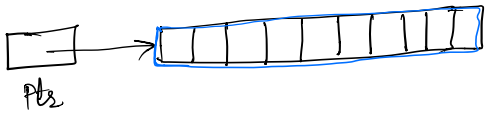
Solution of Mid Term Re-Examination - 2019
Data Structure (051403)

Q1. a) Queue data structure will be used, since ticketing is done on first come first serve basis.

b) Ptr is a pointer to an array containing 10 integers.



c)

Open Addressing	Closed Addressing
i) All the addresses (index) are open for all keys i.e. if we get different keys hashing to same index then elements will be stored at different index depending on collision resolution Technique.	i) All the key hashing to the same index will be accessed from the same index.
ii) No need of any additional data structure. All the keys resides in the hash table only.	ii) Additional data structure is required to avoid collision i.e linked list is used to store keys.

d) Minimum height of the tree having 8192 nodes

$= 8192 - 1 = 8191$

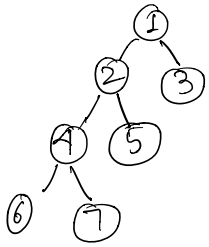
These are generally skewed trees.

Maximum height of the tree having 8192 nodes

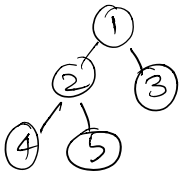
$$= \lfloor \log_2 n \rfloor = \lfloor \log_2 8192 \rfloor = 13$$

These are generally full / complete tree.

e) Strictly binary Tree is the one in which each node has either 0 or 2 children.

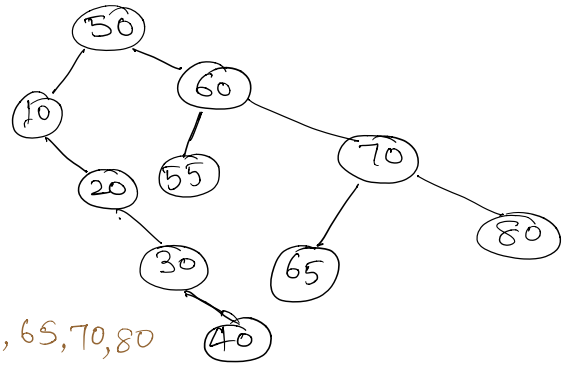


Complete binary tree is the one where all levels except last one is full and nodes at last level is left aligned.



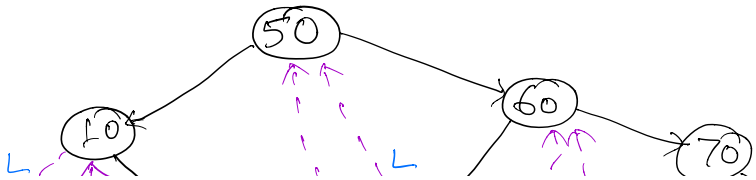
Q2. a) – Build a Binary Search Tree (BST).
– Traverse the tree in Inorder.
– Set the empty pointers of nodes to InOrder successor and predecessor.
i.e. left pointer will point to predecessor } If not pointing to left/right subtree child.
right pointer will point to successor }

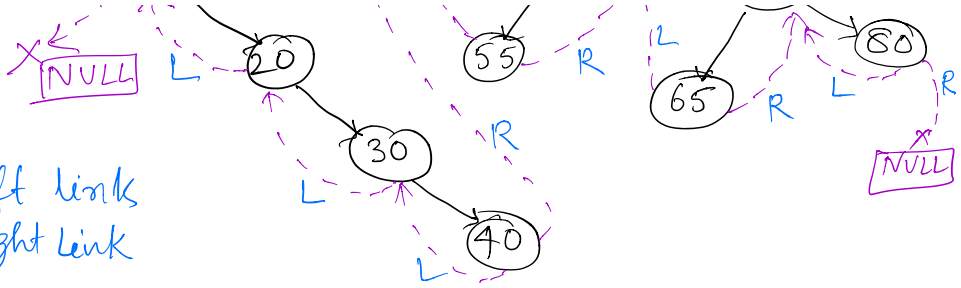
BST ⇒



InOrder Traversal
⇒ 10, 20, 30, 40, 50, 55, 60, 65, 70, 80

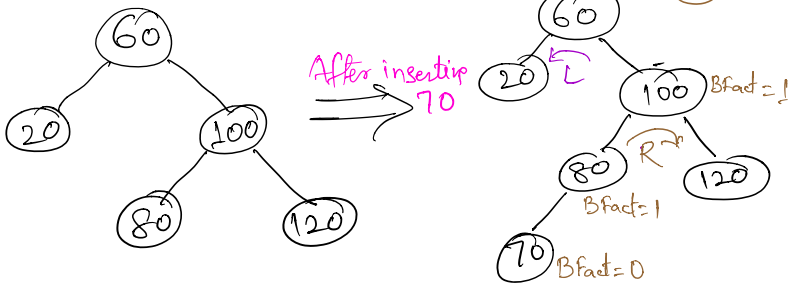
In-threaded BST





L ⇒ Left link
R ⇒ Right Link

b) AVL Tree



Tree is not balanced since 60 has -2 balance factor.

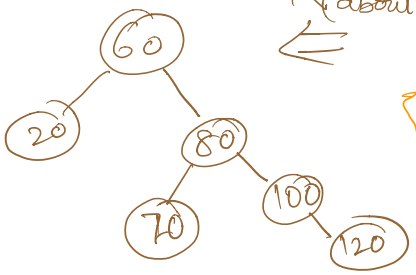
Balance factor = LH - RH

Empty tree height = -1

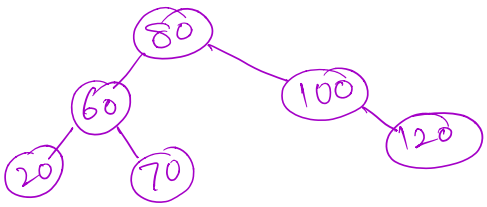
i.e Bfact of 70 = -1 - (-1)
= 0

(RL) Rotation is required since 70 is in 60's right subtree (R) and 70 is in left subtree/node of 100 (L).

R (about 100)



L (About 60)



Final AVL

c) Given compiler is Gcc & int size = 4 bytes

i) 12 bytes

ii) 28 bytes

Q3. a) i) A * B/C - (D+E ^F) - G

- G - (F ^E+D) - C/B * A ⟸ Reversed string
- Perform postfix

Scanned Symbol	stack	output (Prefix)
G		G
-	-	G
(	-(	G
F	-(	GF
^	-(^	GF
E	-(^	GFE
+	-(+	GFE^
D	-(+	GFE^D
)	-	GFE^D+
-	-	GFE^D+-
C	-	GFE^D+-C
/	-/	GFE^D+-C
B	-/	GFE^D+-CB
*	-*	GFE^D+-CB/
A	-*	GFE^D+-CB/A

GFE^D+-CB/A*-

Reverse the output

⇒ -*A/BC-+D^EFG

Yet another way A*B/C-(D+E^F)-G

Reverse ⇒ G-(F^E+D)-C/B*A

⇒ G-(L+D)-C/B*A , L = FE^

⇒ G-M-C/B*A , M = LD+

⇒ G-M-N*A , N = CB/

⇒ G-M-O , O = NA*

⇒ P-O , P = GM-

⇒ PO-

⇒ GM-O-

⇒ GLD+-O-

⇒ GFE^D+-O-

⇒ GFE^D+-NA*-

⇒ GFE^D+-CB/A*-

Without stack

Reverse ⇒ -*A/BC-+D^EFG

ii) Similar steps

Ans: +*A*B/^DCE-FG

b) REVERSE(S, i, j) ⇒ Reverse the data between i & J (inclusive)

for ex: If S array has

1	2	3	4	5	6	7	8	← Index
1	3	4	5	2	6	7	8	

S ⇒

@ REVERSE(S, 2, 5) is called, then Array S will be

1 2 5 4 3 6 7 8

Reversed

So.

REVERSE(S, 1, k) ⇒ Reverse first K elements

REVERSE(S, K+1, n) ⇒ Reverse remaining n-k elements

REVERSE(S, 1, n) ⇒ Reverse all elements

Considering above array & K = 3

REVERSE (S, 1, 3) ⇒ 4 3 1 5 2 6 7 8

REVERSE (S, 4, 8) ⇒ 4 3 1 8 7 6 2 5

REVERSE (S, 1, 8) ⇒ 5 2 6 7 8 1 3 4

This is nothing but K times shifting the elements toward left with rotation (first elements goes to last Place)

c) It has a memory leak. Since free is called after loosing valid pointer in P. That is, before P=NULL, it was containing a valid pointer in heap. P=NULL lost that pointer & using free with NULL has no effect. So the memory is not freed by lost which is nothing but memory leak.

Q4. a) InOrder Traversal ⇒ 4 2 5 1 6 7 3 8

PostOrder Traversal ⇒ 4 5 2 6 7 8 3 1

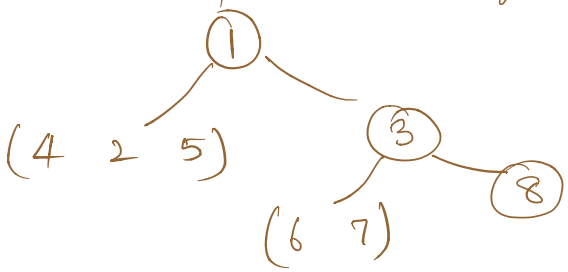
* Post Order Traversal reads the root of the tree at the end. So 1 is root of the tree here.

Using above knowledge, Inorder Traversal can be splitted in two half. (Root is some where in between nodes)

i) 1 is root (from post order Traversal)

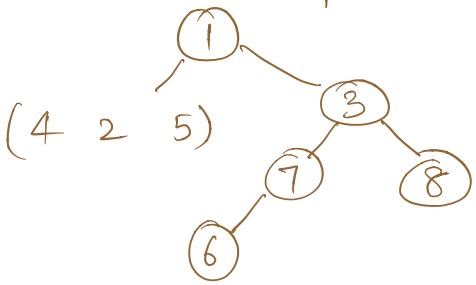
Tree will be_

ii) 3 is the root, which is in right subtree of 1.



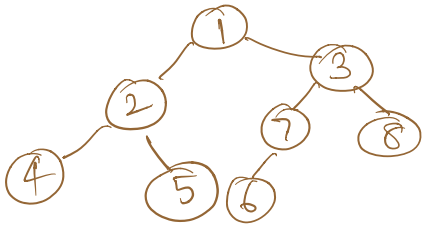
iii) 8 is the root. Since it is a single element at right subtree there is nothing to do.

iv) 7 is the root of left subtree of 3.



v) 6 is the root and single element in subtree. So nothing to do.

vi) 2 is the root.



vii) 4 & 5 are next roots at left & right subtrees respectively and single elements. So there is no need of any further step.

b) Input: A Queue with 1 2 3 4 5 6 7 8 9 10

Temp. storage: Stack (only one)

(No more Queue or stack or any other data structure is allowed)

Output: 1 6 2 7 3 8 4 9 5 10

[Queue is automatically shifting according to the no. of elements]

* Since it is Queue, we can jump to any element. There is only one exit that is from front.

Since the given example Queue has 10 elements, half of it will be 5.

i) Deque $n/2$ elements and push in stack
ii) Pop all $n/2$ elements one by one, enque $n/2 - 1$ in Q and print last element.
iii) perform step i) & ii) once again
iv) Deque $n/2 - 1$ elements to stack & enque it back.
v) Perform step iv) once again

Perform these step $n/2$ times —
i.e run a loop from
($n/2 \rightarrow 0$ times)
(5, 4, 3, 2, 1)

Queue	stack	output
6 7 8 9 10	1 2 3 4 5	
6 7 8 9 10 5 4 3 2	6 7 8 9 10	1
5 4 3 2		1 6
5 4 3 2 10 9 8 7		
10 9 8 7 2 3 4 5		1 6 2
2 3 4 5 7 8 9 10	2 3 4 5	
7 8 9 10		1 6 2 7
7 8 9 10 5 4 3	7 8 9 10	
5 4 3 1 0 9 8		1 6 2 7 8 4 3 5 1 0

c) Size of the table = 10

Index = key % 10 + i^2, $i \geq 0$

$Index_{60} = 60 \% 10 + 0$
$= 0$

$Index_{24} = 24 \% 10 + 0 = 4$

Similarly — —

0	1	2	3	4	5	6	7	8	9
60	30	22	23	24	45	25	37	10	

$Index_{30} = 30 \% 10 = 0 \Rightarrow$ Collision
$Index_{30} = 30 \% 10 + 1^2 = 1$

Q5. a) i)

```
struct Node * Loopcheck(struct Node *start)
{
  struct Node *P1 = start; int flag = 1;
  struct Node *P2 = start;
  while ( P2->next != NULL || P2 != NULL )
  {
    P1 = P1->next;
    P2 = P2->next->next;
    if( P1 == P2)
    {
      flag = 0; break;
    }
  }
  if(flag) return NULL;
  else return P1;
}
```

ii)

```
struct Node * LoopStart ( struct Node * start)
{
  struct Node * loop = Loopcheck(start);
  if ( loop == NULL) return NULL;
  else { struct Node *P1 = start
         struct Node *P2 = loop;
    while ( P1 != P2 )
    {
      P1 = P1->next;
      P2 = P2->next;
    }
    return P1;
  }
}
```

b) AVL is a height balanced tree, where there is a Balance factor -1, 0, 1. Where as BST is binary search tree which has no restriction on height of the tree.

for Ex: 10 5 20 15 25 30

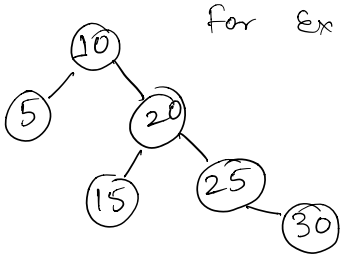


figure 1

For this input seq^n, figure 1 is the BST, whose height of the tree is 3.

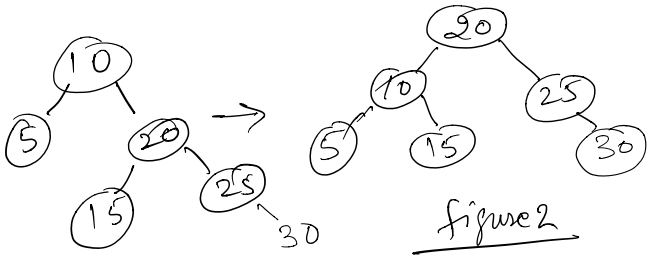
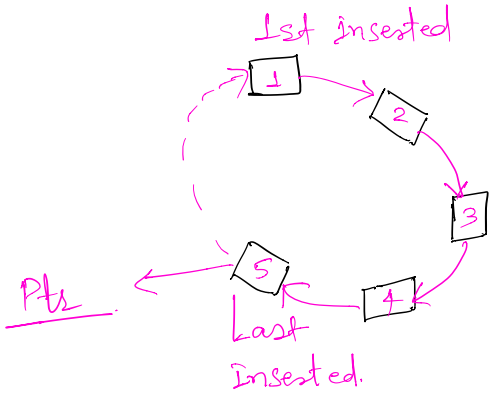


figure 2

For the same input, figure 2 is the tree and the height of the tree is 2.

c) Circular list will be used to implement Queue.



Just by using one pointer Ptr, it is possible to enque & deque in the Queue most efficiently in few steps.