ASSIGNMENT 1

1. Write a program to implement a stack using an array.

Ans.
```
#include<stdio.h>
void push(char element, char stack[], int *top, int stackSize){
 if(*top == -1){
  stack[stackSize - 1] = element;
  *top = stackSize - 1;
 }
 else if(*top == 0){
  printf("The stack is already full. \n");
 }
 else{
  stack[(*top) - 1] = element;
  (*top)--;
 }
}
void pop(char stack[], int *top, int stackSize){
 if(*top == -1){
   printf("The stack is empty. \n");
 }
 else{
  printf("Element popped: %c \n", stack[(*top)]);
  // If the element popped was the last element in the stack
  // then set top to -1 to show that the stack is empty
  if((*top) == stackSize - 1){
    (*top) = -1;
  }
  else{
    (*top)++;
  }
 }
}
int main() {
  int stackSize = 4;
  char stack[stackSize];
  // A negative index shows that the stack is empty
  int top = -1;
  push('a', stack, &top, stackSize);
  printf("Element on top: %c\n", stack[top]);
  push('b',stack, &top, stackSize);
  printf("Element on top: %c\n", stack[top]);
  pop(stack, &top, stackSize);
  printf("Element on top: %c\n", stack[top]);
```

```
  pop(stack, &top, stackSize);
  printf("Top: %d\n", top);
  pop(stack, &top, stackSize);
  return 0;
```

2.Write a program to implement a queue using an array.

And./*

```
 * C Program to Implement a Queue using an Array
 */
#include <stdio.h>

#define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
   int choice;
   while (1)
   {
      printf("1.Insert element to queue \n");
      printf("2.Delete element from queue \n");
      printf("3.Display all elements of queue \n");
      printf("4.Quit \n");
      printf("Enter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
         insert();
         break;
         case 2:
         delete();
         break;
         case 3:
         display();
         break;
         case 4:
         exit(1);
         default:
```

```
            printf("Wrong choice \n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */

void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
} /* End of insert() */

void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
```

```
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
} /* End of display() *
```

3.Write a program to implement queue using linked list

Ans.

```
# Python3 program to demonstrate linked list
# based implementation of queue

# A linked list (LL) node
# to store a queue entry

class Node:



    def __init__(self, data):

        self.data = data

        self.next = None

# A class to represent a queue

# The queue, front stores the front node
# of LL and rear stores the last node of LL

class Queue:



    def __init__(self):
```

```
    self.front = self.rear = None

def isEmpty(self):
    return self.front == None

# Method to add an item to the queue
def EnQueue(self, item):
    temp = Node(item)

    if self.rear == None:
        self.front = self.rear = temp
        return
    self.rear.next = temp
    self.rear = temp

# Method to remove an item from queue
def DeQueue(self):
```

```
        if self.isEmpty():
            return
        temp = self.front
        self.front = temp.next

        if(self.front == None):
            self.rear = None
# Driver Code
if __name__== '__main__':
    q = Queue()
    q.EnQueue(10)
    q.EnQueue(20)
    q.DeQueue()
    q.DeQueue()
    q.EnQueue(30)
    q.EnQueue(40)
    q.EnQueue(50)
```

```
q.DeQueue()

print("Queue Front " + str(q.front.data))

print("Queue Rear " + str(q.rear.data))
```

OUTPUT:
Queue front:40
Queue rear:50

4.Explain the applications of Stack and Queue.
Ans.We can implement a stack and queue using both array and linked list. Stack Applications: During Function Calls and Recursive Algorithms, Expression Evaluation, Undo feature in computer keyboard, Converting an Infix to Postfix, During Depth First Search (DFS) and Backtracking Algorithms etc.
Application of the Stack
A Stack can be used for evaluating expressions consisting of operands and operators.
Stacks can be used for Backtracking, i.e., to check parenthesis matching in an expression.
It can also be used to convert one form of expression to another form.
It can be used for systematic Memory Management.
Application of Queue in Data Structure
Managing requests on a single shared resource such as CPU scheduling and disk scheduling.
Handling hardware or real-time systems interrupts.
Handling website traffic.
Routers and switches in networking.
Maintaining the playlist in media players.

5.Explain the need for a binary search tree

Ans.Binary search trees allow binary search for fast lookup, addition, and removal of data items, and can be used to implement dynamic sets and lookup tables.

Simply put, a binary search tree is a data structure that allows for fast insertion, removal, and lookup of items while offering an efficient way to iterate them in sorted order.

Binary Search is a searching algorithm for finding an element's position in a sorted array. In this approach, the element is always searched in the middle of a portion of an array. Binary search can be implemented only on a sorted list of items. If the elements are not sorted already, we need to sort them first.

6.What do you mean by binary recursion?

Ans.In binary recursion, the function calls itself twice in each run. As a result, the calculation depends on two results from two different recursive calls to itself. If we look at our Fibonacci sequence generation recursive function, we can easily find that it is a binary recursion.

Binary recursion occurs whenever there are two recursive calls for each non base case. Example is the problem to add all the numbers in an integer array A. An array A and integers i and n. Note that the value at least doubles for every other value of nk.