B-Tech 3rd Semester Mid Term Examination

December 2020

Subject: Data Structures And Algorithms

[CS 2103]

Date of Examination: 26/12/2020

Name: Abhiroop Mukherjee

Enrolment Number: 510519109

Previous Enrolment Number: 510719007

G-Suite ID: 510519109.abhirup@students.iiests.ac.in

No. of Sheets uploaded: 10

1) a) given $(A+B)*D + E/(F+A*D)+C$

adding brackets

$$\left[\left[\left[(A+B)*D\right] + \left\{\left[E / (F+(A*D))\right]+C\right\}\right]\right]$$

shifting operators to right.

$$\left[\left[\left[(AB+)D*\right] * \left[E\ (F\ (AD*)+)\right]+\right] C\ +\right]$$

$\therefore$ Ans: $AB+D*EFAD*++C+$

1) b) → consider converting a^b^c to postfix

Step	expression	stack	output
1	a	empty	a
2	^	^	a
3	b	^	ab
4	^	^	ab^
5	c	^	ab^c
6	$	empty	ab^c^

now if we convert ab^c^ to infix, we get

$$(a^b)^c = a^{bc} \neq a^{(b^c)}$$

→ for solving this problem, we need different priority for instack and incoming operator.

1) d) To find k^{th} node from last in linked list, we do the following.

i) Initialise two pointer
   a) p which points to first element
   b) q which points to k^{th} node from start.

ii) iterate through the linked list, ~~going~~ increment "incrementing" both p & q, till q is in end.

iii) p will point to k^{th} element from last.

Procedure

// list has ~~three~~ two attribute, ~~[illegible]~~ data, next

```
kth-last (node* list)
{
  set p = list
  set q = kth node from start

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

  return p;  // p will point to kth node from last.
}
```

1)e → A Binary Search Tree is a Binary Tree with following properties

i) ~~Eo~~ Data of Left child of a node will be less than Data of node

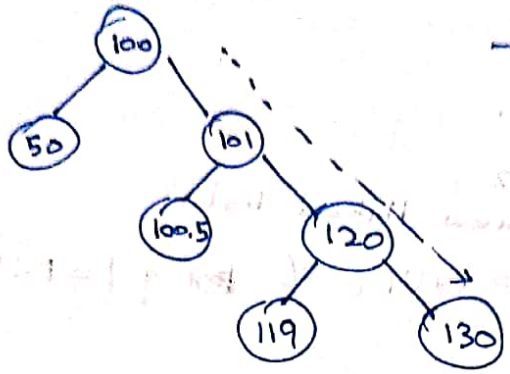
ii) Data of right child of a node will be more than ~~rig~~ the data of the node

iii) Both children of ~~the~~ node are also Binary Search Tree.

→ If for a Binary Tree, if every node of tree follows these three property, then we can say that the Binary Tree is a Binary Search Tree.

Df) → Due to the structure of Binary Search Tree, the data with maximum value will always be in the right most node.

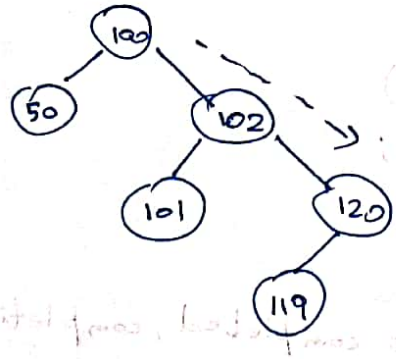
Eg



→ maximum element = 130

→ So to get max element, we start from root, and go to rchild of node, till it does not exist

Eg 2



max element = 120

procedure // root has three attribute: rchild, lchid, data.

```
max_BST (treenode * root)
{
    set p = root;
    while ( p->rchild != NULL )
    {
        p = p->rchild;
    }
    return p->data;
}
```

3)a) ~~to sort L_1 and L_2, we do the f~~

procedure

```
//node has two attibute, data, next
merge (node* L1, node* L2)
{
  set p = L1
  set q = L2
  make r, a new linked list.
  while (p != NULL & q != NULL)
  {
    if (p->data < q->data)
    { insert(r, p)  // insert node p in r
      p = p->next;
    }
    else
    {
      insert (r, q)
      q = q->next;
    }
  }

  // now if one LL is completed, completing other one.

  while (q != NULL)
  {
    insert (r, q)
    q = q->next;
  }
  while (p != NULL)
  { insert (r, p)
    p = p->next;
  }

}// end of function.
```

3) b) we can implement this using Linked list

→ suppose a node has following attribute,

i) data to be accessed

ii) no. of time accessed.

→ and suppose list is sorted according to no. of time accessed, such that the first element of Linked list ~~has~~ is the element with maximum value of "no. of time accessed".

→ This way, the program will iterate through the data with ~~max~~ high frequency before going to low frequency.

→ Also new data will be added in end with 0 ~~no~~ in "no. of time accessed".

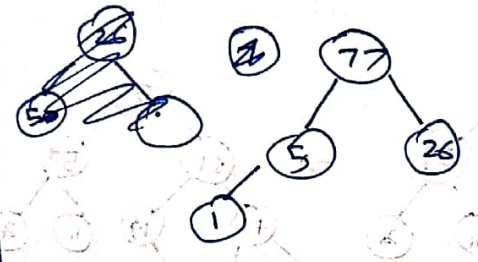
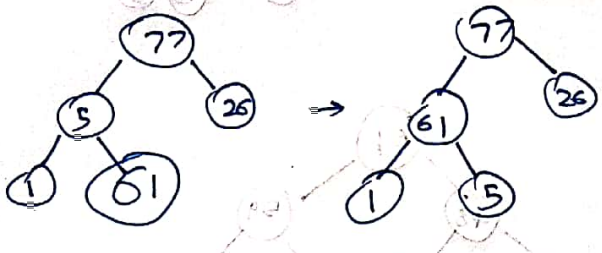
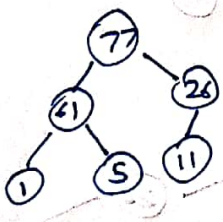
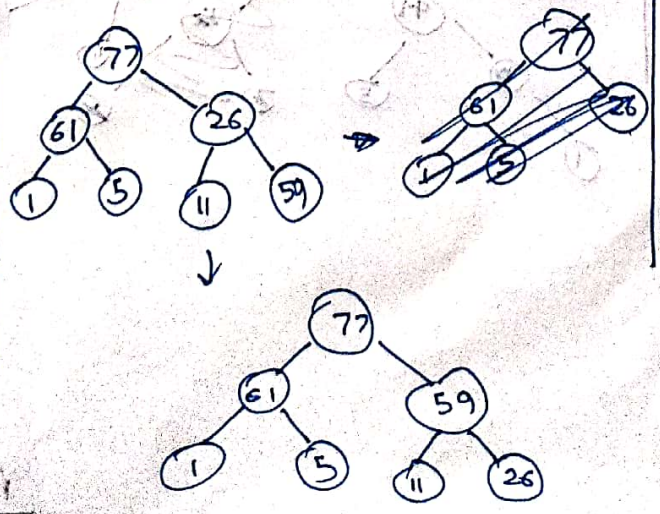
→ when a data is accessed, it's frequency ("no. of time accessed") ~~is~~ will increment, and that ~~data~~ node will change it's position accordingly to make the whole linked list sorted.

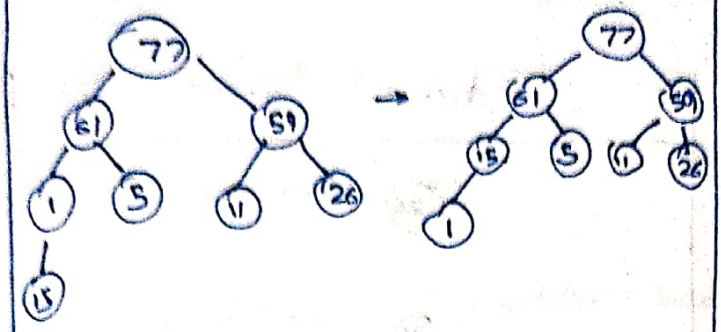
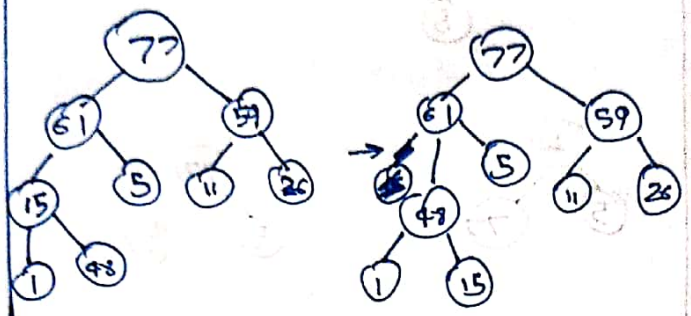
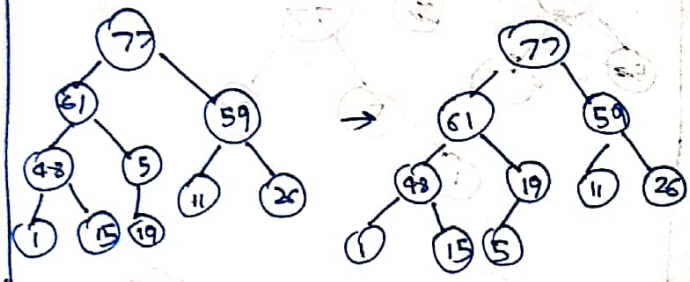
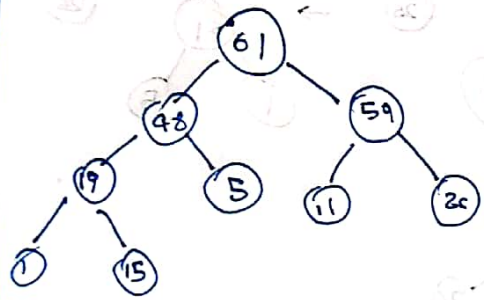
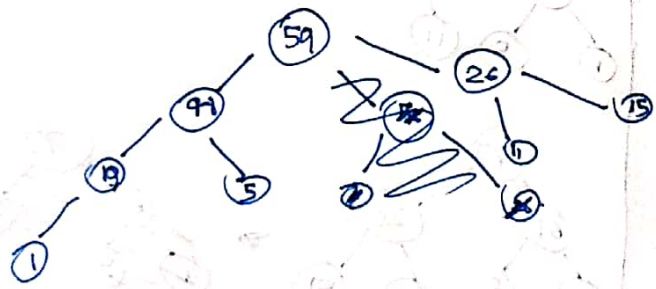
4) a) ~~proc~~ procedure [Recursive]

~~swap_LR (node*~~

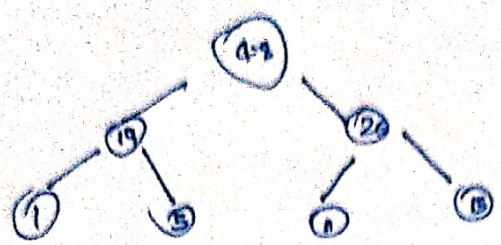
```
swap_LR (treenode* node)
{
    // node has following attribute: lchild, rchild, data

    if (node == NULL)
        return ;

    treenode* temp = node -> lchild ;
    node -> lchild = node -> rchild ;
    node -> rchild = temp;

    swap_LR (node -> lchild); // recursion to
    swap_LR (node -> rchild); // its child.
}
```

4)b) given 26, 5, 77, 1, 61, 11, 59, 15, 48, 19

no.	tree	~~output~~ output
1) 26		
2) 5		
3) 77		
4) 1		
5) 61		
6) 11		
7) 59		

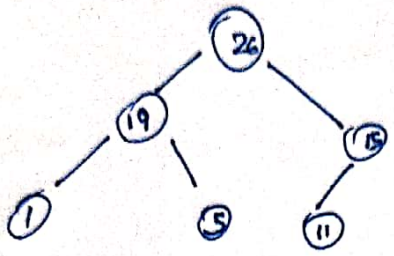
number	Tree	Output
8) 15		
9) 48		
10) 19		
11) delete		77
12) delete		77, 61

13)



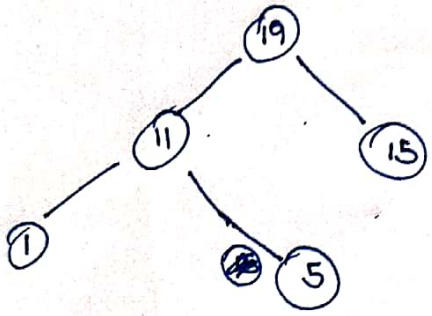
77, 61, 59

14)



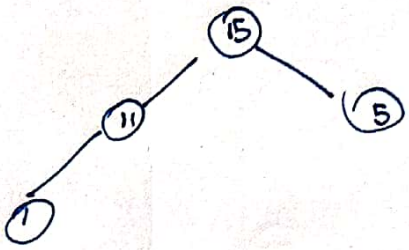
77, 61, 59, 48

15)



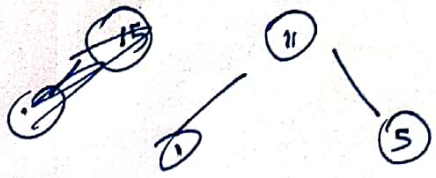
77, 61, 59, 48, 26

16)



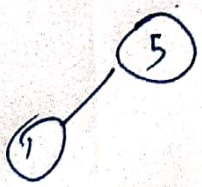
77, 61, 59, 48, 26, 19

17)



77, 61, 59, 48, 26, 19, 15

18)



77, 61, 59, 48, 26, 19, 15, 11

19) 77, 61, 59, 48, 26, 19, 15, 11, 5

20) 77, 61, 59, 48, 26, 19, 15, 11, 5, 1