Question 1

Write a function “insert\_any()” for inserting a node at any given position of the linked list. Assume position starts at 0.

void insert\_any(int pos,int value)

{

int i=0;

struct node \*p = head;

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

temp->data = value;

temp->next = NULL;

if(pos>length()){

printf("index out of range\n");

return 0;

}

if(pos==0)

{

temp->next = head;

head = temp;

}

else{

while(i<pos-1){

p= p->next;

i++;

}

temp->next = p->next;

p->next = temp;

}

}

Question 2

Write a function “delete\_beg()” for deleting a node from the beginning of the linked list.

void delete\_beg(){

struct node\*temp;

temp = head;

head = head->next;

temp->next = NULL;

free(temp);

}

Question 3

Write a function “delete\_end()” for deleting a node from the end of the linked list

void delete\_end()

{

struct node\*temp,\*p;

temp = head;

p = temp->next;

while(p->next){

p = p->next;

temp = temp->next;

}

temp->next = NULL;

free(p);

}

Question 4

In the Binary Search algorithm, it is suggested to calculate the mid as

beg + (end - beg) / 2 instead of (beg + end) / 2. Why is it so?

we consider the two lines in a more generic setting, not related to binary search, the following observations can be made:

You are correct that the problem the second form tries to avoid is overflow, attempting to represent a number that is larger than the maximum representable number.

There is no restriction on how large the individual numbers beg and end are, so potentially they can both be larger than half of the maximum representable number. Adding them means that the intermediate result (beg+end) can overflow.

The second solution seems to eliminate the risk of overflowing, but introduces another one. If the values are signed values, their difference can again overflow (or underflow, depending on their signs). Unsigned values have no problem.

There is another solution which you didn't post:

mid = beg/2 + end/2

Question 5

Write the algorithm/function for Ternary Search.

#include <stdio.h>

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

{

if (r >= l) {

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

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

if (ar[mid1] == key) {

return mid1;

}

if (ar[mid2] == key) {

return mid2;

}

if (key < ar[mid1])

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

}

else if (key > ar[mid2]) {

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

}

else {

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

}

}

return -1;

}

// Driver code

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);

}

// C program to illustrate

// recursive approach to ternary search

#include <stdio.h>

// Function to perform Ternary Search

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;

}

// Driver code

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);

}