**Assignment 4 | 7th January 2021**

**Question 1**

**Solution 1**

void insert\_any()

{

int data\_value, key;

printf("\nEnter data of the node: ");

scanf("%d", &data\_value);

printf("\nEnter data of the node after which new node is to be inserted: ");

scanf("%d", &key);

temp = (struct node \*) malloc(sizeof(struct node));

//Traverse till key is found or end of the linked list is reached.

ptr = header;

while(ptr->link != NULL && ptr->data != key)

{

ptr = ptr->link;

}

if(ptr->data == key)

{

temp->data = data\_value;

temp->link = ptr->link;

ptr->link = temp;

}

else

{

printf("\nValue %d not found\n",key);

}

}

**Question 2**

**Solution 2**

#include<stdio.h>

#include<stdlib.h>

void delete\_beg()

    {

        struct node \*ptr;

        if(head == NULL)

        {

            printf("\nList is empty");

        }

        else

        {

            ptr = head;

            head = ptr->next;

            free(ptr);

            printf("\n Node deleted from the begining ...");

        }

}

**Question 3**

**Solution 3**

void delete\_end() {

       if(head == NULL) {

        printf("List is empty \n");

        return;

    }

    else {

           if(head != tail ) {

            struct node \*current = head;

            while(current->next != tail) {

             current = current->next;

            }

            tail = current;

            tail->next = NULL;

        }

            else {

            head = tail = NULL;

        }

    }

}

**Question 4**

**Solution 4**

* Binary Search Algorithm is one of the widely used searching techniques.
* It can be used to sort arrays. This searching technique follows the divide and conquer strategy.
* The search space always reduces to half in every iteration.
* In each iteration, the search space is getting divided by 2.
* That means that in the current iteration you have to deal with half of the previous iteration array.

**Question 5**

**Solution 5**

Begin

   if start <= end then

      midFirst := start + (end - start) /3

      midSecond := midFirst + (end - start) / 3

      if array[midFirst] = key then

         return midFirst

      if array[midSecond] = key then

         return midSecond

      if key < array[midFirst] then

         call ternarySearch(array, start, midFirst-1, key)

      if key > array[midSecond] then

         call ternarySearch(array, midFirst+1, end, key)

      else

         call ternarySearch(array, midFirst+1, midSecond-1, key)

   else

      return invalid location

End