Question 1 answer:

// insertion at a specified position

#include <stdio.h>

#include<stdlib.h>

struct Node {

int data;

struct Node\* next;

};

int size = 0;

struct Node\* getNode(int data)

{

struct Node\* newNode = (struct Node\*)malloc(sizeof (struct Node));

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void insert\_any(struct Node\*\* current, int pos, int data)

{

if (pos < 1 || pos > size + 1)

printf ("Invalid position!\n" );

else {

while (pos--) {

if (pos == 0) {

struct Node\* temp = getNode(data);

temp->next = \*current;

\*current = temp;

}

else

current = &(\*current)->next;

}

size++;

}

}

void printList(struct Node\* head)

{

while (head != NULL) {

printf (" %d", head->data);

head = head->next;

}

printf ("\n");

}

int main()

{

int data ,pos;

struct Node\* head = NULL;

head = getNode(3);

head->next = getNode(5);

head->next->next = getNode(8);

head->next->next->next = getNode(10);

size = 4;

printf ("Linked list before insertion: ");

printList(head);

printf ("enter data to be inserted \n");

scanf ("%d",&data);

printf("enter position where to be inserted \n");

scanf ("%d",&pos);

insert\_any(&head, pos+1, data);

printf("Linked list after insertion : ");

printList(head);

return 0;

}

Question 2 answer:

void del\_beg()

{

struct node \*temp;

if (head ==NULL)

printf ("\n unable to delete because list is empty \n");

else

{temp =head;

head=temp->next;

free(temp);

}

}

Question 3 answer:

void del\_end()

{

struct node \*t1, \*t2;

if (head ==NULL)

printf ("\n unable to delete because list is empty \n");

else

{ while (t1->next!=NULL)

{

t1=head;

t2=t1;

}

t2->next =NULL;

free(t1);

}}

Question 4 answer:

Binary search is a searching algorithm that works efficiently with a sorted list.

The mechanism of binary search can be better understood by an analogy of a telephone directory. When we are searching for a particular name in a directory, we first open the directory from the middle and then decide whether to look for the name in the first part of the directory or in the second part of the directory. Again, we open some page in the middle and the whole process is repeated until we finally find the right name.s

The same mechanism is applied in the binary search.

Middle can be calculated 2 ways :

MID = (BEG + END)/2

MID =BEG+ ( END-BEG)/2

(BEG + END)/2=BEG+ ( END-BEG)/2

Note :both ways of calculating mid are accepted.

Question 5 answer:

// function for ternary search

#include <stdio.h>

int ternarySearch(int l, int r, int key, int ar[])

{

if (r >= l) {

// Find the mid1 and mid2

int mid1 = l + (r - l) / 3;

int mid2 = r - (r - l) / 3;

// Check if key is present at any mid

if (ar[mid1] == key) {

return mid1;

}

if (ar[mid2] == key) {

return mid2;

}

// Since key is not present at mid,

// check in which region it is present

// then repeat the Search operation

// in that region

if (key < ar[mid1]) {

// The key lies in between l and mid1

return ternarySearch(l, mid1 - 1, key, ar);

}

else if (key > ar[mid2]) {

// The key lies in between mid2 and r

return ternarySearch(mid2 + 1, r, key, ar);

}

else {

// The key lies in between mid1 and mid2

return ternarySearch(mid1 + 1, mid2 - 1, key, ar);

}

}

// Key not found

return -1;

}

int main()

{

int l, r, p, key;

// Get the array

// Sort the array if not sorted

int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Starting index

l = 0;

// length of array

r = 9;

// Checking for 5

// Key to be searched in the array

key = 5;

// Search the key using ternarySearch

p = ternarySearch(l, r, key, ar);

// Print the result

printf("Index of %d is %d\n", key, p);

// Checking for 50

// Key to be searched in the array

key = 50;

// Search the key using ternarySearch

p = ternarySearch(l, r, key, ar);

// Print the result

printf("Index of %d is %d", key, p);

}l