1. Implement deletion operation from the end of the linked list and Insertion operation from the beginning of the linked list.
2. def insertAtBeginning(self, data):

new\_node = Node(data)

new\_node.next = self.head

self.head = new\_node

def deleteNode(self, position):

if self.head == None:

return

temp\_node = self.head

if position == 0:

self.head = temp\_node.next

temp\_node = None

return

for i in range(position - 1):

temp\_node = temp\_node.next

if temp\_node is None:

break

if temp\_node is None:

return

if temp\_node.next is None:

return

next = temp\_node.next.next

temp\_node.next = None

temp\_node.next = next

2) Implement binary search using python language.

(Write a function which returns the index of x in given array arr if present, else returns -1)

1. def binary\_search(arr, low, high, x):

if high >= low:

mid = (high + low) // 2

if arr[mid] == x:

return mid

elif arr[mid] > x:

return binary\_search(arr, low, mid - 1, x)

else:

return binary\_search(arr, mid + 1, high, x)

else:

return -1

3) Write a Python program to find the middle of a linked list.

1. class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

def printMiddle(self):

slow\_ptr = self.head

fast\_ptr = self.head

if self.head is not None:

while (fast\_ptr is not None and fast\_ptr.next is not None):

fast\_ptr = fast\_ptr.next.next

slow\_ptr = slow\_ptr.next

print("The middle element is: ", slow\_ptr.data)

list1 = LinkedList()

list1.push(5)

list1.push(4)

list1.push(2)

list1.push(3)

list1.push(1)

list1.printMiddle()