STUDENT PORTFOLIO

Name: SAI KETHAN PS
Register Number: RA2111030010059
Mail ID: ps8462@srmist.edu.in
Department: NWC
Specialization: **Cyber Security**
Semester: 3rd

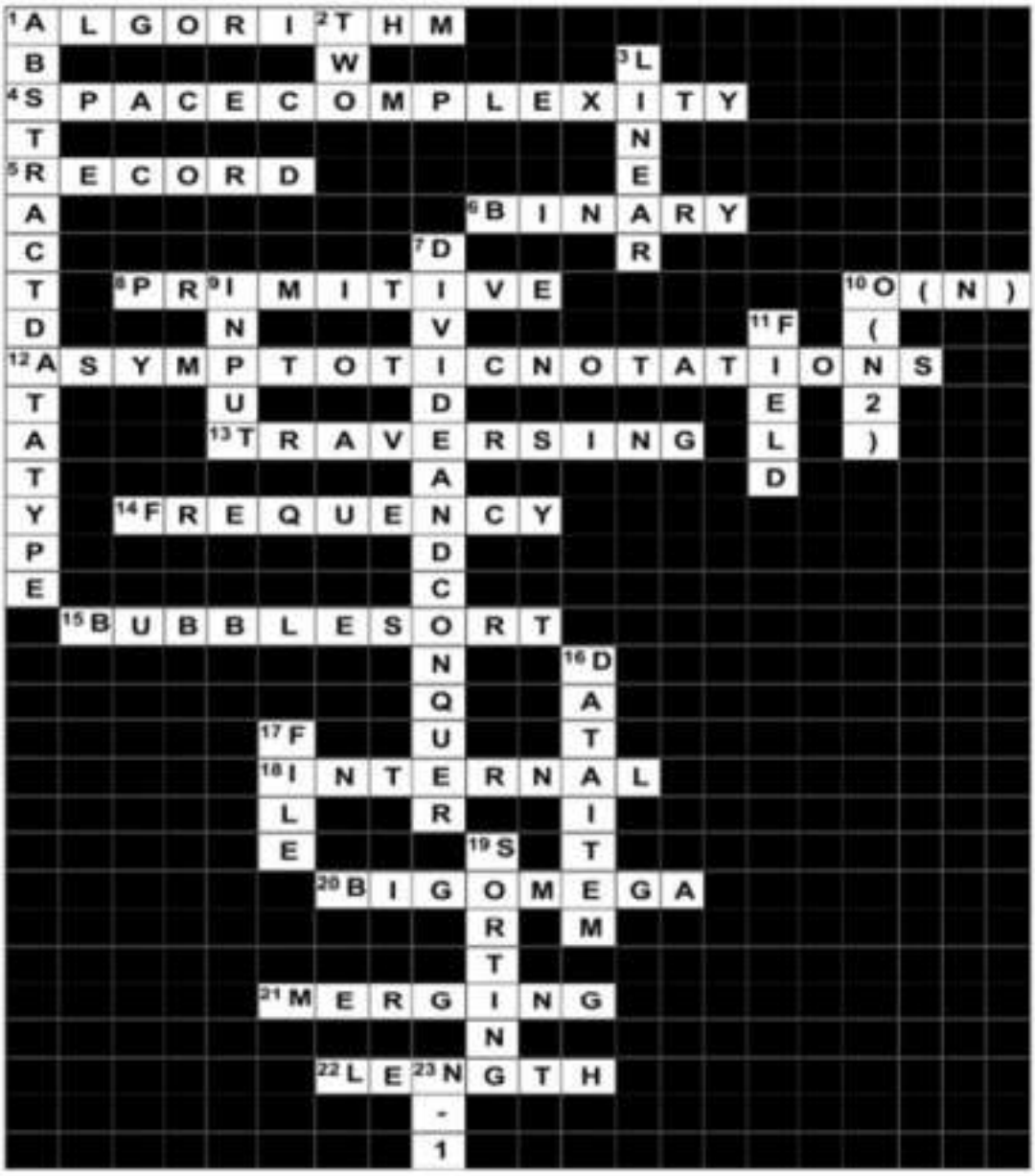
Subject Title: 18CSC201J Data Structures and Algorithms

Handled By: (**Dr.M.Jeyaselvi)**

Assignment – CrossWord Puzzle (Unit 1,2,3, & 4)
(Write about the assignment questions and how u solved differently)

UNIT -1 DATA STRUCTURES

Prepared by Dr.D.SHINY IRENE
AP/CSE/SRMIST



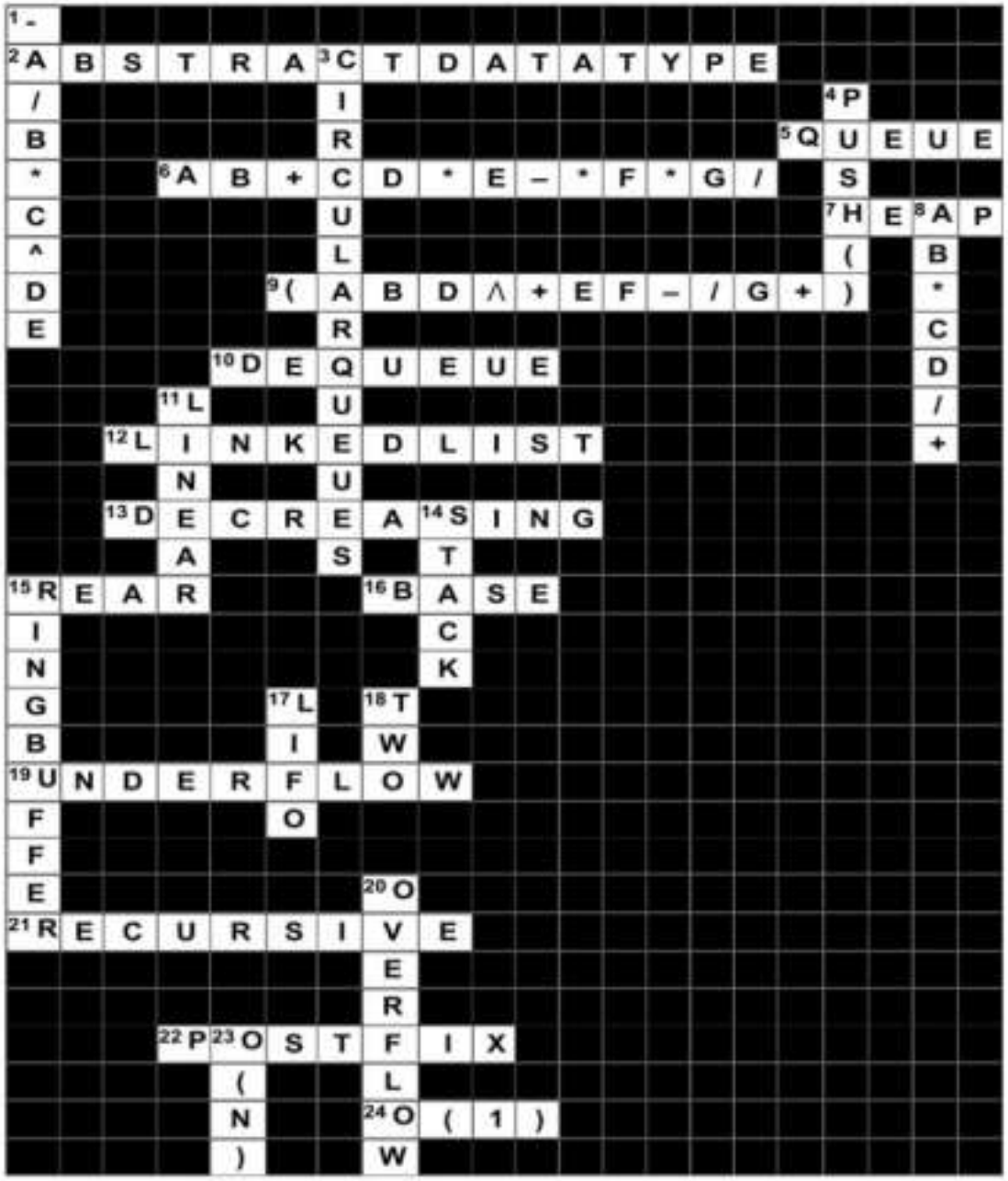
UNIT -2 DATA STRUCTURES

Prepared by Dr.D.SHINY IRENE

AP/CSE/SRMIST

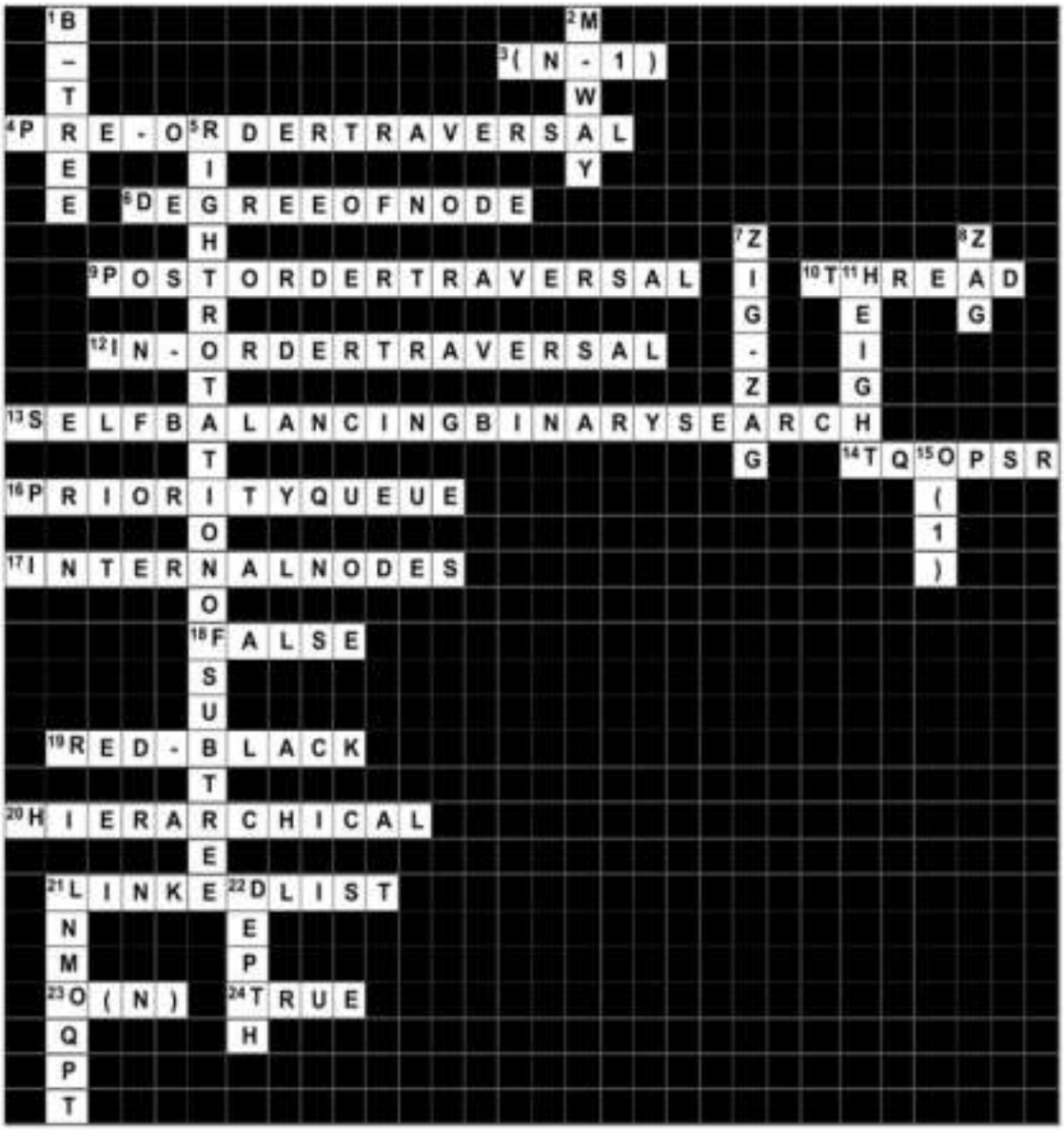
UNIT -3 DATA STRUCTURES & ALGORITHMS

Prepared by Dr.D.SHINY IRENE
AP/CSE/SRMIST



UNIT -4 DATA STRUCTURES & ALGORITHMS

Prepared by Dr.D.SHINY IRENE
AP/CSE/SRMIST



Assignment
(what is the most interesting part in the assignment)

Solving the puzzle was quite good . I was able to recall all the topics and at the same time I was not feeling bored.

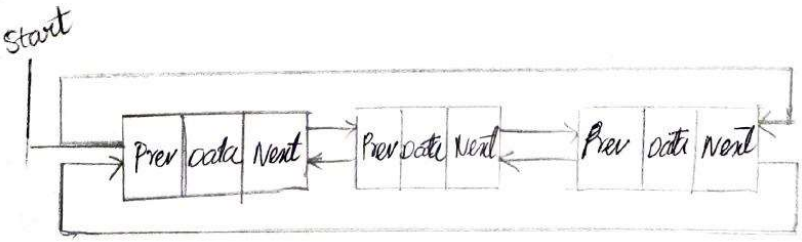
Saikethan P.S
RA2111030010005

Assignment - I

Circular doubly linked list

A circular doubly linked has the properties of both doubly linked list and circular linked list in which two consecutive elements are linked or connected by the previous and next pointer and last node points to the first node by the next pointer and last node and also the first node points to the last node by the previous pointer and also the first node points to the last node by the previous pointer
Each node consist three fields

1) Previous
2) Data
3) Next



Algorithm

1) Start
2) Create a structure node with three variables namely.
3) struct node *prev;
   int data;
   Struct node *next;
4) Create a new node
5) If list is empty assign prev and next to the newnode itself update newnode as head go to step
6) Else assign next of new_node as start node and assign prev of new node as last node goto step
7) update newnode as the last node. goto step 3 until user enters 0
8) print the elements of doubly circular linked list
9) stop.

4. Code for insertion and deletion in circular doubly linked list.

A C program to implement the circular doubly linked list

```
#include<stdio.h>
#include<stdlib.h>
Struct node
  {
    struct node *prev;
    int data;
    Struct node *next
  };
Struct node *head = NULL;
Struct node* create(int);
void insert_begin(int);
void insert_end(int);
void insert_mid(int, int);
void delete_beegin();
void delete_end();
void delete_mid();
void display();
int get_data();
```

```
int get_position();
int main()
{
  int choice;
  int data, position;
printf("\n Enter your choice : ");
Scanf ("%d", &choice);
Switch(choice){
    case 1:
     printf("\n Inserting a node at beginning");
     data = get_data();
     insert_begin(data);
      break;
 case2 :
     printf("\n Inserting a node at end");
     data = get_data();
     insert_end(data);
  case 3 :
     printf("\n Inserting a node at the given
          position");
     data = get_data();
     position = get_position();
     insert_mid(position, data);
```

```
case 4:
    printf("\n Deleting a node from beginning\n
    delete_begin();
    break;
case 5:
    printf("\n Deleting a node from end \n");
    delete_end();
    break;
case 6:
    printf("\n Delete a node from given position");
    position=get_position();
    delete_mid(position);
    break;
default:
    printf("\n Invalid choice \n");
}
printf("\n Do you want to continue ? :");
}
scanf
}
```

```
Struct node *create (int data)
    {
      Struct node* new_node = (struct node*) malloc
                (sizeof (structnode));
     if(newnode == NULL)
       {
        printf("\n can't be allocated \n");
        return NULL;
       }
     new_node → data = data;
     new-node → next = NULL;
     new-node → prev = NULL;
      return new_node;
   }
// inserting node at beginning
void insert_begin(int data)
    {
      structnode* new_node = create (data);
     if(new_node)
       {  if (head == NULL
          {
            new_node → next = new_node;
```

```
    new_node -> prev = new_node;
    head = new_node;
    return;
}
    head -> prev -> next = new_node;
    newnode -> prev = head -> prev;
    newnode -> next = head;
    head -> prev = new_node;
    head = new_node;
  }
}
// Inserting at the end
void insert_end (int data)
  {
    struct node* new_node = create (data);
    if (new_node) {
        if (head == NULL)
        {
          new_node -> next = newnode;
          new_node -> prev = new_node;
          head = new_node;
          return;
        }
```

```
head -> prev -> next = new_node;
newnode -> prev = head -> prev;
newnode -> next = head;
head -> prev = new_node;
    }
}
// inserting node at given position.
void insert_mid(int position, int data)
{
    if (position <= 0)
    {
        printf("\n Invalid position\n"); }
    else if (head == NULL && position > 1){
        printf("\n Invalid position\n");
    }
    else if (head != NULL && position > large_size()){
        printf("\n Invalid position\n");
    }
    else if( position == 1){
        insert_begin(data);
    }
```

```
else {
    struct node * new_node = create(data);
  if(new_node != NULL){
        struct node *temp = head, *prev = NULL;
        int i = 1;
    while( i <= position) {
        Prev = temp;
        temp = temp -> next;
    }
  prev -> next = new_node;
  new_node -> next = temp;
  }
 }
}
// Deleting a node at beginning
void delete_begin() {
        if(head == NULL) {
        printf("\n List is Empty \n")
        return;
    }
 }
```

```
else if (head -> next == head){
        free (head);
        head = NULL;
        return;
    }
Struct node * temp = head;
head -> prev -> next = head -> next;
head -> next -> prev = head -> prev;
head = head -> next
free (temp);
temp = NULL;
}
// Deleting a node at last
void delete_end(){
    if (head == NULL){
        printf("\n List is empty \n");
        return;
    }
    else if (head -> next == head){
        free(head);
        head = NULL;
        return;
        }
```

```
Struct node * last_node = head -> prev;
head -> prev = last_node -> prev;
free (last_node);
last_node = NULL; }

// Deleting the node from given position
void delete_mid(int position) {
    if (position <= 0 && position > list_size()) {
        printf("\n Invalid position \n");
    }
    else if(position == 1) {
        delete_begin();
    }
    else if (position == list_size()) {
        delete_end(); }
    else {
        Struct node *temp = head;
        Struct node *prev = NULL;
        int i = 1;
        while (i < position) {
            prev = temp;
            temp = temp -> next;
            i += 1; }
```

```
prev->next = temp->next;
temp->next->prev = prev;
free(temp);
temp = NULL; }
// display list
void display(){
    if(head==NULL){
        printf("\n List is empty!\n");
        return;
    }
// display list
void display(){
    if(head==NULL){
        printf("\n List is empty!\n");
        return 0;
    }
Struct node *temp = head;
do {
    printf("%d", temp->data);
    temp = temp->next; }
```

```
while(temp != head);
}
int get_data()
{
    int data;
    printf("\n Enter data : \n");
    Scanf("%d", &data);
    return data; }
int get_position(){
        int position;
        printf("Enter position : ");
        Scanf("%d", &position);
        return position;
    }
```

Advantages and disadvantages of circular double linked lis

Advantages

- Implementation of advantages data structures likes fibonacci heap.
- used with data where we have to navigate front an back
- circular doubly linked lists are used in multiprocessin
- can be traversed in both sides.

disadvantages

- Requires additional memory
- More complex than Singly linked list
- If not used properly then probly of infinite loop co occur.

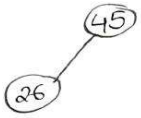
Binary Search tree
Assignment - II

P.S Saikethan
RA2111030010059

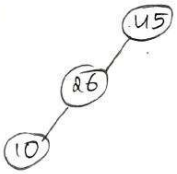
Binary Search tree

45, 26, 10, 60, 70, 30, 40

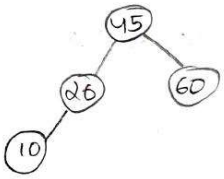
inserting 26 :



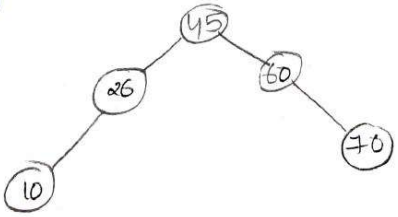
inserting 10 :



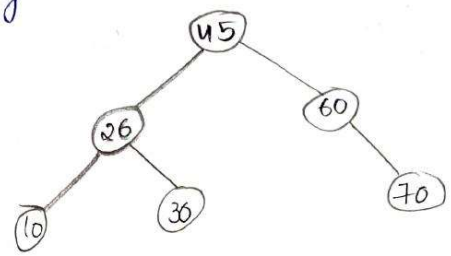
inserting 60 :



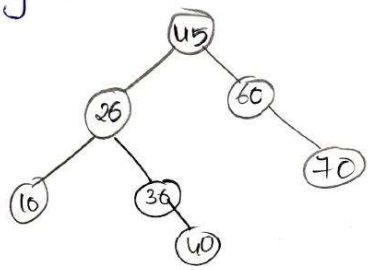
inserting 70 :



inserting 30 :

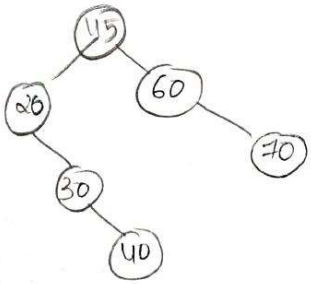


inserting 40 :

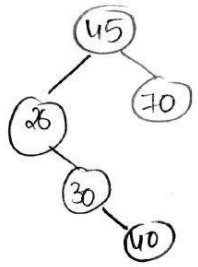


Deleting elements 10, 60, 45

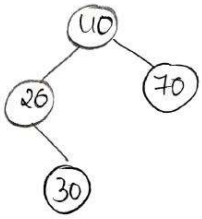
Deleting 10



Deleting 60:



Deleting 45:

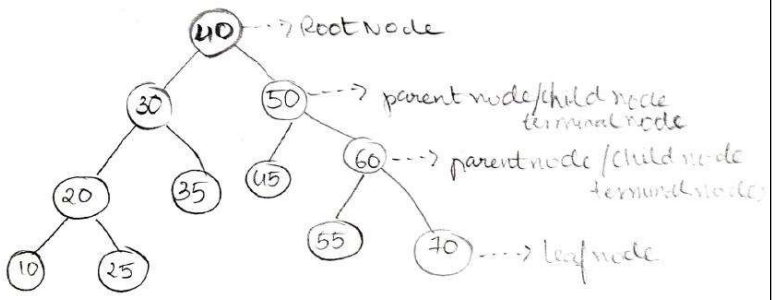


Binary Search Tree

Binary search tree is a node-based binary tree data structure which has following properties

1) The left subtree of a node contains only nodes with keys lesser than the node's key.
2) The right subtree of a node contains only nodes with keys greater than the node's key
3) The left and right subtree each must also be a binary search tree.

Graphical representation of BST



Algorithm :-

Step I :- start

Step II :- Create a new node

Step III :- If tree is empty make new node as root

Step IV :- Else compare new node with root element and the next nodes if it is greater put it in the right part else put it in the left part.

Step V :- Repeate II until user enters 0

Step VI :- End.

Code :-

```
#include <stdio.h>
#include <stdlib.h>
Struct node {
    int key;
    struct node * left, * right
    Struct newNode (int item){
        struct node * temp = (struct node*) malloc(size of
                                (struct node));
        temp → key = item
        temp → left = NULL;
        temp → right = NULL;
        return temp; }
        void inorder (struct node *root) {
            if (root != NULL)
            { inorder (root → left);
              printf ("%d", root → key);
              inorder (root → right);
            }
        }
```

```
struct node *insert (struct node * node, int key) {
if (node == NULL)
    return newNode (key);
  if (key < node -> key)
    node -> left = insert(node -> left, key);
  else
    node -> right = insert (node -> right, key);
  return node;
  }
struct node * deleteNode (struct node * root, int key ){
    if (root == NULL)
      return root;
    if (key < root -> key)
      root -> left = deleteNode (root -> left, key)
    else if ( key > root -> key)
      root -> right = deleteNode (root -> right, ke
    else {
      if (root -> left = NULL) {
        struct node * temp = root -> right
        free(root);
        return temp; }
```

```
else if (root -> right == NULL){
    struct node * temp = root -> left
    free (root);
    return temp;
  }
}
int main(){
    struct node *root = NULL;
    int n=1, data, x;
    while(n != 0)
    {scanf("%d", &data)
     root = insert(root, data)
     printf("Enter 0 to exit \n");
     scanf("%d", &n);
    }
    printf("Enter node you want to delete");
    scanf("%d", &x);
    inorder(root);
    root = deleteNode(root, x);
    inorder(root);
    return 0;
}
```

Advantages of BST

1) Binary search tree is fast in insertion and deletion when balanced.
2) We can also do range queries find keys between N and M.
3. Binary search tree is simple as compared to other data structures.

Disadvantages

1) The main disadvantage is that we should always implement a balanced binary search tree.
2) Accessing the element in BST is slightly slower than array.
3) A BST can be imbalanced or degenerated which can increase the complexity.

P.S Saikethan
RA2111030010059

Assignment - 3 Hashing

Given keys : 1, 3, 8, 10

$h(x) = 3x + 4 \mod 7$

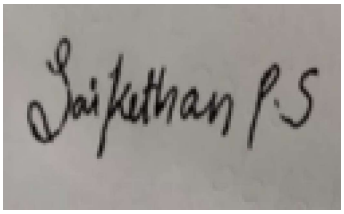
Key	Location
1	$[3(1)+4] \% 7 = 0$
3	$[3(3)+4] \% 7 = 6$
8	$[3(8)+4] \% 7 = 0$ [Put in the next availa space] =
10	$[3(10)+4] \% 7 = 6$ [Put in the next av space] = 2

0	1
1	8
2	10
3	
4	
5	
6	3

→ 1, 8, 10, -, -, -, 3

Therefore answer is B

Codechef Achievements https://www.codechef.com/users/srmcse_159 https://www.hackerrank.com/dashboard
Any other (Write if you registered or practise apart from Codechef(ex. Hackerrank, Leetcode etc.)



Signature

Note: Enclose the assignment and relevant certificates along with the profile