Sub Name Code: DSA
Subject Code: CS-2001
Program Name: B.Tech
Semester: III (Regular)
Year - 2019

AUTUMN END-SEMESTER SOLUTION- 2019
KIIT, Deemed to be University, Bhubaneswar-24
Data Structure and Algorithm
CS-2001

Time: 3 Hours **Full Mark: 50/60**

Answer any SIX questions.
Question paper consists of four sections-A, B, C, D.
Section A is compulsory.
Attempt minimum one question each from Sections B, C, D.
The figures in the margin indicate full marks.
Candidates are required to give their answers in their own words as far as practicable and <u>all parts of a question should be answered at one place only</u>.

SECTION A

Q 1. [1×10 =10] or [2×10 =20]

a **Define an Abstract Data Type (ADT). Give any two examples.**

Scheme of Evaluation : Definition [0.5]
Example [0.5]

Model Solution:

Abstract Data type(ADT) is a type (or class) for objects whose behaviour is defined by a set of value and a set of
operations. It only specifies, what operations are to be performed but implementation details of these operations are
not specified. It is called “abstract” because it gives an implementation-independent view. The process of
providing only the essentials and hiding the details is known as abstraction.
Example:
Stack ADT: In Stack ADT Implementation instead of data being stored in each node, the pointer to data is
stored. The program allocates memory for the data and address is passed to the stack ADT.
List ADT: The data is generally stored in key sequence in a list which has a head structure consisting
of count, pointers and address of compare function needed to compare the data in the list.

b **Write a function to find out the number of nodes of degree one, two, and zero in a binary tree.**
Scheme of Evaluation: 2 marks should be distributed among one, two and zero degree judiciously.

Model Solution:

```
struct Node{
    int data;
    Node *left, *right;
};
int d1 = 0, d0 = 0, d2 = 0;
int nodecount (struct Node* root)
{
if (root == NULL)
    return 0
else
{
    if (root->left==NULL && root->right==NULL) //count the nodes with degree zero
       d0++;
   if ((root->left == NULL && root->right != NULL) ||
   (root->left != NULL && root->right == NULL)) //count the nodes with degree one
       d1++;
   if (root->left!=NULL && root->right!=NULL) //count the nodes with degree two
       d2++;
   nodecount(root->left);
   nodecount(root->right);
}
}
```

c **Name the data structure used for recursion. Explain with an example.**

Scheme of Evaluation: Name of the data structure [0.5 marks]
Explanation with example [0.5 marks]

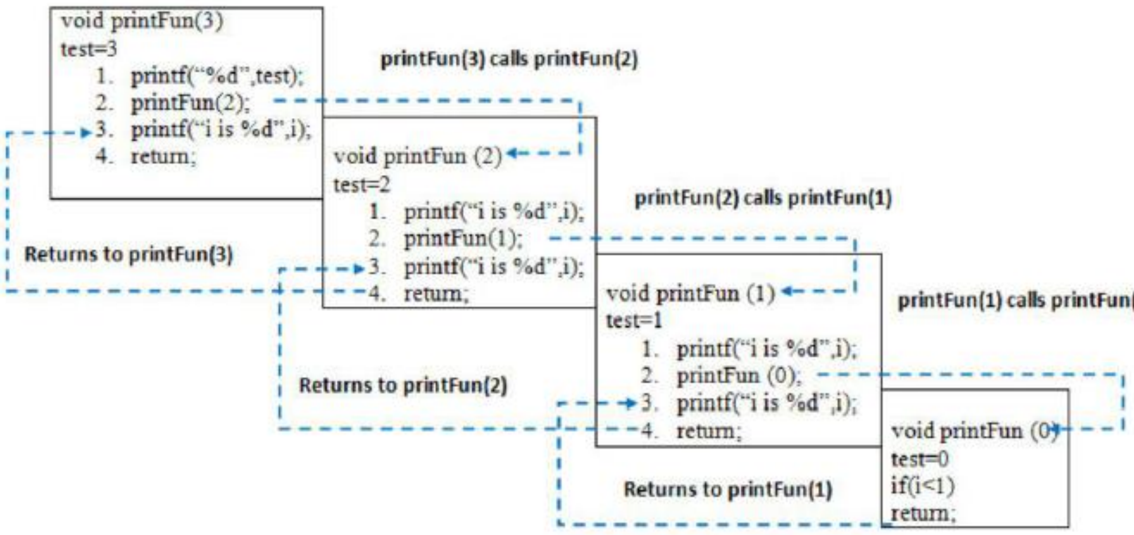
Model Solution:

Stack is used for recursion function.
Example:

```
  void printFun(int test)
  {
    if (test < 1)
       return;
    else
   {
      Printf("%d",test)
      printFun(test-1); // statement 2
      Printf("%d",test)
      return;
    }
  }
int main()
{
int test = 3;
printFun(test);
}
```

When printFun(3) is called from main(), memory is allocated to printFun(3) and a local variable test is initialized to 3 and statement 1 to 4 are pushed on the stack as shown in below diagram. It first prints ‘3’. In statement 2, printFun(2) is called and memory is allocated to printFun(2) and a local variable test is initialized to 2 and statement 1 to 4 are pushed in the stack.

Similarly, printFun(2) calls printFun(1) and printFun(1) calls printFun(0). printFun(0) goes to if statement and it return to printFun(1). Remaining statements of printFun(1) are executed and it returns to printFun(2) and so on. In the output, value from 3 to 1 are printed and then 1 to 3 are printed. The memory stack has been shown in below
diagram.


printFun(3) calls printFun(2)
printFun(2) calls printFun(1)
printFun(1) calls printFun(0)
Returns to printFun(3)
Returns to printFun(2)
Returns to printFun(1)

```
void printFun(3)
test=3
  1. printf("%d",test);
  2. printFun(2);
  3. printf("i is %d",i);
  4. return;
```

```
void printFun (2)
test=2
  1. printf("i is %d",i);
  2. printFun(1);
  3. printf("i is %d",i);
  4. return;
```

```
void printFun (1)
test=1
  1. printf("i is %d",i);
  2. printFun (0);
  3. printf("i is %d",i);
  4. return;
```

```
void printFun (0)
test=0
if(i<1)
return;
```

Note: Any recursive function example can be considered.

d **What is a polynomial? How it can be represented using array and linked list? Compare both the representation.**

Scheme of Evaluation: Polynomial representation [0.5 marks]
Comparison [0.5 marks]

Model Solution:

Polynomial is an expression constructed from one or more variables and constants, using only the operations of addition, subtraction, multiplication, and constant positive whole number exponents.
A polynomial may be represented using arrays and linked lists.

Array representation assumes that the exponents of the given expression are arranged from 0 to the highest value (degree), which is represented by the subscript of the array beginning with 0. The coefficients of the respective exponent are placed at an appropriate index in the array. Considering single variable polynomial expression, array representation is

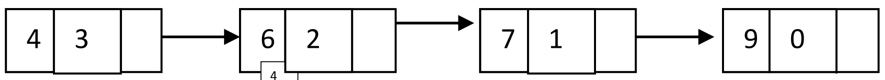
4x^3+ **6**x^2+**7**x+9

arr	9	7	6	4	(coefficients)
	0	1	2	3	(exponents)

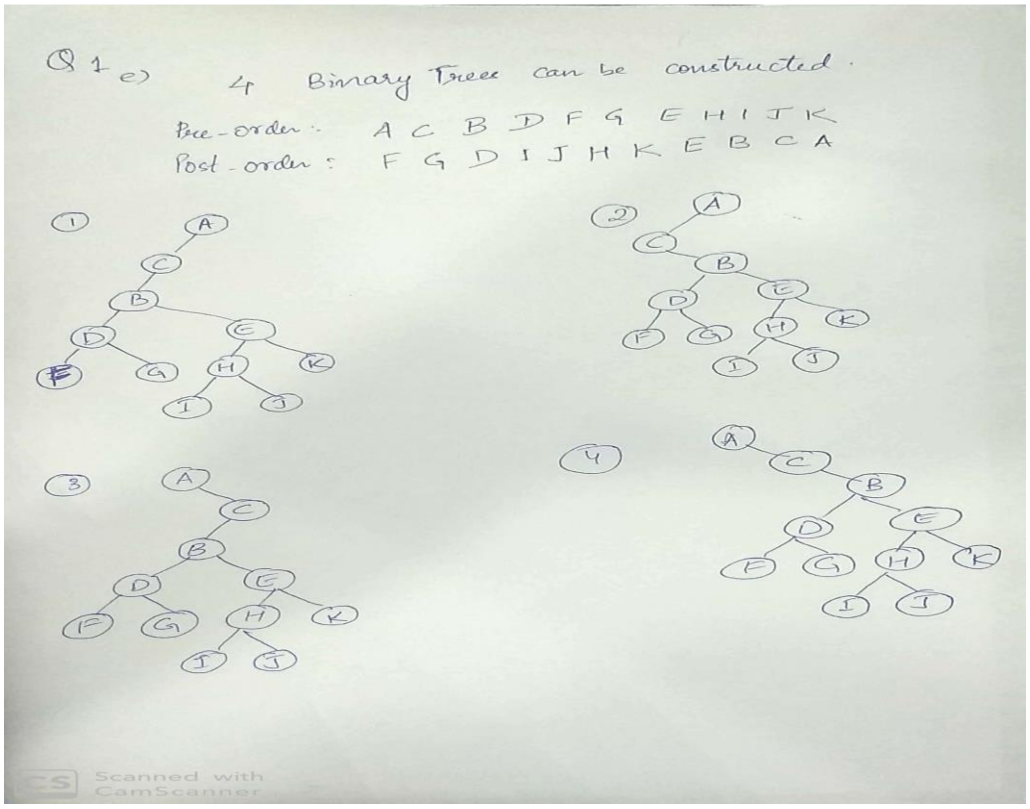
Polynomial can be represented using Linked List :

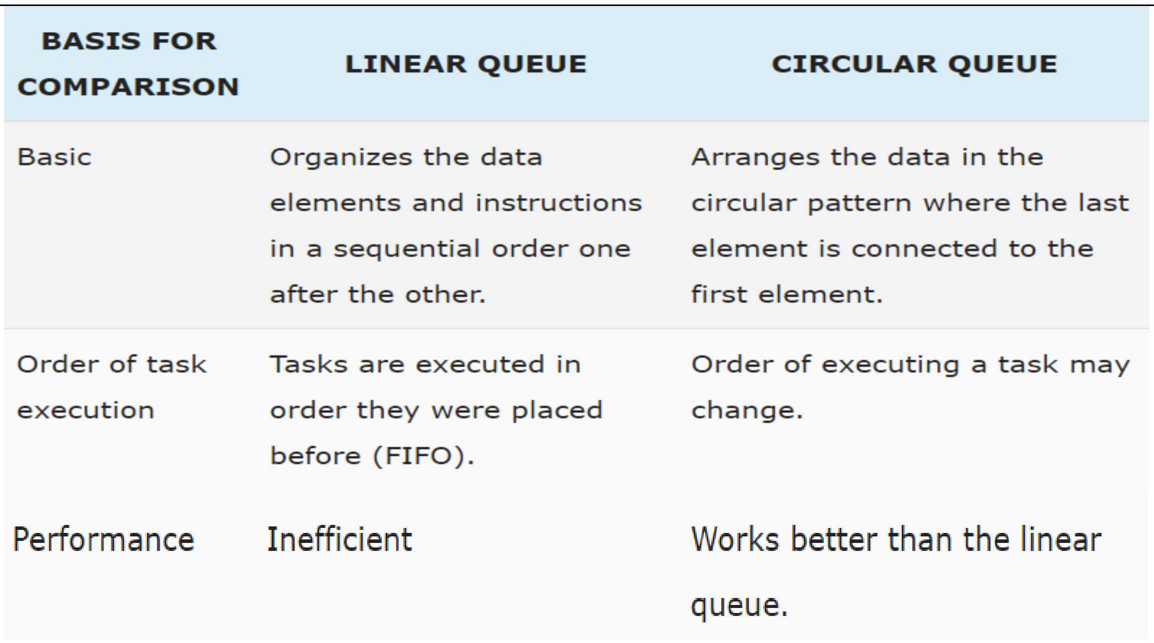
In each node the exponent field will store the corresponding exponent and the coefficient field will store the corresponding coefficient. Link field points to the next item in the polynomial.

4x^3+ **6**x^2+**7**x+9



Linked list representation of polynomial is better than the array representation as it stores only

	the required data value.
e	**How many number of binary tree can be constructed from the following set of tree traversal? Draw all trees.** **Scheme of Evaluation**: Count of different type of tree: [0.5 mark] Drawing of all the tree: [0.5 mark] **Model Solution:**  Q 1 e) 4 Binary Tree can be constructed. Pre-order : A C B D F G E H I J K Post-order : F G D I J H K E B C A
f	**Differentiate between linear queue and circular queue. List any two application of queue ADT.** **Scheme of Evaluation:** Differentiate: 0.5 mark Application: 0.5 mark **Model Solution:**



BASIS FOR COMPARISON	LINEAR QUEUE	CIRCULAR QUEUE
Basic	Organizes the data elements and instructions in a sequential order one after the other.	Arranges the data in the circular pattern where the last element is connected to the first element.
Order of task execution	Tasks are executed in order they were placed before (FIFO).	Order of executing a task may change.
Performance	Inefficient	Works better than the linear queue.

Application of Queue ADT
1) Queue of network data packets to send
2) Queue of programs/processes to be run

g

Write a function to copy the content of one stack ADT to another stack ADT without using any additional user defined stack ADT.

Scheme of Evaluation: For correct answer full mark should be awarded. No step marking. Variation of solution/ code are possible.

Model Solution:

```
void insert_at_bottom(node **stack, int data)
{
    if( isempty(*stack) ){
        push(stack,data);
        return;
    }
    int temp=pop(stack);
    insert_at_bottom(stack,data);
    push(stack,temp);
}


void rev_stack(node **stack)
{
    if( isempty(*stack) ) return;
```

```
      int temp = pop(stack);
      rev_stack(stack);
      insert_at_bottom(stack,temp);
  }
```

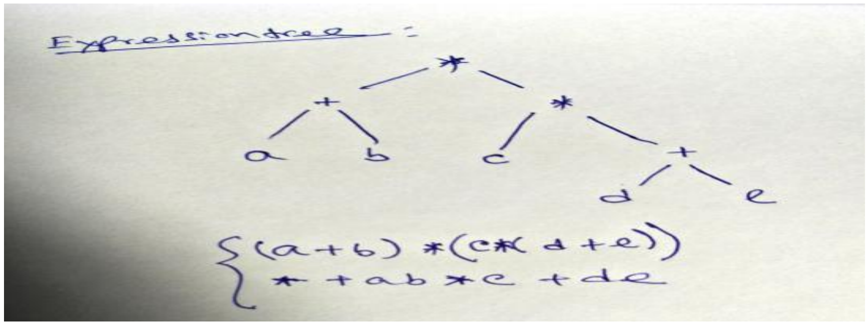
h **What is expression tree? Write the steps followed to construct an expression tree from the following expression.**

Scheme of Evaluation: About Expression tree [0.5 mark]
Steps for the construction [0.5 mark]

Model Solution:

A binary expression tree is a specific kind of a binary tree used to represent expressions. In this tree each internal node corresponds to operator and each leaf node corresponds to operand.

Construction of Expression Tree:
For constructing expression tree for the given expression the stack will be used. The input expression will be scanned from right to left and dependin on the symbol the following steps will be carried out.
1) If character is operand push that into stack
2) If character is operator pop two values from stack make them its child and push current node again.
At the end only element of stack will be root of expression tree.



i **Write the two limitation of applying binary search algorithm on a data structure. Write and justify the time complexity of binary search algorithm.**

Scheme of Evaluation : Limitation [0.5 mark]
Time complexity [0.5 mark]

Model Solution:

<u>Limitation:</u>
1. Array should be in sorted order.
2. Size of array should be large.

Time Complexity:

1. Best Case: In first chance that is at the middle of the array the element can be found. Ω(1)
2. Worst case: Binary search algorithm break the problem size into half in each iteration. So the number of times we need to divide the array size by 2 until only one element is log(n) and thus time complexity is O($\log_2 n$)

j **Suppose FIRST and LAST represents the address of start node and last node in a double linked list. Write a function to convert the double linked list into a double circular linked list.**

Scheme of Evaluation: 2 marks should be given judiciously.

Model Solution:

```
fun(struct node **FIRST, struct node **LAST)
{
  (*LAST)->next = FIRST;
  (*FIRST)->prev = LAST;
}
```

SECTION B

Q 2.

a **What is the advantages of binary search tree over binary tree? Write non-recursive functions for the following operations for a binary search tree**

i) Traverse the tree in pre-order [4+4]

ii) Traverse the tree in level-order

Scheme of Evaluation : Advantage s [1 mark]
Pre-order Traversal [1.5 mark]
Level –order traversal[1.5 marks]

Model Solution:

Advantages of binary search tree over binary tree is

- When we do inorder traversal on BST, we get elements in sorted order. Or conversely if we need to traverse the increasing order of the elements , we can do the in-order traversal of the tree and for decreasing order, we can do reverse in-order traversal on the tree.
- If we implement a balanced binary search tree, we can keep the cost of insert, delete, lookup to O(logN) where N is the number of nodes in the tree. Whereas in a binary tree it cannot be certain to get O(logN) search time always.
- In case of BST, given a random node we can say whether the node is present in left subtree or right subtree, but this cannot be said in case of a binary tree.

Function to traverse the tree in pre-order:

```
preorder_iterative(struct node *ptr)
{
     push(Stack, ptr);
     while(top!=-1)
    {
       ptr = pop(Stack);
       if(ptr!=NULL)
      {
         printf("%d", ptr->info);
         push(Stack , ptr->rchild);
         push(Stack , ptr->lchild);
      }}}
```

Function to traverse the tree in level-order:

```
 front = rear = -1;
   void levelorder (struct node *ptr)
   {
      if (ptr==NULL) return;
      enqueue(Queue, ptr);
      front++;
      while(front ≤ rear)
      {
         ptr= dequeue(Queue);
         printf("%d", ptr->info);
         if(ptr->lchild != NULL)
                 enqueue(Queue, ptr->lchild);
         if(ptr->rchild != NULL)
                 enqueue(Queue, ptr->rchild);

   }}
```

b **What are the limitations of a sparse matrix? How to represent a sparse matrix using a header linked list. Write a function to multiply two sparse matrices represented using header linked list.**

Scheme of Evaluation: Limitation [0.5 mark]
Representation [0.5 marks]
Multiplication Function [3 marks]

Model Solution :

Limitations of a sparse matrix:

- The major limitation of a sparse matrix is that it can contain a lot less information than a non-sparse matrix, since a vast majority of its entries must be equal to 0.
- It reduces the total computation time taken for operations as a lot of elements are zero.

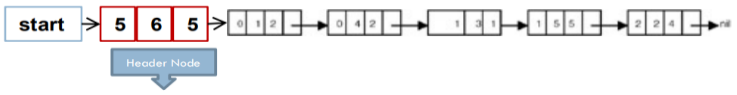
A sparse matrix can be represented using a header linked list as follows:

Header Linked List: For example, consider the sparse matrix

0	2	0	0	2	0
0	0	0	1	0	5
0	0	4	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

```
struct node
{
    int row;
    int column;
    int value;
    struct node *next;
} *start;
```

The linked list can be represented using linked list as shown below



Represents the header node and consists of:

1. Number of rows in the sparse matrix
2. Number of columns in the sparse matrix
3. Number of non-zero elements in the sparse matrix

```
struct node
{ int r, c, ele;
struct node *next;};

struct head
{ int tr, tc, tele;
struct node *next;};

void multiply_sparse(struct head *head1, struct head *head2, struct head **head3)
{
if(head1 → tc != head2 → tr) return;
*head3 → tr = head1 → tr;
*head3 → tc = head2 → tc;
struct node *p1, *p2, *prev;
int c=0;
for(p1= head1→next; p1! = NULL; p1 = p1→next)
for(p2= head2→next; p2! = NULL; p2 = p2→next)
if(p1 → c = = p2 → r)
{
add_end(head3, p1 → r, p2 → c, p1 →info × p2 → info) // this
function insert the new node at the end of 3rd sparse matrix head3
c++; // Initially count the number of node in 3rd sparse matrix
}
for(p1= head3→next; p1!= NULL; p1 = p1→next)// duplicate remove
{ prev = p1;
for(p2 = p1→ next; p2 != NULL;)
if(p1 → r = = p2→ r && p1 → c = = p2→ c)
{
p1 → info = p1 → info + p2 → info
prev → next = p2 → next;
free(p2);
p2 = prev → next;
c--; //Number of nodes decreased due to duplicate removal
}
else
{
prev=p2;
p2 = p2 → next;
}
}
*head3 → tele = c;
}
```

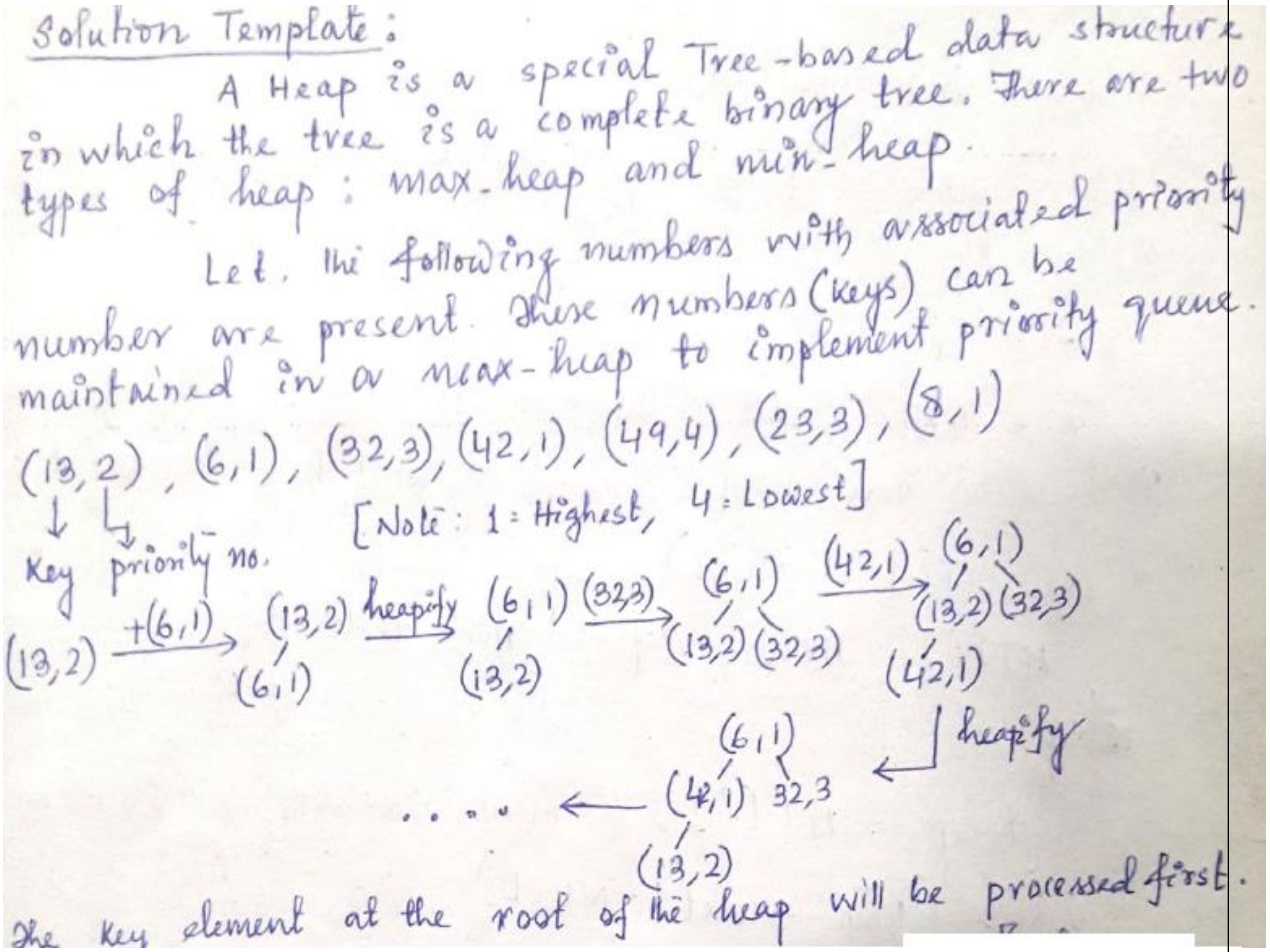
Q 3. [4+4]

a **What is heap? Discuss with example, how priority queue can be implemented using heap.**

Scheme of Evaluation:

What is heap [1 mark]
Explanation of priority queue [2 mark]
Example[1 mark]

Model Solution:



Solution Template:

A Heap is a special Tree-based data structure in which the tree is a complete binary tree. There are two types of heap: max-heap and min-heap.

Let, the following numbers with associated priority number are present. These numbers (keys) can be maintained in a max-heap to implement priority queue.

(13, 2), (6,1), (32,3), (42,1), (49,4), (23,3), (8,1)

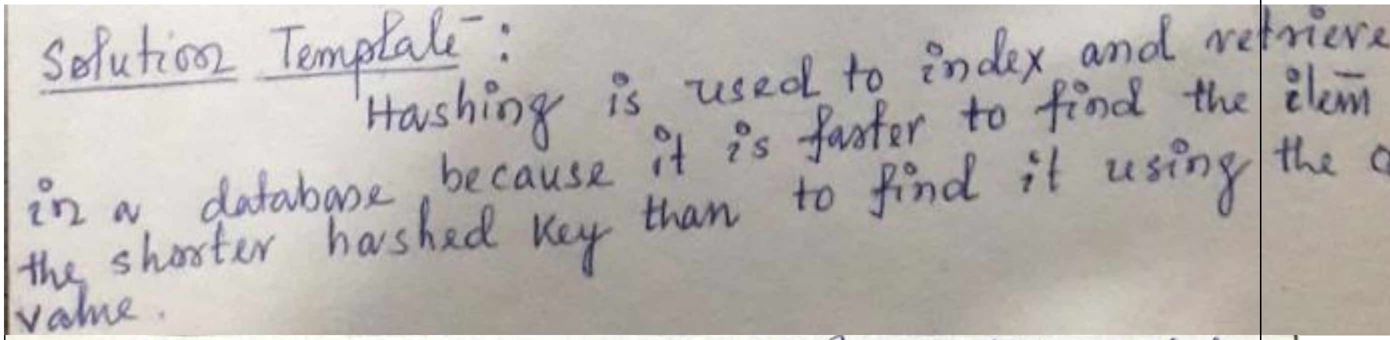
↓ ↳ Key priority no. [Note: 1: Highest, 4: Lowest]

The key element at the root of the heap will be processed first.

b **Define hashing. What is a collision in hashing? Explain with an example. Write a function to store data in the hash table, avoiding collision by using chaining- a collision resolution method.**

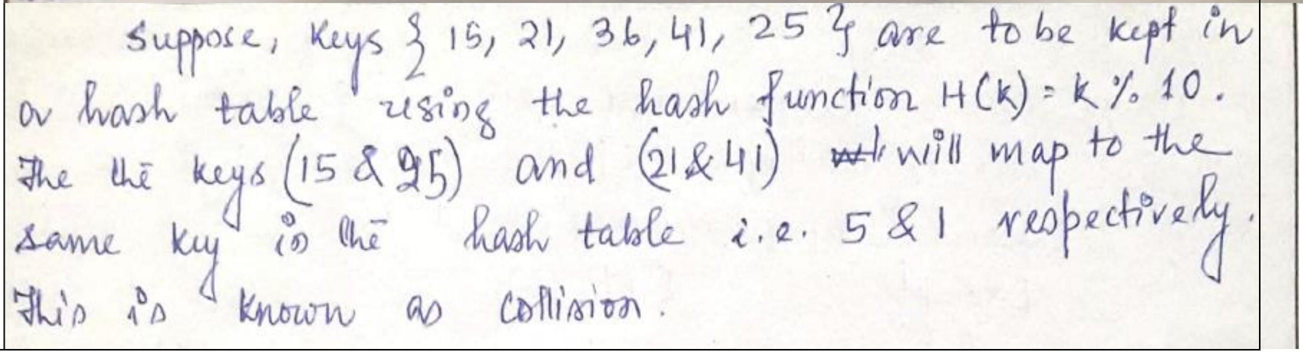
Scheme of Evaluation: Define hashing: 1 mark
Collision of hashing with example: 1.5 mark
Function for hashing: 1.5 mark

Model Solution:

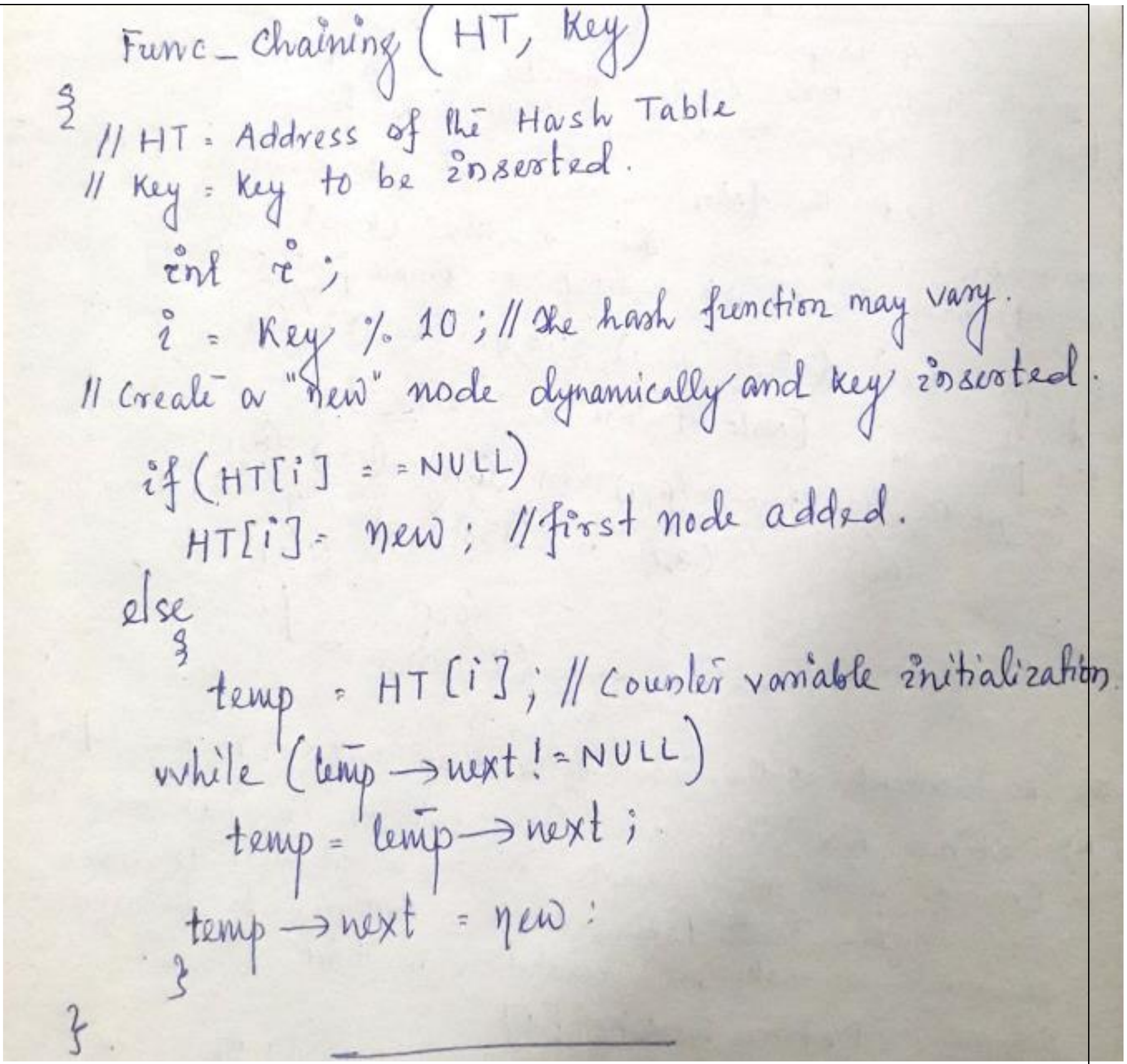


Solution Template:

Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the o value.



Suppose, Keys $\{15, 21, 36, 41, 25\}$ are to be kept in a hash table using the hash function $H(k) = k \% 10$. The the keys (15 & 25) and (21 & 41) ~~wt~~ will map to the same key in the hash table i.e. 5 & 1 respectively. This is known as collision.



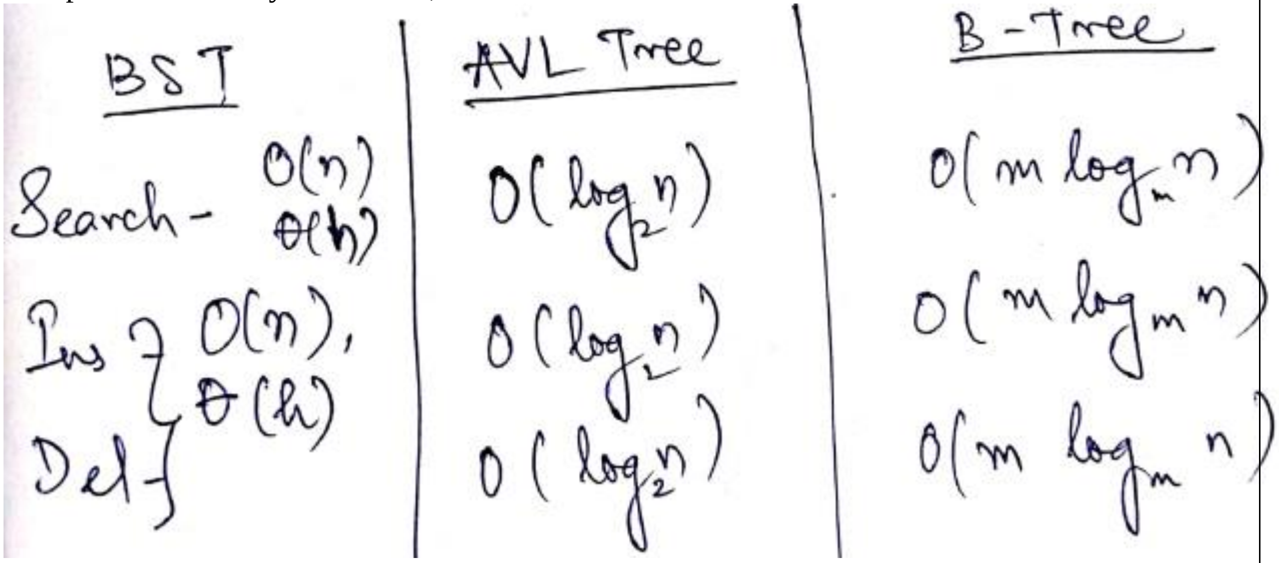
```
Func_Chaining (HT, Key)
{
 // HT : Address of the Hash Table
 // Key : Key to be inserted.
     int i ;
     i = Key % 10 ; // the hash function may vary.
 // Create a "new" node dynamically and key inserted.
     if (HT[i] = = NULL)
        HT[i] = new ; //first node added.
     else
     {
        temp = HT [i] ; // Counter variable initialization
       while (temp → next != NULL)
          temp = temp → next ;
       temp → next = new ;
     }
}
```

SECTION C

Q 4.	[4+4]
a	**For a given set of records, analyze the efficiency of processing the records by storing the data in binary search tree, AVL tree and B-tree. Construct an AVL and B-tree for the following set of values.** **15, 10, 8, 12, 14, 25, 30, 20, 18, 35** **Scheme of Evaluation :** Comparison of trees [1 Mark] Construction of AVL Tree [1.5 Marks] Construction of B-tree [1.5 Marks]

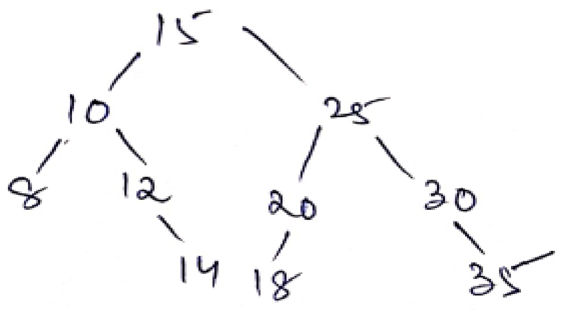
Model Solution:

Comparison of binary search tree, AVL tree and B-tree:

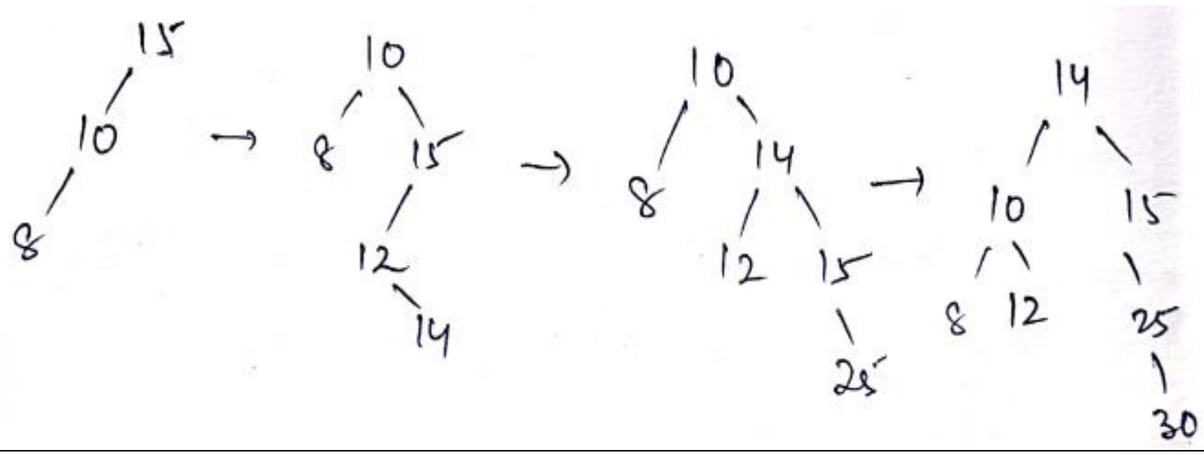


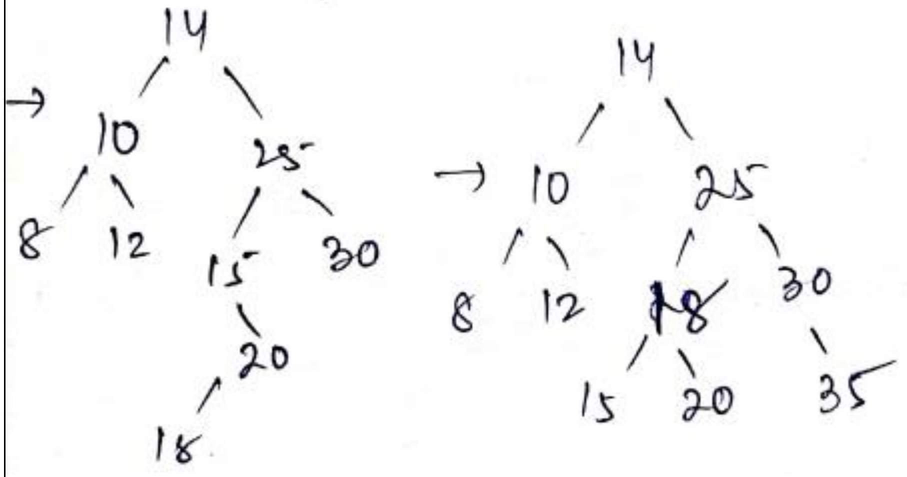
	BST	AVL Tree	B-Tree
Search -	$O(n)$ $\theta(h)$	$O(\log_2 n)$	$O(m \log_m n)$
Ins }	$O(n)$, $\theta(h)$	$O(\log_2 n)$	$O(m \log_m n)$
Del }		$O(\log_2 n)$	$O(m \log_m n)$

binary search tree:

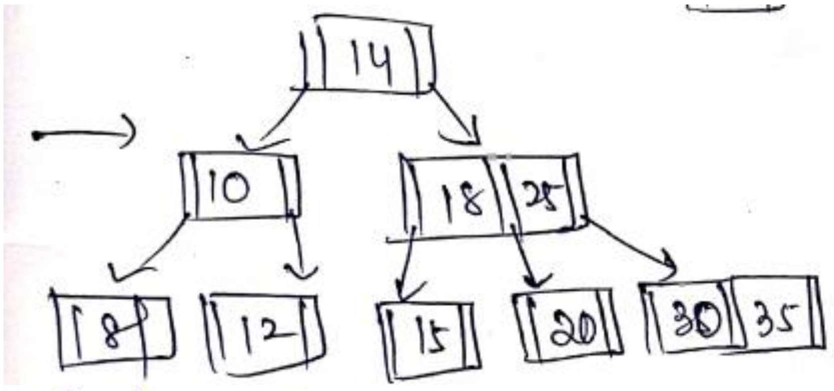


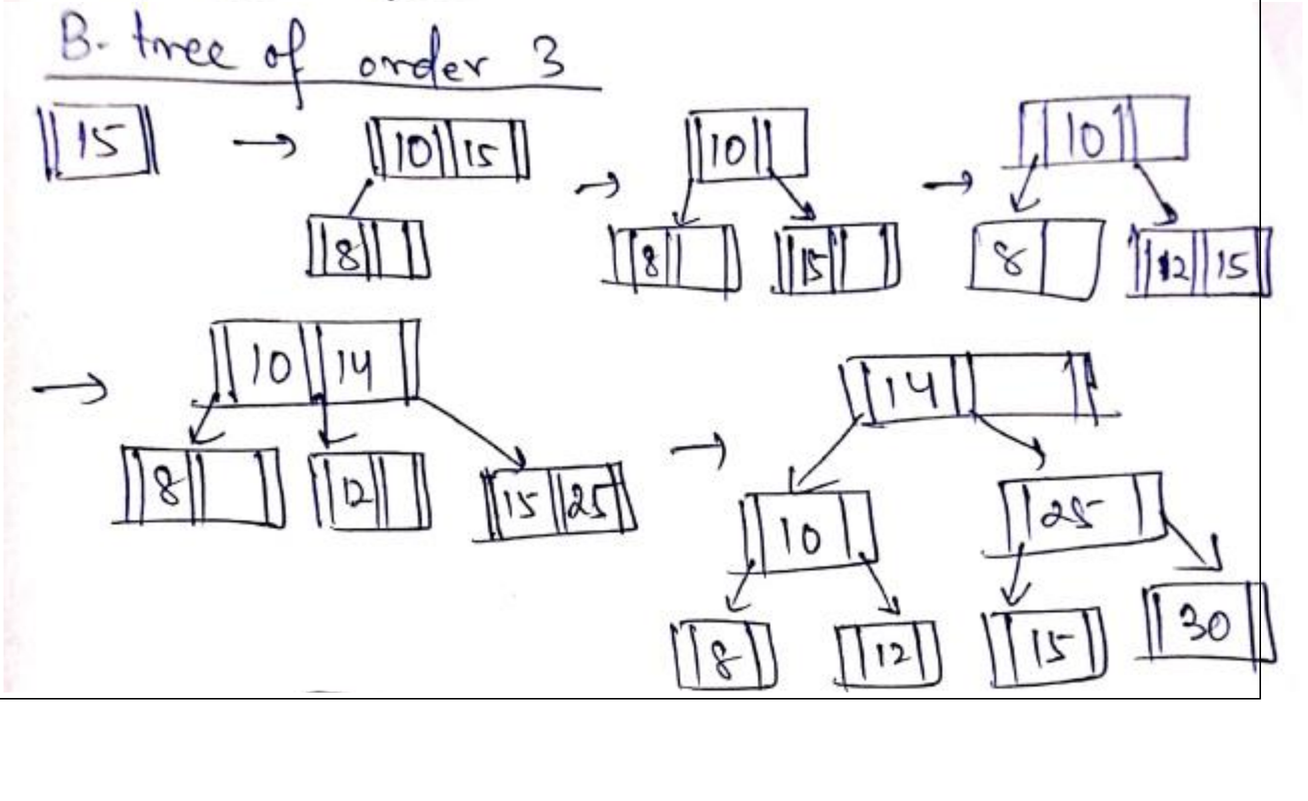
AVL tree:

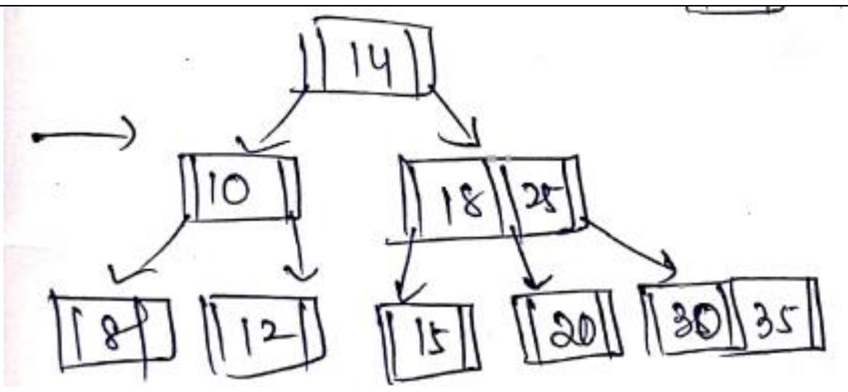



14
10
25
8
12
15
30
20
18
14
10
25
8
12
18
30
15
20
35

B-tree:


14
10
18
25
8
12
15
20
30
35


B-tree of order 3
15
10
15
8
10
8
15
10
8
12
15
10
14
8
12
15
25
14
10
25
8
12
15
30

	
b	**Write the pseudo code/function to perform the following sorting algorithms. Analyze its best case and worst case time complexity.** **i) Insertion sort** **ii) Quick Sort** **Scheme of Evaluation:** Analysis and function code of insertion sort [2 Marks] Analysis and function code of quick sort [2 Marks] **Model Solution** **Insertion Sort:**

```
void insertion_sort(int arr[], int n)
{
  int i, key, j;
  for (i = 1; i < n; i++)
  {
     key = arr[i];
     j = i-1;
     while (j >= 0 && arr[j] > key)
     {
        arr[j+1] = arr[j];
        j = j-1;
     }
     arr[j+1] = key;
  } }
```

Best case - $\Omega(n)$
Worst case – $O(n^2)$

Quick Sort:

```
quickSort (arr[], low, high)
{
   if (low < high)
   {
       pi = partition(arr, low, high);

       quickSort(arr, low, pi - 1);  // Before pi
       quickSort(arr, pi + 1, high); // After pi
   }}

partition (arr[], low, high)
{

   pivot = arr[high];

   i = (low - 1)  // Index of smaller element

   for (j = low; j <= high- 1; j++)
   {
       // If current element is smaller than the pivot
       if (arr[j] < pivot)
       {
           i++;    // increment index of smaller element
           swap arr[i] and arr[j]
       }
   }
   swap arr[i + 1] and arr[high])
   return (i + 1)
}
```

Best case - $\Omega(n \log_2 n)$
Worst case – $O(n^2)$

Q 5. [4+4]

a **Write a function to check whether the parenthesis and curly braces present in a given infix expression are in correct order or not. Use ADT for the implementation.**

Evaluation Scheme: Full mark (4) for correct answer. Step-wise mark may be awarded judiciously depending on the partial correctness of the solution.

Model Solution:

Pseudo code:
i) Traverse a string.
ii) If the current character is starting bracket ‘{‘, ‘(‘, ‘[‘ then enque it in a queue.
iii) If the current character is closing bracket ‘}’, ‘)’, ‘]’ and the front most of the queue is starting and matching bracket then deque from the queue.

iv) After complete traversal, if the queue is empty then it is balanced parentheses otherwise it is not balanced.

Function code:

```
void parenthesesBalanced( char input[], int n)
{
  char c, FLAG=0;

  If (n <=1 ) {
    display("invalid input expression")
    return;
   }
  for(int i = 0; i < n; i++) {
     c = input[i];
     if (c == '{' || c == '[' || c == '(') {
          enqueue(Q1, c); // enqueue is the ADT to insert the element
      }
      else if ((c == ']' || c == '}' || c == ')' ) {
          while(Isempty(Q1)
           {    k= dequeue(Q1);
                enqueue(Q2, k);
           }

           if((c == ']' && k != '[') ||  (c == '}' && k != '{') || (c == ')' && k == '('))
           {  FLAG=1; exit; }          deque(q);  // deque is the ADT to delete the element

           while(Isempty(Q2)
           {    k= dequeue(Q2);
                enqueue(Q1, k);
           }
           k= dequeue(Q1);
      }// end of else if
   } // end of for loop

   if (FLAG=0) // isEmpty is the ADT to check whether Q is empty or not
       display("Parentheses are balanced in the expression");
   else
       display("Parentheses are not balanced in the expression");

} //end of function
```

b **Suppose the student information i.e. <roll no, name, CGPA> are stored using binary search tree. Write a function/pseudo code to delete all the information of a student with CGPA < 5.0.**

Evaluation Scheme: Full mark for correct answer. Step-wise mark may be awarded judiciously depending on the partial correctness of the solution.

Model Solution:

```
struct BSTnode // node structure
{
    int rollNo;
    char *name;
    float CGPA;
    struct BSTnode *left, *right;
} ;

int main() // main function
{
   struct BSTnode *root … // followed by insertion code
   inorder(root);
   …
}

void inorder(struct BSTnode *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        if (root-> CGPA < 5.0)
        {
            deleteNode(root, root-> rollno);
        }
        inorder(root->right);
    }
}

void deleteNode(struct BSTnode *root, int rollNo)
{
   // base case
    if (root == NULL) return;

    if (rollNo < root-> rollNo)
        root->left = deleteNode(root->left, rollNo);
    else if (rollNo > root-> rollNo)
        root->right = deleteNode(root->right, rollNo);
    else
    {
        if (root->left == NULL) {
            struct BSTnode *temp = root->right;
            free(root);
        }
```

	```
	        else if (root->right == NULL)
	        {
	            struct BSTnode *temp = root->left;
	            free(root);
	            return temp;
	        }

	        struct BSTnode* temp = minValueNode(root->right);

	        // Copy the inorder successor's content to this node
	        root-> rollNo = temp-> rollNo;

	        // Delete the inorder successor
	        root->right = deleteNode(root->right, temp-> rollNo);
	    }
	}

	struct BSTnode* minValueNode(struct BSTnode* node)
	{
	    struct BSTnode* current = node;

	    /* loop down to find the leftmost leaf */
	    while (current->left != NULL)
	        current = current->left;

	    return current;
	}
	```
Q 6.	[4+4]
a	**Suppose the bank customer information i.e. < customer ID, customer name, customer address, account type, and account balance are stored using a header circular linked list. The header node contains the bank name to which the customer belongs to. One linked list are maintained for each bank. Write pseudo code/ function to perform the following operations:** **i) Print the customer details for a given bank** **ii) Print the customers' details with low balance (less than 5000) for a given bank.** **iii) Delete the detail of all customers with balance 0 for a given branch.** **Scheme of Evaluation**: Bit I: [1 mark] Bit II: [1 mark] Bit III: [2 mark]

Model Solution:

```
struct node
{        char cust_name[30], cust_add[50] , acc_type[10];
         float acc_balance;
         struct node * next;
};
typedef struct headernode
{
   char bankname[10];
   char branch[10];
   struct node *next;
}header;

header *start[2];
Bit I:

void show_cust(char *bankn)
{
   struct node *ptr=NULL;
   int i;
  for(i=0;i<2;i++)
   {
      if(strcmp(bankn,start[i]->bankname)==0)
      {
         ptr=start[i]->next;
         while(ptr!=NULL)
         {
            printf("\nCust name=%s\n", ptr->custname);
          printf("\nCust name=%s\n", ptr->cust_add);
          printf("\nCust name=%s\n", ptr->bank_type);
            printf("balance=%f\n", ptr->balance);
            ptr=ptr->next;
         }
         break;
      }
   }
   if(i==2)
   {
      printf("Doesnot exist");
   }
}
Bit II:
void show_custlessbal(char *bankn)
{
   struct node *ptr=NULL;
   int i;
  for(i=0;i<2;i++)
   {
      if(strcmp(bankn,start[i]->bankname)==0)
```

```
        {
           ptr=start[i]->next;
           while(ptr!=NULL)
           {
              if(ptr->balance<5000)
              {
              printf("\nCust name=%s\n", ptr->custname);
              printf("balance=%f\n", ptr->balance);
              }
              ptr=ptr->next;
           }
           break;
        }
     }

}
```

Bit III:

```
void delete_nodelessbal(char *branch)
{
   struct node *temp, *prev, *ptr;
   int i;
  for(i=0;i<2;i++)
   {
      if(strcmp(branch,start[i]->branch)==0)
      {
         ptr=start[i]->next;
         prev=start[i]->next;
         while(ptr!=NULL)
         {

            if(ptr->balance==0 && start[i]->next==ptr)
            {
               temp=ptr;
               start[i]->next=ptr->next;
               ptr=ptr->next;
               free(temp);
               //prev=start[i]->next;
            }
            else if(ptr->balance==0 && prev->next==ptr)
            {
               temp=ptr;
               prev->next=ptr->next;
               ptr=ptr->next;
               free(temp);
            }


            else
            {
               prev=ptr;
```

```
                ptr=ptr->next;
            }
}
```

b **“A graph is an acyclic tree.” (True/ False) List the applications of graph data structures. What are the different ways to represent graph ADT in memory? Write a function to represent a graph ADT using an efficient data structure.**

Scheme of Evaluation A graph is an acyclic tree- True. [1 mark]
Applications of graph data structure: [1 mark]
Representation in Memory [0.5 mark]
Function to represent a graph [1.5 mark]

Model Solution:

A graph is an acyclic tree- True. [1 mark]
Applications of graph data structure: [1 mark]

i) In OS, Resource allocation graph

ii) For Navigation System

iii) It can be used for flow of computation

iv) Design circuit connection

Different ways to represent graph ADT in memory:[0.5 mark]

- Using Array:- Adjacency Matrix and Incidence Matrix
- Using Linked List:- Adjacency List

```
void represent_graph (int V)      [1.5 mark]
{       int adj_mat[V][V], i,j;
        printf(“Enter 1 if two vertices are adjacent otherwise enter 0”);
        for( i=0;i<V;i++)
        {       for( j=0;j<V;j++)
                {       scanf(“%d”, adj_mat[i][j]);
                }
        }
}
```

SECTION D

Q 7. [4+4]

a **Suppose the mobile locations of different persons are represented by nodes in a graph. Write a pseudo code/ algorithm to find the nearest mobile phones present for a given mobile location. Then, find the subsequent neighbors' as per user's requirement.**

Scheme of Evaluation: Full mark for correct answer. Step-wise mark may be awarded judiciously depending on the partial correctness of the solution.

Model Solution:

Algorithm

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

Function: using adjacency list (adjacency matrix may also be used)
Modified BFS which prints level along with BFS output

```
void bfs(struct Graph* graph, int startVertex)
{int level=0;
 startVertex->level=level;
 struct queue* q = createQueue();
 graph->visited[startVertex] = 1;
 enqueue(q, startVertex);
 while(!isEmpty(q))
 {
  printQueue(q);
  int currentVertex = dequeue(q);
  printf("Visited %d\n", currentVertex);
  struct node* temp = graph->adjLists[currentVertex];
  level++;
  while(temp)
   {
     int adjVertex = temp->vertex;
     if(graph->visited[adjVertex] == 0)
     {
     graph->visited[adjVertex] = 1;
     adjVertex->level=level;
     enqueue(q, adjVertex);
      }
      temp = temp->next;
     }
   }
  }

struct node
{
    int vertex;
    struct node* next;
int level;
};
struct Graph
{
    int numVertices;
    struct node** adjLists;
    int* visited;

};
```

b **Two binary trees are identical if the same elements are present at the same position. Write a function to check whether two given binary trees are identical or not.**

Scheme of Evaluation: Full mark for correct answer. Step-wise mark may be awarded judiciously depending on the partial correctness of the solution.

Algorithm

```
1. If both trees are empty then return 1.

2. Else If both trees are non -empty

     (a) Check data of the root nodes (tree1->data ==  tree2->data)

     (b) Check left subtrees recursively  i.e., call sameTree( tree1->left_subtree, tree2->left_subtree)

     (c) Check right subtrees recursively  i.e., call sameTree( tree1->right_subtree, tree2->right_subtree)

     (d) If a,b and c are true then return 1.

3  Else return 0 (one is empty and other is not)
```

Recursive Function

```
struct node
{
    int data;
    struct node* left;
    struct node* right;
};
```

```
struct node* newNode(int data)
{
    struct node* node = (struct node*) malloc(sizeof(struct node));
    node->data  = data;
    node->left  = NULL;
    node->right = NULL;
            return(node);
}

int identicalTrees(struct node* a, struct node* b)
{
    if (a==NULL && b==NULL)
        return 1;
   if (a!=NULL && b!=NULL)
  {
        return( a->data == b->data && identicalTrees(a->left, b->left) &&

            identicalTrees(a->right, b->right)
        );
    }
      return 0; }
```

Q 8. [4+4]

a **Suppose a computer system has one processor to execute different tasks. Each task has a time of execution. Each task is assigned with a priority number depending upon the type of task: *Local Printing (Lowest Priority -1), Web Applications (Priority-2), I/O interfacing (Highest Priority -3)*. Every time a task is generated, its execution time and priority number are entered and stored**.

Which data structure can efficiently maintain task waiting for the processor? Write functions for insertion and deletion operations for the tasks with the following conditions.

i) A task will be processed first with minimum execution time.

ii) A task will be processed first with highest priority.

Scheme of Evaluation: Full mark for correct answer. Step-wise mark may be awarded judiciously depending on the partial correctness of the solution.

Model Solution

```
struct node{
                    int time;
                    int pri;
}

struct node data[MAX];
int heap_size;
```

Heap data structure is used for insertion and deletion operation.

```
//for insertion based on highest priority
insert(int t,int p){
        struct node tmp={t,p};
        int i=heap_size++;
        int j=(i-1)/2;
        while(i>0){
                if(tmp.pri > data[j].pri){
                        i=j;
                        j=(i-1)/2;
                }else
                        break;
        }
        data[i]=tmp;
}

//for insertion based on minimum execution time
insert(int t,int p){
        struct node tmp={t,p};
        int i=heap_size++;
        int j=(i-1)/2;
        while(i>0){
                if(tmp.time < data[j].time){
                        i=j;
                        j=(i-1)/2;
                }else
                        break;
        }
        data[i]=tmp;
}

delete(int type){
        //type==0: deletion based on minimum execution time
        //type==1: deletion based on highest priority
        struct node tmp=data[0];
        data[0]=data[heap_size-1];
        heap_size--;
        if(type==0)
                min_heapify(data,0);
        else
                max_heapify(data,0);
}
```

Note: Student can write any particular type of insertion

b **Design a solution to evaluate a postfix expression. Identify the appropriate data structure used for the evaluation. The expression contains one and two digit numbers. The possible operators are ‘+’, ‘-‘, ‘/’, ‘*’, and ‘%’. Write a function for the evaluation.**

Scheme of Evaluation: Full mark for correct answer. Step-wise mark may be awarded judiciously depending on the partial correctness of the solution.

Model Solution:

```
STACK s1;
void evaluate(int type){
        int b=pop(s1);
        int a=pop(s1);
        if(type==1)
                push(s1,a+b);
        else if(type==2)
                push(s1,a-b);
        else if(type==3)
                push(s1,a*b);
        else if(type==4)
                push(s1,a/b);
        else if(type==5)
                push(s1,a%b);
}
int post_eva(char *post){
        int i=0,k=0;
        char data[3];
        while(post[i]!='\0'){
                switch(post[i]){
                        case '+': evaluate(1); k=0; break;
                        case '-': evaluate(2); k=0; break;
                        case '*': evaluate(3); k=0; break;
                        case '/': evaluate(4); k=0; break;
                        case '%': evaluate(5); k=0; break;
                        default:  if(post[i]!=' '){
                                                data[k++]= post[i];
                                        }else if(post[i]!=' ' && k>0){
                                                data[k]='\0'
                                                push(s1,atoi(data));
                                        }

                }
                i++;
        }

        return pop(s1);
}
```