Q. 1. Implement Sequential and Binary Search

/\*Implement Sequential Search.\*/

#include <iostream>

using namespace std;

int sequentialSearch(int array[], int size, int key) {

for (int i = 0; i < size; i++) {

if (array[i] == key) {

return i; // return the index of the key if found

}

}

return -1; // return -1 if key is not found

}

int main() {

int array[] = {1, 2, 3, 4, 5};

int size = sizeof(array) / sizeof(array[0]);

int key = 3;

int index = sequentialSearch(array, size, key);

if (index != -1) {

cout << "Key found at index " << index << endl;

} else {

cout << "Key not found" << endl;

}

return 0;

}

/\*Implement Binary Search.\*/

#include <iostream>

using namespace std;

int binarySearch(int arr[], int n, int key) {

int left = 0, right = n - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (arr[mid] == key) {

return mid;

}

else if (arr[mid] < key) {

left = mid + 1;

}

else {

right = mid - 1;

}

}

return -1;

}

int main() {

int arr[] = {1, 2, 3, 4, 5};

int n = sizeof(arr) / sizeof(arr[0]);

int key = 3;

int index = binarySearch(arr, n, key);

if (index != -1) {

cout << "Element found at index " << index << endl;

}

else {

cout << "Element not found" << endl;

}

return 0;

}

Q. 2. Create binary tree. Find height of the tree and print leaf nodes. Find mirror image, print original

and mirror image using level-wise printing

#include <bits/stdc++.h>

using namespace std;

/\* A binary tree node has data, pointer

to left child and a pointer to right child \*/

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* newNode(int data)

{

struct Node\* node

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

node->data = data;

node->left = NULL;

node->right = NULL;

return (node);

}

void mirror(struct Node\* node)

{

if (node == NULL)

return;

else {

struct Node\* temp;

/\* do the subtrees \*/

mirror(node->left);

mirror(node->right);

/\* swap the pointers in this node \*/

temp = node->left;

node->left = node->right;

node->right = temp;

}

}

/\* Helper function to print

Inorder traversal.\*/

void inOrder(struct Node\* node)

{

if (node == NULL)

return;

inOrder(node->left);

cout << node->data << " ";

inOrder(node->right);

}

// Driver Code

int main()

{

struct Node\* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

/\* Print inorder traversal of the input tree \*/

cout << "Inorder traversal of the constructed"

<< " tree is" << endl;

inOrder(root);

/\* Convert tree to its mirror \*/

mirror(root);

/\* Print inorder traversal of the mirror tree \*/

cout << "\nInorder traversal of the mirror tree"

<< " is \n";

inOrder(root);

return 0;

}