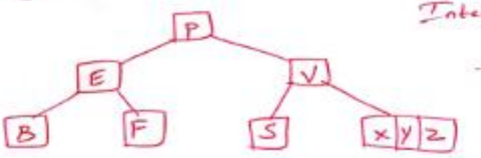
(OR)				
12)i)	Write an algorithm to check if the given array represents a Binary Max-Heap or not. The algorithm should print 1 if the array could represent the max heap else print 0. Input 1: A[] = {90, 15, 10, 7, 12, 2} Output 1: 1 Input 2: A[] = {9, 15, 10, 7, 12, 11} Output 2: 0	10	CO2	K3

ANS	#include <stdio.h>
	#include <limits.h>
	bool isHeap(int arr[], int i, int n)
	{
	  if (i > (n - 2)/2)
	      return true;
	  if (arr[i] >= arr[2*i + 1] && arr[i] >= arr[2*i + 2] &&
	      isHeap(arr, 2*i + 1, n) && isHeap(arr, 2*i + 2, n))
	      return true;
	  return false;
	}
	int main()
	{
	   int arr[] = {90, 15, 10, 7, 12, 2, 7, 3};
	   int n = sizeof(arr) / sizeof(int);
	   isHeap(arr, 0, n)? printf("Yes"): printf("No");
	   return 0;
	}			

12)ii)	A 3-ary max heap is like a binary max heap, but instead of 2 children, nodes have 3 children. A 3-ary heap can be represented by an array as follows: The root is stored in the first location, a[0], nodes in the next level, from left to right, is stored from a[1] to a[3]. The nodes from the second level of the tree from left to right are stored from a[4] location onward. An item x can be inserted into a 3-ary heap containing n items by placing x in the location a[n] and pushing it up the tree to satisfy the heap property. Which one of the following is a valid sequence of elements in an array representing 3-ary max heap? Justify your answer diagrammatically. (A) 1, 3, 5, 6, 8, 9 (B) 9, 6, 3, 1, 8, 5 (C) 9, 3, 6, 8, 5, 1 (D) 9, 5, 6, 8, 3, 1	04	CO2	K4
ANS	D) 9, 5, 6, 8, 3, 1 			
12)iii)	Consider a B+-tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node?	02	CO2	K4
ANS	Assuming order of B+ tree as p, maximum number of keys will be (p – 1). As it is given that, p – 1 = 5 => p = 6 Therefore, minimum number of keys: ceil(p/2) – 1 = 2			
13)i)	Consider the following 2-3-4 tree. Perform the deletion of nodes in the order {A, N, H, R, C, P, E, F, V, B, X, Y, S, Z} one after the other. Specify the type of operation used to overcome the violation. Clearly show the tree structure before and after each deletion.  P C H V A B E F N R S X Y Z	12	CO3	K2

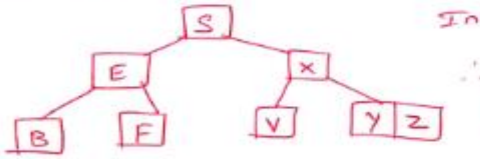
ANS	 1) Delete (A)  Leaf Node 'A' 2) Delete (N)  Leaf Node 'N' 3) Delete(H)  Leaf Node 'H' 4) Delete (R)  Leaf Node 'R'			

5) Delete (c)



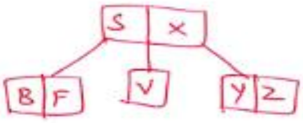
Internal Node 'c'
∴ swap 'c' & 'E'

6) Delete (P)



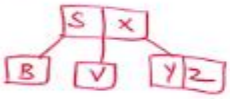
Internal Node 'P'.
∴ swap 'P' & 'S'.
Redistribute.

7) Delete (E)



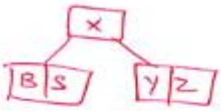
Internal Node 'E'
swap 'E' & 'F'
Merge.

8) Delete (F)

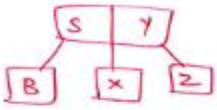


Leaf node 'F'

9) Delete (v)

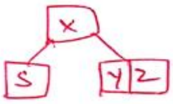


(or)

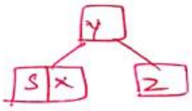


Merge. Leaf Node 'v' Redistribute.

10) Delete (B)



(or)

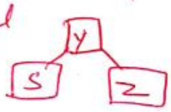


Merge.

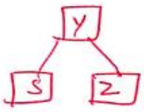
Leaf Node 'B'.

11) Delete (x)

Internal Node 'x'

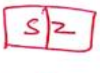


(or)



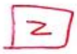
Leaf Node x.

12) Delete (y)



Internal Node 'y'
Merge.

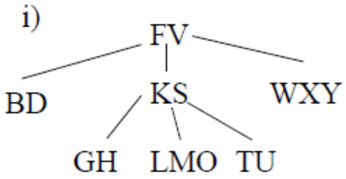
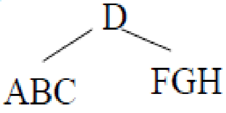
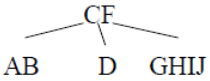
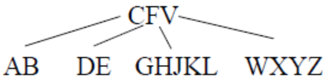
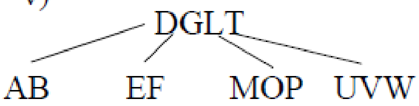
13) Delete (S)



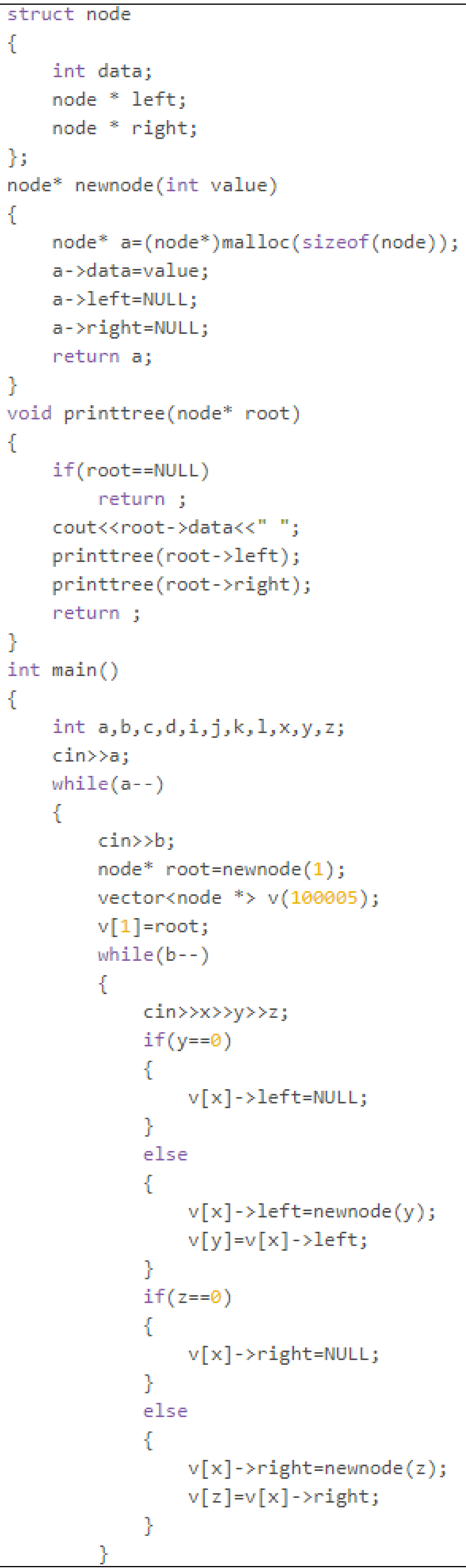
Leaf Node 'S'

14) Delete (z) Leaf Node 'z'.

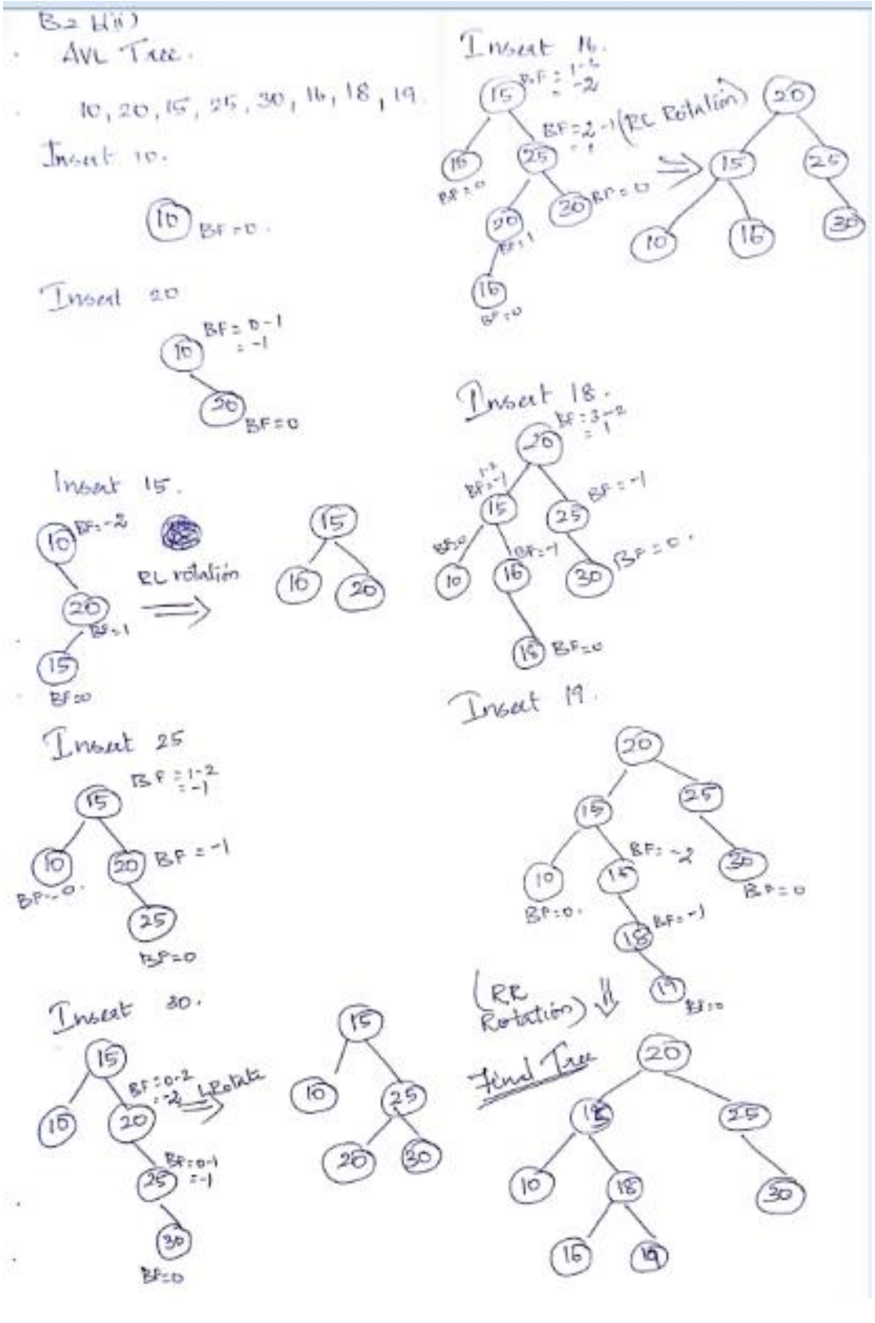
—

13)ii) ANS	Which of the following are legal B-trees when the order is 4? For those that are not legal, give one or two sentence very clearly explaining what property was violated. i)  ii)  iii)  iv)  v)  (i): Not legal since the height is not balanced. More specifically, both the node with "BD" and "KS" are at the same level but "BD" is a leaf and "KS" is not. (ii): This is legal. Remember, that the root can have just a single key. (iii): Not legal – the key "D" has less than the minimum allowable size of 2 keys. (iv): Not legal – GHJKL 5 keys in a node which is a violation. (v): Not legal – there's no leaf node corresponding to the keys between G and L.	04	CO3	K1
	(OR)			
14)i)	BlackRobber was a very intelligent Robber on Robber Island. In order to get eligible for marriage, he had to pass the Graduation Exam conducted by C.B.S.E (Central Board of Robber Education). Seeing the intelligence level of the BlackRobber, the head of C.B.S.E decided to conduct the exam himself. BlackRobber performed very well in all the levels of the exam but got stuck on the last one. At the final level of the exam, the Head of C.B.S.E sent him to one of the Black trees and BlackRobber's task was to report all the fruits on the tree in a specific way. Black tree was a very special tree which had fruits of various colors. Even the smallest Black trees have a fruit with color 1. As the tree grows larger, two more fruits with unique colors get attached to some existing fruit of the tree as Fruit left and Fruit right. BlackRobber has to travel the Black Tree in a very specific way and report the answer at the end. A fruit with color 1 is assumed to be present on every Black Tree. Black Sail starts with color 1 and notes it down. On reaching any fruit, he notes down its color and then moves to the left part of the tree. on completing the entire left sub-tree, he moves to the right subtree and does the same. Refer the sample test cases for more clarity. While he was going to submit the answer sheet, the sheet was blown away by wind. You need to help the BlackRobber generate all the answers without traveling the entire tree again. Design algorithm to display all the fruit colors on the Black Tree as described. Input Format: The first line of the input contains the number of test cases. The first line of each test case consists of a single integer n (the number of relations describing the tree). 'n' lines follow describing the n relations. Each relation has 3 space separated integers X, Y and Z. X is some existing fruit on the tree. Y and Z are the colors of the left fruit and right fruit of X	16	CO2	K5

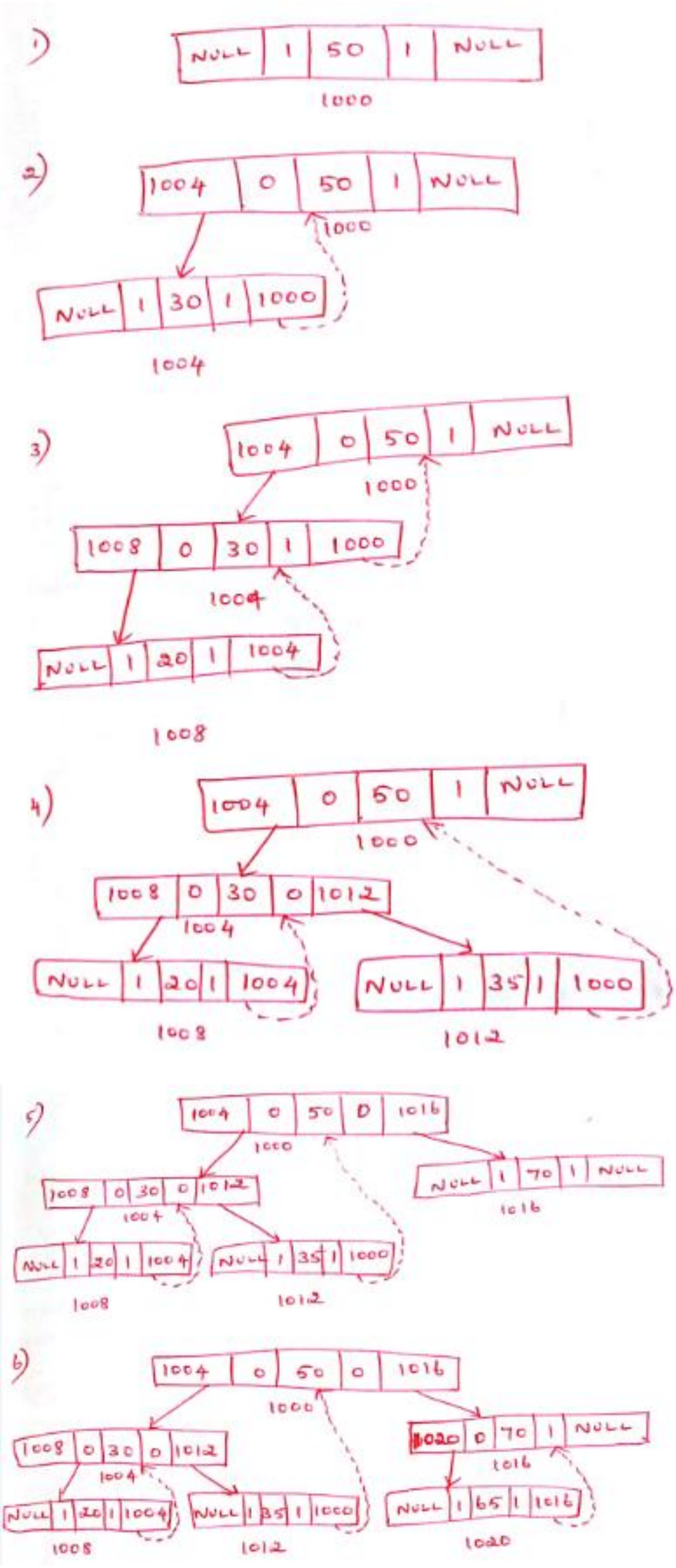
ANS	respectively. If Y or Z = 0, it implies that X has no left fruit or right fruit respectively. Output Format: You need to display all the fruit colors on the Black Tree in the described manner. Input 2 7 1 7 3 7 9 14 3 10 5 9 6 12 14 8 4 10 2 0 5 11 0 4 1 11 13 11 14 7 14 5 9 7 6 4 Output 1 7 9 6 12 14 8 4 3 10 2 5 11 1 11 14 5 9 7 6 4 13			



```
struct node
{
    int data;
    node * left;
    node * right;
};
node* newnode(int value)
{
    node* a=(node*)malloc(sizeof(node));
    a->data=value;
    a->left=NULL;
    a->right=NULL;
    return a;
}
void printtree(node* root)
{
    if(root==NULL)
        return ;
    cout<<root->data<<" ";
    printtree(root->left);
    printtree(root->right);
    return ;
}
int main()
{
    int a,b,c,d,i,j,k,l,x,y,z;
    cin>>a;
    while(a--)
    {
        cin>>b;
        node* root=newnode(1);
        vector<node *> v(100005);
        v[1]=root;
        while(b--)
        {
            cin>>x>>y>>z;
            if(y==0)
            {
                v[x]->left=NULL;
            }
            else
            {
                v[x]->left=newnode(y);
                v[y]=v[x]->left;
            }
            if(z==0)
            {
                v[x]->right=NULL;
            }
            else
            {
                v[x]->right=newnode(z);
                v[z]=v[x]->right;
            }
        }
```

15)i)
 	
ANS
	Insert the following sequence of elements into an AVL tree, starting with an empty tree: 10, 20, 15, 25, 30, 16, 18, and 19. Show the resultant tree and specify its height.
	
	 B.2 ii(i)
	AVL Tree.
	10, 20, 15, 25, 30, 16, 18, 19.
	Insert 10.
	BF=0.
	Insert 20
	BF=0-1 =-1
	BF=0
	Insert 15.
	BF=-2
	RL rotation
	BF=1
	BF=0
	Insert 25
	BF=1-2 =-1
	BF=-1
	BF=0.
	BF=0
	Insert 30.
	BF=0-2 =-2 L Rotate
	BF=0-1 =-1
	BF=0
	Insert 16.
	BF=1-3 =-2
	BF=2-1 (RL Rotation)
	BF=0
	BF=1
	BF=0
	Insert 18.
	BF=3-2 =1
	BF=-1
	BF=-1
	BF=0.
	Insert 19.
	BF=-2
	BF=0.
	BF=0
	BF=-1
	(RR Rotation)
	Final Tree
	Height of the tree is 3.
		06	CO2	K4

15)ii)
 	
ANS
	Insert the following sequence of data {50, 30, 20, 35, 70 and 65} one by one into an empty Right Threaded Binary Search Tree and show the linked list representation of each insertion step clearly and complete the following function used for inserting the data into the tree.
	TBST* Insert(TBST *tree, int element)
	{// Inserts data into TBT
	_____________
	_____________
	}
		10	CO3	K3



```
struct node *insert(struct node *tree, int element)
{
        struct node *newnode,*temp;
```

```
    if(tree==NULL)
    {
        tree=(struct node *)malloc(sizeof(struct node));
        tree->data=element;
        tree->left=tree->right=NULL;
        tree->lflag=tree->rflag=1;
    }
    else
    {   temp=tree;
        while(temp!=NULL)
        {
            if(element<temp->data)
            {
                if(temp->lflag==0)
                    temp=temp->left;
                else
                {
                  newnode=(struct node *) malloc (sizeof
                (struct node));
                  newnode->data=element;
                  newnode->left=newnode->right=NULL;
                  newnode->lflag=newnode->rflag=1;
                  newnode->right=temp;
                  temp->left=newnode;
                  temp->lflag=0;
                  break;
                }
            }
           else if(element>temp->data)
            {
                if(temp->rflag==0)
                    temp=temp->right;
                else
                {
                newnode=(struct node *)malloc(sizeof(struct
            node));
                newnode->data=element;
                newnode->left=newnode->right=NULL;
                newnode->lflag=newnode->rflag=1;
                newnode->right=temp->right;
                temp->right=newnode;
                temp->rflag=0;
                break;
                }
    }
    else
        printf("\nElement already exists\n");
    }
    return tree;
        }
}
```

(OR)

16)i)	Given a Binary Search Tree and a target sum, write a function that returns true if there is a pair with sum equals to target sum, otherwise return false. Input: First line consists of T test cases. First line of every test case consists of N and target, denoting the number of elements in bst and target sum. Second line consists of elements of BST. Output: Return True if target sum pair is found else False. Input: 2 7 10 1 2 3 4 5 6 7 7 33 15 10 20 8 12 16 25 Output: 1 1	08	CO2	K4

ANS

```
int countPairs(Node* root1, Node* root2, int x)
{
    if (root1 == NULL || root2 == NULL)
        return 0;
    stack<Node*> st1, st2;
    Node* top1, *top2;
    int count = 0;
    while (1) {
        while (root1 != NULL) {
            st1.push(root1);
            root1 = root1->left;
        }
        while (root2 != NULL) {
            st2.push(root2);
            root2 = root2->right;
        }
        if (st1.empty() || st2.empty())
            break;
        top1 = st1.top();
        top2 = st2.top();
        if ((top1->data + top2->data) == x) {
            count++;
            st1.pop();
            st2.pop();
            root1 = top1->right;
            root2 = top2->left;
        }
        else if ((top1->data + top2->data) < x) {
            st1.pop();
            root1 = top1->right;
        }
        else {
            st2.pop();
            root2 = top2->left;
        }
    }
    return count;
}
```

16)ii)	Joker is at a Code Expo where many coders are present discussing, solving, sharing, and eavesdropping on solutions. He recently learnt that a former HackerPlant employee, Batman, who is now working at KodeKarma stole some questions for their KoolKontest. Joker wants to find Batman, but the only data he has on him, and everyone else present, are their CodeXP ratings, which are distinct. Joker decides to find Batman through his rating by asking different people present at the expo. Everyone present are arranged in such a way that, assuming a person with rating X, every person with a rating higher than X are on the person's right, while every person with a rating lower than X are on the person's left. Everyone present there knows Batman's rating, except Joker, who now asks people where he could find Batman. Joker initially asks an arbitrary person, who replies by giving information on whether Batman is to the person's left or the person's right, depending on whether this person has a higher or lower rating than Batman. Joker proceeds in that direction for an arbitrary distance and stops and asks the next person encountered, and repeats this procedure. However, Joker will never go beyond a person whose rating Joker has asked before. For example, if Joker was walking to the left and finds a person who already told him to walk to the right then he will not continue going to the person's left. Joker finally finds Batman when he finally stops in front of Batman. During Joker's search for Batman, he wrote the sequence of ratings of the people Joker asked in the exact same order, including Batman's rating, which is the last integer in the sequence. Towards the end, Joker feels he might've made a mistake while writing this sequence. Given the sequence that Joker wrote and Batman's rating, write pseudocode to find out whether this sequence is possible or has mistakes. INPUT: First line contains number of test cases T. The first line of each test case contains the number of people Chef meets, N and Reziba's rating R separated by a space. Second line of each test case contains N number of space separated ratings from A_1 to A_N where A_N is always equal to R. OUTPUT: For each test case, output a single line with either "YES" if the sequence is correct or print "NO" if the sequence has mistakes, without quotes. Input: 2 5 200 600 300 100 350 200 5 890 5123 3300 783 1111 890 Output: NO YES	08	CO3	K3

ANS

```
for(i=0;i<n-1;i++)
{
    if(a[i]>min)
     {
        if(a[i]<a[n-1])
        {
          min=a[i];
          k++;
         }
```

```
      }
       if(a[i]<max)
       {
         if(a[i]>a[n-1])
         {
           max=a[i];
           k++;
           }
         }
}
if(k==n-1)
      printf("YES\n");
else
      printf("NO\n");
```

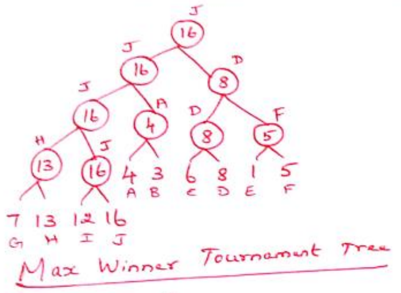
17)i) Construct the following Tournament Trees for the given data that represents the scores of various players in the world cup series. Find the first and second Best Player in the entire tournament. — 12 — CO2 — K4

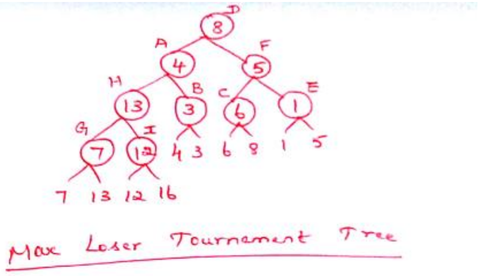
i) Max Winner Tree and Max Loser Tree
ii) Min Winner Tree and Min Loser Tree
iii) Find the first and second Best Player in the entire tournament.
iv) Write an algorithm to construct any one of the trees given.

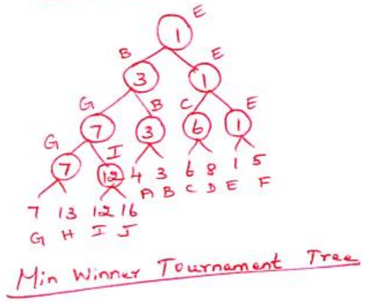
A	B	C	D	E	F	G	H	I	J
4	3	6	8	1	5	7	13	12	16

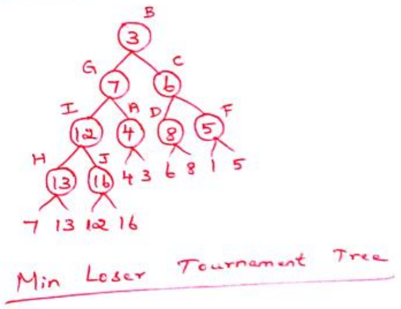
ANS

i), ii)





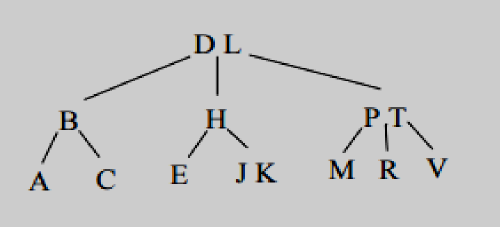
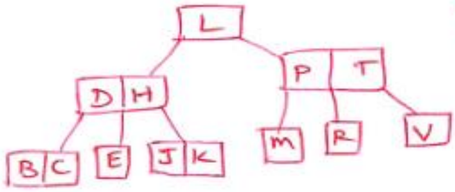
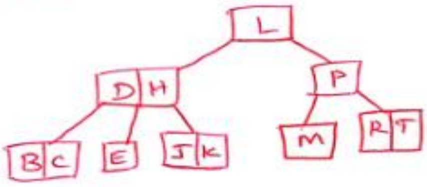
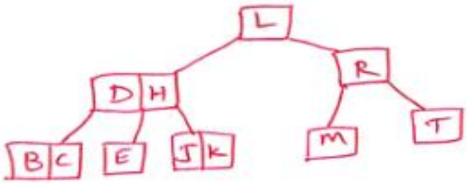




iii)
Max Winner/Loser Tournament Tree:
First Best Player : J with score 16
Second Best Player : D with score 8
Min Winner/Loser Tournament Tree:
First Best Player : E with score 1
Second Best Player : B with score 3
iv)

```
int minn(int a, int b)
{         return (a<b)?a:b;    }
```

```
main()
{         int n,size,i,j,x;
          printf("\nEnter the no of elements:");
          scanf("%d",&n);
          size=2*n-1;
          int minwinner[size];
          printf("\nEnter the elements:");
          for(i=n;i<=size;i++)
          {         scanf("%d",&minwinner[i]);  }
          for(i=size;i>1;i=i-2)
          {    minwinner[i/2]=minn(minwinner[i],minwinner[i-1]);   }
          printf("\n The Min winner tree is :\n");
          for(i=1;i<=size;i++)
               printf("%d\t",minwinner[i]);
}
```

17ii)	Delete the element A,V and P in the following B-Tree with the order 3. Specify the type of operation used after deleting each and every node.  	04	CO3	K3
ANS	1) Delete (A)  Leaf Node 'A' Merge. 2) Delete (v)  Leaf Node 'v' Merge 3) Delete (P)  Internal Node 'P' swap 'P' & 'R'			
(OR)				
18)i)	Little Monk also met who he thinks is the new God of Indian cricket: Virat Kohli. Now Monk is extremely fond of Kohli -- not just as a T20 player, but a player in all formats. He loves the statistics involved in Kohli's career. Little Monk knows that Kohli has played N matches in all three	12	CO3	K1

formats of his career, ODI, T20 and Test Cricket. He wants to show-off his knowledge about Kohli's career so he decides to answer Q questions asked by Kohli. Kohli gives the Monk three different arrays with N numbers each denoting the runs in the i^{th} ODI match, i^{th} Test Match and i^{th} T-20 match respectively.

The value of Kohli's i^{th} match would be the sum of his score in the i^{th} T-20 match, i^{th} Test match and i^{th} ODI match. Kohli knows that Monk is extremely quick at finding the k th smallest value of all his innings, so he twists his query a bit. He asks the Monk to delete the k^{th} smallest value once that is answered by the Monk. If Kohli comes up with a number k which is greater than the number of matches remaining in Kohli's career, the Monk should say that the answer is 1. So much complication confuses the Little Monk and he asks for your help!

1) Identify the type of data structure used for the implementation of this problem.
2) Write an algorithm and give its implementation to help monk.

Input format: The first line contains an integer N, which denotes the number of matches played by Virat Kohli. The next three lines contain N integers each denoting the number of runs scored in ODI, T20 and Test respectively. The next line contains an integer Q, denoting the number of questions Virat is going to ask. The next Q lines contain an integer K, denoting the K^{th} smallest value which the Monk has to answer.

Output format: Print answer to each query in a new line. In case of an invalid query, print -1. Note: The test data contains a lot of I/O operations, it is thus recommended to use faster input / output mechanisms.

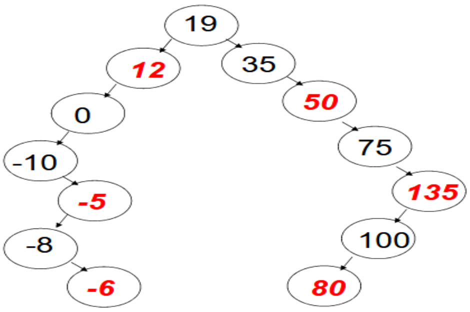
SAMPLE INPUT
```
5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
3
1
3
11
```
SAMPLE OUTPUT
```
3
12
-1
```
Explanation: The N values are: 3, 6, 9, 12, and 15. So, the first smallest is 3, and now the new value array is: 6, 9, 12, 15, so the third smallest value now is 12. The third query is invalid because the number of values array contains is only 3, and query is asking the Monk to find the 11th smallest number, so the answer is -1.

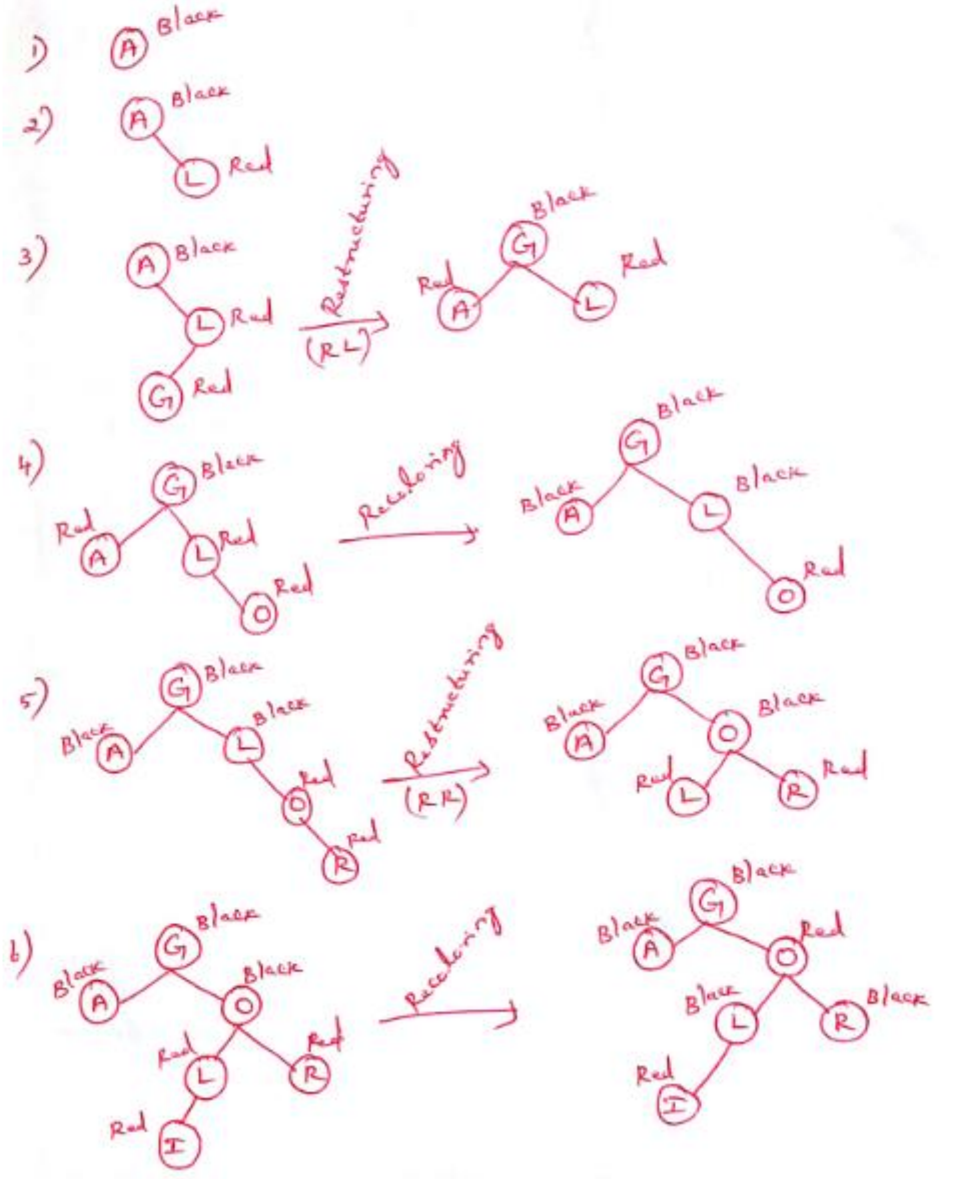
ANS
```
typedef long long int ll;
ll max[1000001];
ll min[1000001];
ll arr[1000001];
int size1=0;
int size2=0;
void swap_max(int i,int j)
{
 ll temp=max[i];
 max[i]=max[j];
```

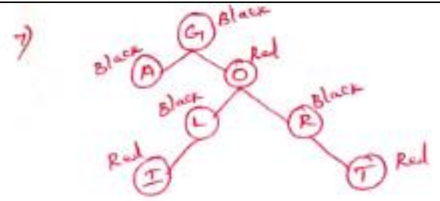
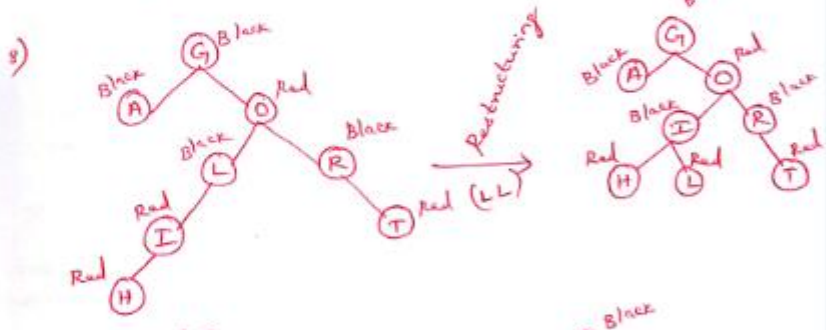
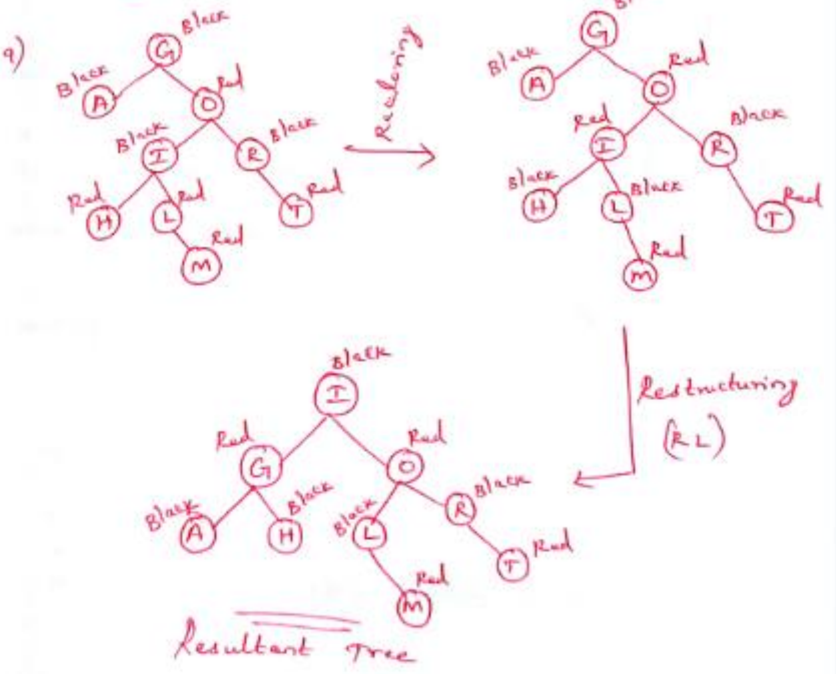
```
  max[j]=temp;
}
void swap_min(int i,int j)
{
  ll temp=min[i];
  min[i]=min[j];
  min[j]=temp;
}
void push_max(ll data)
{
  size1=size1+1;
  max[size1]=data;
  int i=size1;
  while(i>1 && (max[i/2] < max[i]))
  {
        swap_max(i/2,i);
        i=i/2;
  }
}
void push_min(ll data)
{
  size2=size2+1;
  min[size2]=data;
  int i=size2;
  while(i>1 && (min[i/2] > min[i]))
  {
        swap_min(i/2,i);
        i=i/2;
  }
}
void min_heapify(int index)
{
  int smallest;
  int left=2*index;
  int right=2*index+1;

  if(left<=size2 && (min[left] < min[index]))
   smallest=left;
  else
   smallest=index;

  if(right<=size2 && (min[right] < min[smallest]))
   smallest=right;

  if(smallest != index)
  {
        swap_min(smallest,index);
        min_heapify(smallest);
  }
}
void max_heapify(int index)
{
  int largest;
  int left=2*index;
  int right=2*index+1;
  if(left<=size1 && (max[left] > max[index]))
   largest=left;
  else
   largest=index;
  if(right<=size1 && (max[right] > max[largest]))
   largest=right;
  if(largest != index)
  {
          swap_max(largest,index);
          max_heapify(largest);
  }
}
ll extract_max()
{
  ll maxi=max[1];
  swap_max(1,size1);
  size1=size1-1;
  max_heapify(1);
  return maxi;
}
ll extract_min()
{
  ll mini=min[1];
  swap_min(1,size2);
  size2=size2-1;
  min_heapify(1);
  return mini;
}
void print_array()
{
  int i;
  for(i=1;i<=size1;i++)
   printf("%lld ",max[i]);
  printf("\n\n");
  for(i=1;i<=size2;i++)
   printf("%lld ",min[i]);
  printf("\n");
}
int main()
{
  int n;
  scanf("%d",&n);
  int i;
  ll k,j;
  for(i=0;i<n;i++)
   scanf("%lld",&arr[i]);
  for(i=0;i<n;i++)
  {
          scanf("%lld",&k);
          arr[i]+=k;
  }
```

```
    for(i=0;i<n;i++)
    {
        scanf("%lld",&k);
        arr[i]+=k;
        push_min(arr[i]);
    }
    int q;
    scanf("%d",&q);
    while(q--)
    {
        scanf("%lld",&k);
        if(k>size1+size2)
        {
            printf("-1\n");
            continue;
        }
        else
        {
            if(size1>k)
            {
                while(size1!=k)
                {
                    j=extract_max();
                    push_min(j);
                }
            }
            else
            {
                while(size1!=k)
                {
                    j=extract_min();
                    push_max(j);
                }
            }
            printf("%lld\n",extract_max());
        }
    }
    return 0;
   }
```

18)ii)	BT&T has 256 million customers. Their current telephone directory consists of many heavy volumes typeset in small point size, which are expensive to print and inconvenient to use. To overcome the above problems, BT&T has decided to set up an online computerized directory, and their software engineers are debating what the most efficient data structure is for the purpose. Assume that the BT&T computer can compare two names in one microsecond. i) One of the engineers suggests implementing the on-line directory as an unsorted linked list. With this implementation, give an estimate of the worst-case search and insertion times. ii) Another engineer wants to implement the online directory as an AVL tree. With this implementation, give an estimate of the worst-case search and insertion times. iii) A third engineer proposes the use of a sorted array. What kind of	04	CO3	K3

ANS	argument can the third engineer make to support the proposal? iv) Compare and contrast between all the three proposals and state which one among them is more efficient. i) For worst case search, time complexity is O(n). If the insertion into linked list is done without using tail pointer, then time complexity for insertion takes O(1), else O(n). ii) AVL tree is a height balanced search tree. Hence it takes O(logn) for both insertion and search. iii) Binary search can be used if the array is sorted. Hence it takes O(logn) for search and O(n) for insertion. iv) Usage of AVL tree is best compared to sorted array.			
19)i) ANS	Perform the following operations using Red Black Tree. 1) List out the properties of red black tree. State the balancing property of red black tree. 2) Insert the letters {A, L, G, O, R, I, T, H, M} in this order into an empty red-black tree. Show the sequence of operations performed in each step. 3) Justify the answers for the following assertions regarding the Tree data structure. a) Is it true that in the worst case, a red-black tree insertion requires O(1) rotations? b) Is it true that walking a red-black tree with ‘n’ nodes in in-order takes Θ(nlogn) time? c) Given a red-black tree with ‘n’ elements, how fast can you sort them using the tree? d) Is it true that all Red black trees are BSTs and vice-versa? 4) Is the tree given below a binary search tree or a red black tree? Justify with the properties of each tree. Nodes: 12, 50, -5, -6, 135, 80 are colored RED Nodes: 19, 35, 75, 100, 0, -10, -8 are colored BLACK  19 12 35 0 50 -10 75 -5 135 -8 100 -6 80 1) • Every node is either red or black. • Every leaf (NULL) is black. • If a node is red, then both its children are black. • Every simple path from a node to a descendant leaf contains the same number of black nodes. 2)	16	CO3	K4


1) A Black
2) A Black
L Red
3) A Black
L Red
G Red
Restructuring (RL)
G Black
A Red
L Red
4) G Black
A Red
L Red
O Red
Recoloring
G Black
A Black
L Black
O Red
5) G Black
A Black
L Black
O Red
R Red
Restructuring (RR)
G Black
A Black
O Black
L Red
R Red
6) G Black
A Black
O Black
L Red
R Red
I Red
Recoloring
G Black
A Black
O Red
L Black
R Black
I Red

	 7) G Black A Black O Red L Black R Black I Red T Red  8) G Black A Black O Red L Black R Black I Red T Red H Red Restructuring (LL) G Black A Black O Red I Black R Black H Red L Red T Red  9) G Black A Black O Red I Black R Black H Red L Red T Red M Red Recoloring G Black A Black O Red I Red R Black H black L Black T Red M Red Restructuring (RL) I Black G Red O Red A Black H Black L Black R Black M Red T Red Resultant Tree 3) a) True. Red black tree will take atmost 2 rotations. Hence requires O(1) rotations. b) False. Walking a red-black tree taes O(n) time for inorder traversal. c) O(n). Using in-order traversal, elements in the tree can be displayed in sorted order d) False. All red black trees are binary search trees but not viceversa. 4) The given tree satisfies all the properties of red black tree as well as binary search tree. Hence the given tree is both binary search tree and red black tree.			
(OR)				
20)i)	Mr. Alfred is the founder of FooLand Constructions. He always maintains a ‘Black list’ of potential employees which can be fired at any moment without any prior notice. This company has N employees (including Mr. Alfred), and each employee is assigned a rank (1 <= rank <= N) at the time of joining the company. Any person can have any rank from 1 to N. No two persons have the same rank. The company has a hierarchically organized management. Each employee has one immediate senior but can have any number of seniors. If a person A is the senior of B and B is senior	16	CO2	K6

of C, this implies that A is also a senior of C. Note that Mr. Alfred has no senior to him. Mr. Alfred has a strange and unfair way of evaluating an employee’s performance. Those employees the sum of whose rank and the number of his/her seniors is a prime number is put up on the ‘Black list’. Write a complete implementation to find out the number of ‘Black listed’ employees.

Note: The list won’t contain Mr. Alfred’s name as he is the founder of the company and the list is managed by him!

Input: The first line of input consists of a single integer T denoting the number of test cases. Then T test cases follow. Each test case consists of two lines. The first line contains a single integer N. The next line contains N integers. The ith integer (1 <= i <= N) represents the rank of the immediate senior of the ith employee (i.e. the employee with rank = i). If the ith integer is 0, it represents that employee with rank = i is Mr. Alfred as he has no superior.

Output: For each test case, print in a new line the number of ‘Black listed’ employees.

```
 Input:
2
4
0 1 1 2
3
2 3 0
Output:
1
2
```

Explanation: For 1st case: The employee with rank 1 is Mr. Alfred. The employee with rank 2 and rank 3 have Mr. Alfred as their senior. 2 + 1 = 3 is a prime and 3 + 1 = 4 is not a prime. The employee with rank 4 has Employee 2 as its immediate senior. Hence, number of seniors of Employee 4 = 2 (i.e. employee 2 and Mr. Alfred) 4 + 2 = 6 is not a prime. Hence, the result is 1. For 2nd case: Employee 3 is Mr. Alfred. Employee 2 is Mr. Alfred’s immediate subordinate. His level is 1. 2 + 1 = 3 is a prime. Employee 1 is Employee 2’s immediate subordinate. His level is 2. 1 + 2 = 3 is a prime. Hence, the result is 2.

ANS

```
#include<stdio.h>
int N;
int arr[100001];
int isPrime[100001];
int cost[100001];
int sum = 0;
int value = 0;
void prime(){
        int i, j;
        for (i = 2; i < 100001; i++)
                isPrime[i] = 1;
        for (i = 2; i < 100001; i++)
                if (isPrime[i])
                        for (j = 2; i*j < 100001; j++)
                                isPrime[i*j] = 0;
        }
int dfs(int MD, int rank){
        int senior;
```

```
    if (rank < 0 || rank > N || cost[rank] != 0)
        return cost[rank];
    if (rank == MD){
        cost[rank] = 0;
        return 0;}
    senior = arr[rank];
  value = dfs(MD, senior) + 1;
     cost[rank] = value;
     return value;
}
int main(){
    int test, T, i, j,x;
    prime();
    scanf("%d", &T);
    for (test = 1; test <= T; test++){
        scanf("%d", &N);
        for (i = 1; i <= N; i++){
            scanf("%d", &arr[i]);
            if (arr[i] == 0)
                x = i;
            cost[i] = 0;
        }
        value = 0;
        sum = 0;
        for (i = 1; i <= N; i++)
            if (x != i)
             dfs(x,i);
        for (i = 1; i <= N; i++)
            if (cost[i] != 0 && isPrime[i + cost[i]])
                sum++;
        printf("%d\n", sum);}
    return 0;
}
```