**QUESTION:**

Write a C Program to find element index.

**EXAMPLE:**

**INPUT:**

arr[]={1,2,3,4}

ele=3

**OUTPUT:**

3

**Exp.No:1.1(A) SEARCH A ELEMENT INDEX**

**DATE:**

**AIM:**

To search a element index in a given rotated array.

**Data Structure Used** : Array

**Data Type** : integer

**Routine** : Iterative

**Time Complexity**  : O(n)

**PSEUDOCODE:**

BEGIN

mid<-0;

read n;

for i<-0 to i<-n

i++;

read arr[i];

end for

read ele;

if(arr[0]>arr[1])

mid=1;

for i<-n-1 to i<-0

i--

if(arr[i]<arr[i-1])

mid=i;

break;

end if

end for

if(arr[0]==ele)

print 0;

if(arr[0]>ele)

for i<-n-1 to i<-mid

i--;

if(arr[i]==ele)

print i;

end for

for i<-0 to i<-mid

i++;

if(arr[i]==ele)

print i;

end if

end for

END

**SOURCE PROGRAM:**

#include<stdio.h>

#include<string.h>

int main()

{

int n,ele,mid=0;

printf("enter the array size");

scanf("%d",&n);

printf("enter the value of array");

int arr[n],i;

for(i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

printf("enter the value of search element");

scanf("%d",&ele);

if(arr[0]>arr[1])

mid=1;

else

{

for(i=n-1;i>=0;i--)

{

if(arr[i]<arr[i-1])

{

mid=i;

break;

}

}

}

if(arr[0]==ele)

{

printf("founded at 0");

}

else if(arr[0]>ele)

{

for(i=n-1;i>=mid;i--)

{

if(arr[i]==ele)

{

printf("founded at %d",i);

}

}

}

else{

for(i=0;i<mid;i++){

if(arr[i]==ele) {

printf("founded at %d",i);

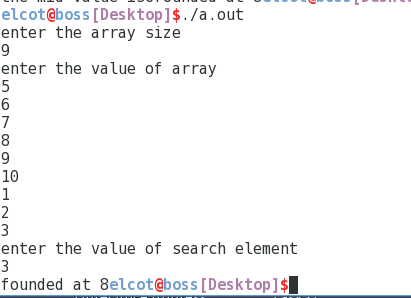
}

}

}

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a C program to find a smallest and second smallest in an array.

**EXAMPLE:**

**INPUT:**

arr[]={12,13,1,10,34,1}

**OUTPUT:**

1

10

**Exp.No:1.1(B) SMALLEST AND SECOND SMALLEST**

**DATE:**

**AIM:**

To find the smallest and second smallest element in an array.

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity :** O(n)

**PSEUDOCODE:**

BEGIN

min1<-99999999,min2<-999999;

read n;

for i<-0 to i<-n

i++;

read arr[i];

end for

for i<-0 to i<-n

i++;

if(arr[i]<min1)

min1=arr[i];

end if

if((arr[i]<min2)&&(min1!=arr[i]))

min2=arr[i];

end if

end for

print min1,min2;

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main()

{

int n,min1=99999999,min2=999999;

printf("enter the array size");

scanf("%d",&n);

printf("enter the value of array");

int arr[n],i;

for(i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

for(i=0;i<n;i++)

{

if(arr[i]<min1)

min1=arr[i];

if((arr[i]<min2)&&(min1!=arr[i]))

{

min2=arr[i];

}

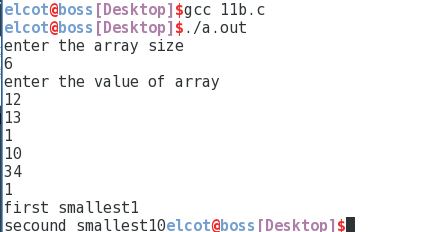
}

printf("first smallest%d\nsecound smallest%d",min1,min2);

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a C program to find no.of.1’s in the decending sorted array.

**EXAMPLE:**

**INPUT:**

arr[]={1,1,1,1,1,1,1}

**OUTPUT:**

7

**Exp.No:1.1(C) FIND THE NO.OF 1’S**

**DATE:**

**AIM:**

To write a C program to find the no.of 1’s in the given array.

**Data Structure Used :** Array

**Data Type** : integer

**Routine** : Iterative

**Time Complexity** : o(n)

**PSEUDOCODE:**

BEGIN

n,count<-0,mid<-0;

read n;

for i<-0 to i<-n

i++;

read arr[i];

end for

mid<-n/2;

if((arr[0]==1)&&(arr[n-1]==1))

print n;

end if

if((arr[mid]==1)&&(arr[mid+1]!=1))

print mid+1;

end if

for i<-0 to i<-n

i++;

if(arr[i]!=0)

count++;

print i;

end if

break;

end for

print count;

END

**SOURCE CODE:**

#include<stdio.h>

int main(){

int n,count=0,mid=0;

printf("enter the size of array");

scanf("%d",&n);

int arr[n],i;

printf("enter the value of array");

for(i=0;i<n; scanf("%d",&arr[i++]));

mid=n/2;

if((arr[0]==1)&&(arr[n-1]==1)) {

printf("the no of 1's%d\n",n);

}

else if((arr[mid]==1)&&(arr[mid+1]!=1)) {

printf("the no of 1's%d\n",mid+1);

}

else{

for(i=0;i<n;i++){

if(arr[i]!=0) {

count++;

printf("%d",i);

}

else break;

}

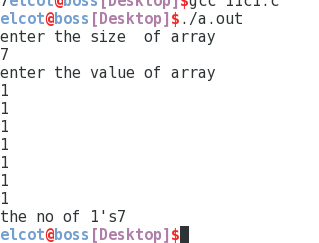
printf("the no of 1's%d\n",count);

}

return 0;

}

**OUTPUT:**

****

**RESULT:**

**QUESTION:**

To write a C program to find the pair in array whose sum is closed to element.

**EXAMPLE:**

**INPUT:**

arr[]={10,22,28,29,30,40}

ele=34

**OUTPUT:**

22

30

**Exp.No:1.1(D) PAIR OF THE SUM**

**DATE:**

**AIM:**

To write a C program to find the pair in array whose sum is closed to element

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity :** O(n2)

**PSEUDOCODE:**

BEGIN

count<-0;

read n;

int arr[n],i;

for i<-0 to i<-n

i++;

read arr[i];

end for

read ele;

\*p<-&arr[0], \*l<-&arr[n-1];

max=(\*(p)+\*(l));

while(1)

if(((\*p+\*l)>max)&&((\*p+\*l)<ele))

max=(\*(p)+\*(l));

fele=\*p;

sele=\*l;

end if

if(\*p==\*l)

p++;

l=&arr[n-1];

count++;

end if

if count==n

break;

end if

l--;

end while

print fele,sele;

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main(){

int n,ele,max,count=0,fele,sele;

printf("enter the array size");

scanf("%d",&n);

int arr[n],i;

printf("enter the value of array");

for(i=0;i<n; scanf("%d",&arr[i++]));

printf("enter the value of element");

scanf("%d",&ele);

int \*p=&arr[0];

int \*l=&arr[n-1];

max=(\*(p)+\*(l));

while(1) {

if(((\*p+\*l)>max)&&((\*p+\*l)<ele)) {

max=(\*(p)+\*(l));

fele=\*p;

sele=\*l;

}

else if(\*p==\*l) {

p++;

l=&arr[n-1];

count++;

}

else if(count==n)

break;

else

l--;

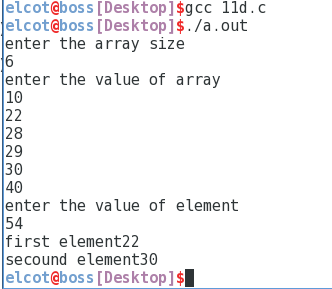
}

printf("first element%d\nsecound element%d\n",fele,sele);

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a C program to find the largest value of sum in unsorted array.

**EXAMPLE:**

**INPUT:**

arr[]={12,34,10,6,40}

**OUTPUT:**

74

**Exp.No:1.1(E) LARGEST SUM**

**DATE:**

**AIM:**

To write a C program to find largest sum value in array.

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity :** O(n)

**PSEUDOCODE:**

BEGIN

max,max1;

read n;

for i<-0 to i<-n

i++;

read arr[i];

end for

max<-arr[0];

for i<-0 to i<-n

i++;

if(max<arr[i])

max1<-max;

max<-arr[i];

end for

print max+max1;

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main()

{

int n;

printf("enter the array size");

scanf("%d",&n);

int arr[n],i,max,max1;

printf("enter the value of array");

for(i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

max=arr[0];

for(i=0;i<n;i++)

{

if(max<arr[i])

{

max1=max;

max=arr[i];

}

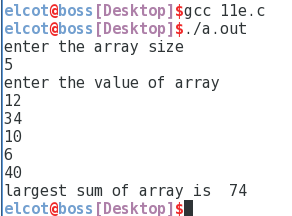
}

printf("largest sum of array is %d\n",max+max1);

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a C program to find two element sum is equal to rest element sum.

**EXAMPLE:**

**INPUT:**

arr[]={2,11,5,1,4,7}

**OUTPUT:**

11

4

**Exp.NO:1.1(F) TWO ELEMENT SUM EQUAL TO REST OF**

**DATE: ELEMENT SUM**

**AIM:**

To write a C program to find two element is equal to rest of element sum.

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity :** O(n2)

**PSEUDOCODE:**

BEGIN

sum<-0,flag<-0;

read n;

for i<-0 to i<-n

i++;

read arr[i];

end for

for i<-0 to i<-n

i++ ;

sum<-sum+arr[i];

end for

i<-0;

for j<-n-1 to j<-0

j--;

if((arr[i]+arr[j])==(sum-arr[i]-arr[j]))

print arr[i],arr[j];

flag<-1;

end if

break;

if(i==j)

i++;

j=n-1;

end if

if(i==n-1)

break;

end if

if(flag==0)

print no element;

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main(){

int n,sum=0,flag=0;

printf("enter the array size");

scanf("%d",&n);

int arr[n],i,j;

printf("enter the value of array");

for(i=0;i<n; scanf("%d",&arr[i++]));

for(i=0;i<n; sum=sum+arr[i++]);

i=0;

for(j=n-1;j>=0;j--){

if((arr[i]+arr[j])==(sum-arr[i]-arr[j])) {

printf("the number are\n%d\n%d",arr[i],arr[j]);

flag=1; break; }

if(i==j)

i++; j=n-1;

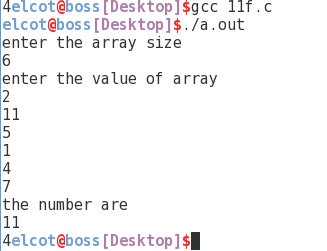
if(i==n-1)

break; }

if(flag==0)

printf("no element"); }

**OUTPUT:**

****

**RESULT:**

.

**QUESTION:**

Write a C program for find the count inversions in an array.

**EXAMPLE:**

**INPUT:**

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

**OUTPUT:**

3

**Exp.no:1.2(A) COUNT OF INVERSION**

**DATE:**

**AIM:**

To write a C program to find the count of inversion in the array.

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity :** O(n2)

**PESUDOCODE:**

BEGIN

temp<-0,n;

read n;

for i<-0 to i<-n

i++;

read arr[i];

end for

i=0;

for j<-n-1 to i<-n

j--;

if(arr[i]>arr[j]) temp++;

end if

if(j==i)

j=n-1; i++;

end if

if(i==n-1) break;

end if

end for

print temp;

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main(){

int i,j,temp=0,n;

printf("enter the size of array");

scanf("%d",&n);

int arr[n];

printf("enter the value of array");

for(i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

i=0;

for(j=n-1;i<n;j--)

{

if(arr[i]>arr[j])

{

temp++;

}

if(j==i)

{

j=n-1;

i++;

}

if(i==n-1)

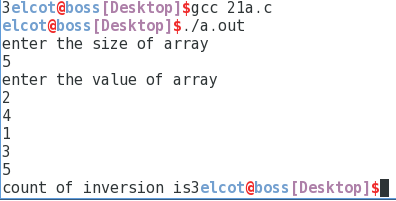
break;

}

printf("count of inversion is%d",temp);

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a C program to find given array is subset or not.

**EXAMPLE:**

**INPUT:**

arr[]={11,1,13,21,3,7}

arr[]={11,3,7,1}

**OUTPUT:**

Subset

**Exp.No:1.2(B) SUBSET(OR)NOT A SUBSET**

**DATE:**

**AIM:**

To write a C program to find the given array is subset or not a subset.

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity :** O(n2)

**PSEUDOCODE:**

BEGIN

n,n1,temp<-0;

read n;

for i<-0 to i<-n

i++;

read arr[i];

end for

read n1;

for i<-0 to i<-n1

i++;

read arr1[i];

end for

i=0;

for j<-0 to j<-n1

j++;

if(arr[j]==arr[i])

temp++;

i++;

j=0;

end if

if(i==n1)

break;

end if

end for

if(temp==n1)

print subset;

end if

print not a subset;

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main() {

int i,j,n,n1,temp=0;

printf("enter the size of array");

scanf("%d",&n);

int arr[n];

printf("enter the value of array");

for(i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

printf("enter the size of array1");

scanf("%d",&n1);

int arr1[n1];

printf("enter the value of array1");

for(i=0;i<n1;i++)

{

scanf("%d",&arr1[i]);

}

i=0;

for(j=0;j<n1;j++)

{

if(arr[j]==arr[i])

{

temp++;

i++;

j=0;

}

if(i==n1)

break;

}

if(temp==n1)

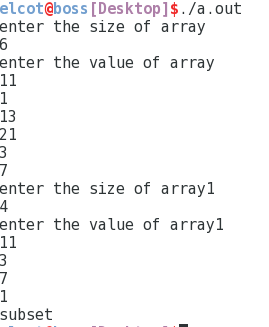
printf("subset\n");

else

printf("not a subset");

}

**OUTPUT:**

****

**RESULT:**

**QUESTION:**

Implement selection sort algorithm.

**EXAMPLE:**

**INPUT:**

arr[]={99,4,54,74,1}

**OUTPUT:**

1

4

54

74

99

**Exp.No:1.2(C) SELECTION SORT**

**DATE:**

**AIM:**

Implement the selection sort algorithm.

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity :** O(n2)

**PESUDOCODE:**

BEGIN

min,temp;

read n;

for i<-0 to i<-n read arr[i];

end for

for i<-0 to i<-n-1

i++;

min=i;

for j<-i+1 to j<-n

j++;

if(arr[j]<arr[min]) min=j;

if(i!=min)

temp=arr[i];

arr[i]=arr[min];

arr[min]=temp;

end for

for i<-0 to i<-n

i++;

print arr[i];

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main(){

int n,i,j,min,temp;

printf("enter the size of array");

scanf("%d",&n);

int arr[n];

printf("enter the value of array");

for(i=0;i<n;i++){

scanf("%d",&arr[i]);

}

for(i=0;i<n-1;i++){

min=i;

for(j=i+1;j<n;j++){

if(arr[j]<arr[min])

min=j;

}

if(i!=min) {

temp=arr[i];

arr[i]=arr[min];

arr[min]=temp;

}

}

printf("sorted array is\n");

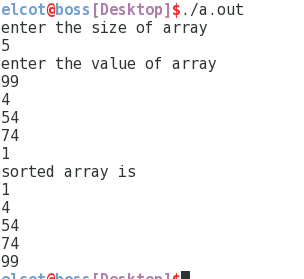
for(i=0;i<n;i++){

printf("%d\n",arr[i]);

}

return 0;

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a C program to implement the insertion sort algorithm.

**EXAMPLE:**

**INPUT:**

arr[]={12,43,45,67,8}

**OUTPUT:**

8

12

43

45

67

**Exp.No:1.2(D) INSERTION SORT**

**DATE:**

**AIM:**

To write a C program to implement the insertion sort.

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity :** O(n2)

**PSEUDOCODE:**

BEGIN

n,temp;

read n;

for i<-0 to i<-n

read arr[i];

end for

for i<-0 to i<-n

i++;

temp<-arr[i];

j<-i-1;

while((j>=0)&&(arr[j]>temp))

arr[j+1]<-arr[j];

j--;

end while

arr[j+1]<-temp;

end for

for i<-0 to i<-n

print arr[i];

end for

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main()

{

int n,i,j,temp;

printf("enter the size of array");

scanf("%d",&n);

int arr[n];

printf("enter the value of array");

for(i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

for(i=0;i<n;i++)

{

temp=arr[i];

j=i-1;

while((j>=0)&&(arr[j]>temp))

{

arr[j+1]=arr[j];

j--;

}

arr[j+1]=temp;

}

printf("sorted array\n");

for(i=0;i<n;i++)

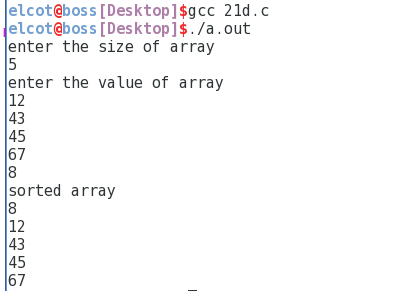
{

printf("%d\n",arr[i]);

}

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a C program to implement the sort array in wave form.

**EXAMPLE:**

**INPUT:**

arr[]={20,10,8,6,4,2}

**OUTPUT:**

20

8

10

4

6

2

**Exp.No:1.2(E) SORT THE ARRAY IN WAVE FORM**

**DATE:**

**AIM:**

To write a C program to implement the array in wave form.

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity : O**(n)

**PSEUDOCODE:**

BEGIN

n,temp;

read n;

for i<-0 to i<-n

i++;

read arr[i];

end for

for i<-0 to i<-n

i++;

if(arr[i]>arr[i+1])

if(arr[i+1]>arr[i+2])

temp<-arr[i+1];

arr[i+1]<-arr[i+2];

arr[i+2]<-temp;

end if

i++;

end if

temp<-arr[i];

arr[i]<-arr[i+1];

arr[i+1]<-temp;

i++;

end for

for i<-0 to i<-n

i++;

print arr[i];

end for

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main(){

int n,i,temp;

printf("enter the size of array");

scanf("%d",&n);

int arr[n];

printf("enter the value of array");

for(i=0;i<n;i++){

scanf("%d",&arr[i]);

}

for(i=0;i<n;i++)

{

if(arr[i]>arr[i+1])

{

if(arr[i+1]>arr[i+2])

{

temp=arr[i+1];

arr[i+1]=arr[i+2];

arr[i+2]=temp;

}

i++;

}

else

{

temp=arr[i];

arr[i]=arr[i+1];

arr[i+1]=temp;

i++;

}

}

printf("after wave sort");

for(i=0;i<n;i++)

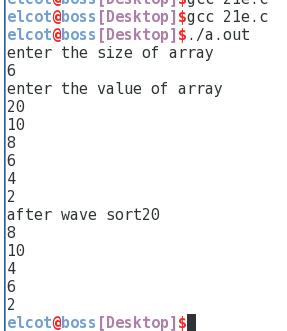
{

printf("%d\n",arr[i]);

}

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a C program to sort the almost sorted array.

**EXAMPLE:**

**INPUT:**

arr[]={99,4,54,74,1}

**OUTPUT:**

1

4

54

74

99

**Exp.No:1.2(F) SORT THE ALMOST SORTED ARRAY**

**DATE:**

**AIM:**

To write a C program to sort a almost sorted array.

**Data Structure Used :** Array

**Data Type :** integer

**Routine :** Iterative

**Time Complexity :** O(n)

**PSEUDOCODE:**

BEGIN

n,temp,k<-0;

read n;

for i<-0 to i<-n

i++;

read arr[i];

for i<-0 to i<-n

i++;

if((arr[i]<arr[i+1])&&(arr[i+1]>arr[i+2]))

swap[k++]<-i+1;

i=i+2;

end if

end for

if(arr[n-2]>arr[n-1])

swap[k++]=n-1;

end if

if(k==2)

temp<-arr[swap[0]];

arr[swap[0]]<-arr[swap[1]];

arr[swap[0]]<-arr[swap[1]];

arr[swap[1]]<-temp; end if

temp<-arr[swap[0]];

arr[swap[0]]<-arr[swap[0]+1];

arr[swap[0]+1]<-temp;

for i<-0 to i<-n

i++;

print arr[i];

end for

END

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int main()

{

int n;

printf("enter the size of array");

scanf("%d",&n);

int arr[n],i,j,k=0,temp;

printf("enter the value of array");

for(i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

int swap[n];

for(i=0;i<n;i++)

{

if((arr[i]<arr[i+1])&&(arr[i+1]>arr[i+2]))

{

swap[k++]=i+1;

i=i+2;

}

}

if(arr[n-2]>arr[n-1])

{

swap[k++]=n-1;

}

if(k==2)

{

temp=arr[swap[0]];

arr[swap[0]]=arr[swap[1]];

arr[swap[0]]=arr[swap[1]];

arr[swap[1]]=temp;

}

else

{

temp=arr[swap[0]];

arr[swap[0]]=arr[swap[0]+1];

arr[swap[0]+1]=temp;

}

printf("after sorting");

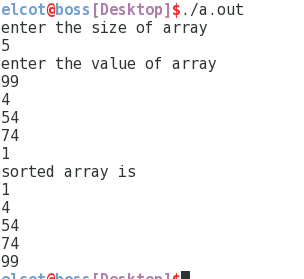
for(i=0;i<n;i++)

printf("\n%d",arr[i]);

return 0;

}

**OUTPUT:**

****

**RESULT:**

**QUESTION:**

Design an algorithm to perform insertion operation on Binary Tree and Implement it using suitable data structure

**EXAMPLE:**

**INPUT :**

15,11,8,7,10,9

**OUTPUT:**

7,11,10,15,9,8

**BINARY TREE**

**EX.NO: 2.1 (A)**

**DATE:**

**AIM:**

To perform insertion operation on Binary Tree.

**Data structure used :** binary tree

**Data type :** integer

**Routine :** iterative

**Time complexity :** O(n)

**PSEUDOCODE:**

BEGIN

void inorder(struct Node\* temp){

if (!temp)

return;

inorder(temp->left);

print ( temp->key);

inorder(temp->right);

}

void insert(struct Node\* temp, int key){

queue<struct Node\*> q;

q.push(temp);

while (!q.empty())

struct Node\* temp = q.front();

q.pop();

if (!temp->left)

temp->left = newNode(key);

break;

else

q.push(temp->left);

if (!temp->right)

temp->right = newNode(key);

break;

else

q.push(temp->right);

}

END

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int key;

struct Node\* left, \*right;

};

struct Node\* newNode(int key) {

struct Node\* temp = new Node;

temp->key = key;

temp->left = temp->right = NULL;

return temp;

};

void inorder(struct Node\* temp) {

if (!temp)

return;

inorder(temp->left);

printf("%d\t", temp->key);

inorder(temp->right);

}

void insert(struct Node\* temp, int key) {

queue<struct Node\*> q;

q.push(temp);

while (!q.empty()) {

struct Node\* temp = q.front();

q.pop();

if (!temp->left) {

temp->left = newNode(key);

break;

} else

q.push(temp->left);

if (!temp->right) {

temp->right = newNode(key);

break;

} else

q.push(temp->right);

}

}

int main()

{

struct Node\* root = newNode(10);

root->left = newNode(11);

root->left->left = newNode(7);

root->right = newNode(9);

root->right->left = newNode(15);

root->right->right = newNode(8);

printf("Inorder traversal before insertion:");

inorder(root);

int key = 12;

insert(root, key);

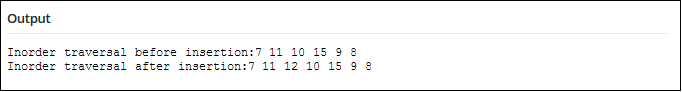
printf("Inorder traversal after insertion:");

inorder(root);

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Design an algorithm to perform insertion operation on Binary Search Tree and Implement it.

**EXAMPLE:**

**INPUT:**

50,30,70,20,40,60,80

**OUTPUT:**

20,30,40,50,60,70,80

**BINARY SEARCH TREE**

**EX.NO: 2.1 (B)**

**DATE:**

**AIM:**

To perform insertion operation on Binary Search Tree.

**Data structure used :** binary search tree

**Data type :** integer

**Routine :** iterative

**Time complexity :** O(n)

**PSEUDOCODE:**  
BEGIN

void inorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left);

print ( root->key);

inorder(root->right);

}

}

struct node\* insert(struct node\* node, int key)

{

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

return node;

}

END

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

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

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void inorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left);

printf("%d \n", root->key);

inorder(root->right);

}

}

struct node\* insert(struct node\* node, int key)

{

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

return node;

}

int main()

{

struct node \*root = NULL;

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

inorder(root);

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION**

Design an algorithm to perform deletion operation on Binary Search Tree and Implement it.

**EXAMPLE**

**INPUT:**

**50,30,70,20,40,60,80**

**20**

**OUTPUT:**

**30,40,50,60,70,80**

**BINARY SEARCH TREE DELETION**

**EX.NO: 2.1(C)**

**DATE:**

**AIM:**

To perform deletion operation on Binary Search Tree.

**Data structure used :** binary search tree

**Data type :** integer

**Routine :** iterative

**Time complexity :** O(n)

**PSEUDOCODE:**

BEGIN

void inorder(struct node \*root){

if (root != NULL){

inorder(root->left);

print ( root->key);

inorder(root->right);

}

}

struct node\* insert(struct node\* node, int key){

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else

node->right = insert(node->right, key);

return node;

}

struct node \* minValueNode(struct node\* node){

struct node\* current = node;

while (current->left != NULL)

current = current->left;

return current;

}

struct node\* deleteNode(struct node\* root, int key){

if (root == NULL) return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else{

if (root->left == NULL){

struct node \*temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL){

struct node \*temp = root->left;

free(root);

return temp;

}

struct node\* temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

END

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

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

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void inorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left);

printf("%d ", root->key);

inorder(root->right);

}

}

struct node\* insert(struct node\* node, int key)

{

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else

node->right = insert(node->right, key);

return node;

}

EX.NO:

DATE:

struct node \* minValueNode(struct node\* node)

{

struct node\* current = node;

while (current->left != NULL)

current = current->left;

return current;

}

struct node\* deleteNode(struct node\* root, int key)

{

if (root == NULL) return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else

{

if (root->left == NULL)

{

struct node \*temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL)

{

struct node \*temp = root->left;

free(root);

return temp;

}

struct node\* temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

int main()

{

struct node \*root = NULL;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

root = insert(root, 60);

root = insert(root, 80);

printf("Inorder traversal of the given tree \n");

Inorder(root);

printf("\nDelete 20\n");

root = deleteNode(root, 20);

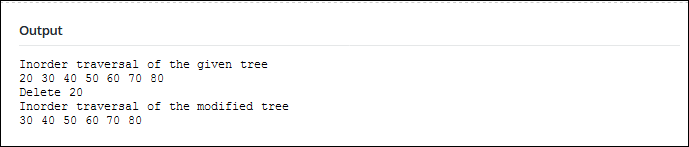
printf("Inorder traversal of the modified tree \n");

inorder(root);

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Design an algorithm to print all external / Leaf nodes in a binary tree and implement it.

**EXAMPLE:**

**INPUT:**

12,7,9,1,3,14,25,20,8,4,10

**OUTPUT:**

20,8,4,10,14,25,22

**LEAF OF THE BINARY TREE**

**EX.NO: 2.1(D)**

**DATE:**

**AIM:**

To design an algorithm to print all external / Leaf nodes in a binary tree and implement it.

**Data structure used :** binary tree

**Data type :** integer

**Routine :** iterative

**Time complexity :** O(n)

**PSEUDOCODE:**

BEGIN

void printLeaves(struct node\* root){

if ( root ){

printLeaves(root->left);

if ( !(root->left) && !(root->right) )

print( root->data);

printLeaves(root->right);

}

}

void printBoundaryLeft(struct node\* root){

if (root){

if (root->left){

print(root->data);

printBoundaryLeft(root->left);

}

else if( root->right ){

print(root->data);

printBoundaryLeft(root->right);

}

}}

void printBoundaryRight(struct node\* root){

if (root){

if ( root->right ) {

printBoundaryRight(root->right);

print(root->data);

}

else if ( root->left ) {

printBoundaryRight(root->left);

print(root->data);

}

}}

void printBoundary (struct node\* root){

if (root){

print(root->data);

printBoundaryLeft(root->left);

printLeaves(root->left);

printLeaves(root->right);

printBoundaryRight(root->right);

}

}

END

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node \*left, \*right;

};

void printLeaves(struct node\* root){

if ( root ) {

printLeaves(root->left);

if ( !(root->left) && !(root->right) )

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

printLeaves(root->right);

}

}

void printBoundaryLeft(struct node\* root){

if (root) {

if (root->left) {

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

printBoundaryLeft(root->left);

}

else if( root->right ) {

printf("%root->data);

printBoundaryLeft(root->right);

}

}}

void printBoundaryRight(struct node\* root){

if (root) {

if ( root->right ) {

printBoundaryRight(root->right);

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

}

else if ( root->left ) {

printBoundaryRight(root->left);

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

}

}

}

void printBoundary (struct node\* root){

if (root){

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

printBoundaryLeft(root->left);

printLeaves(root->left);

printLeaves(root->right);

printBoundaryRight(root->right);

}

}

struct node\* newNode( int data ){

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

temp->data = data;

temp->left = temp->right = NULL;

return temp;

}

int main(){

struct node \*root = newNode(20);

root->left = newNode(8);

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

root->left->right = newNode(12);

root->left->right->left = newNode(10);

root->left->right->right = newNode(14);

root->right = newNode(22);

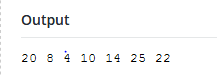
root->right->right = newNode(25);

printBoundary( root );

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Design O(logn)-time algorithm to find smallest element in a list using suitable data structure and implement it.

**EXAMPLE:**

**INPUT:**

**32,23,45,57,1**

**OUTPUT:**

**1**

**MIN ELEMENT IN BINARY SEARCH TREE**

**EX.NO: 2.1(E)**

**DATE:**

**AIM:**

To design O(logn)-time algorithm to find smallest element in a list using suitable data structure and implement it.

**Data structure used :** binary search tree

**Data type :** integer

**Routine :** iterative

**Time complexity :** O(logn)

**PSEUDOCODE:**

BEGIN

int minValue(struct node\* node) {

struct node\* current = node;

while (current->left != NULL) {

current = current->left;

}

return(current->data);

}

END

**SOURCECODE**:

#include <stdio.h>

#include<stdlib.h>

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);

}

struct node\* insert(struct node\* node, int data) {

if (node == NULL) return(newNode(data));

else{

if (data <= node->data) node->left = insert(node->left, data);

else

node->right = insert(node->right, data);

return node;

}

}

int minValue(struct node\* node) {

struct node\* current = node;

while (current->left != NULL) {

current = current->left;

}

return(current->data);

}

int main(){

struct node\* root = NULL;

root = insert(root, 4);

insert(root, 2);

insert(root, 1);

insert(root, 3);

insert(root, 6);

insert(root, 5);

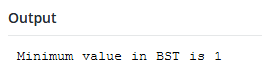
printf("\n Minimum value in BST is %d", minValue(root));

getchar();

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Design O(logn)-time algorithm to find greatest element in a list using suitable data structure and implement it.

**EXAMPLE:**

**INPUT:**

7,5,4,9,8,11

**OUTPUT:**

11

**MAX ELEMENT IN BINARY SEARCH TREE**

**EX.NO: 2.1(F)**

**DATE:**

**AIM:**

To design O(logn)-time algorithm to find greatest element in a list using suitable data structure and implement it.

**Data structure used :** binary search tree

**Data type :** integer

**Routine :** iterative

**Time complexity :** O(n)

**PSEUDOCODE:**

BEGIN

int findMax(struct Node\* root){

if (root == NULL) return INT\_MIN;

int res = root->data;

int lres = findMax(root->left);

int rres = findMax(root->right);

if (lres > res) res = lres;

if (rres > res) res = rres;

return res;

}

END

**SOURCECODE:**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

struct Node{

int data;

struct Node\* left, \*right;

};

struct Node\* newNode(int data){

struct Node\* node = (struct Node\*)

malloc(sizeof(struct Node));

node->data = data;

node->left = node->right = NULL;

return(node);

}

int findMax(struct Node\* root){

if (root == NULL)

return INT\_MIN;

int res = root->data;

int lres = findMax(root->left);

int rres = findMax(root->right);

if (lres > res)

res = lres;

if (rres > res)

res = rres;

return res;

}

int main(void){

struct Node\*NewRoot=NULL;

struct Node \*root = newNode(2);

root->left = newNode(7);

root->right = newNode(5);

root->left->right = newNode(6);

root->left->right->left=newNode(1);

root->left->right->right=newNode(11);

root->right->right=newNode(9);

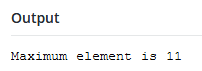
root->right->right->left=newNode(4);

printf ("Maximum element is %d \n", findMax(root));

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Give non-recursive implementation for inorder traversal in TBT and implement the same.

EXAMPLE:

INPUT:

1,2,3,4,5

OUTPUT:

4,2,5,1,3

**EX.NO: 2.2(A)**

**DATE:**

**INORDER TRAVERSAL IN THREADED BINARY SEARCH TREE**

**AIM:**

To give non-recursive implementation for inorder traversal in TBT and implement the same.

**Data Structure used**: Tree

**Data Type**: integer

**Routine**: Non-recursive

**Time complexity:**O(n^2)

**PSEUDOCODE:**

BEGIN

void printInorder(struct node\* node)

if (node == NULL) do

return;

end if

printInorder(node->left)

printf("%d ", node->data)

printInorder(node->right)

root->left = newNode(2);

root->right = newNode(3);

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

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

print("\nInorder traversal of threaded binary search tree is \n")

printInorder(root);

END

**SOURCECODE:**

#include <stdio.h>

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 printInorder(struct node\* node)

{

if (node == NULL)

return;

printInorder(node->left);

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

printInorder(node->right);

}

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);

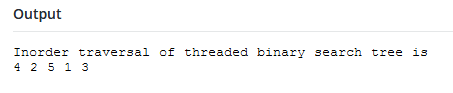
printf("\nInorder traversal of threaded binary search tree is \n");

printInorder(root);

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Given a Binary Tree, check if all leaves are at same level or not.

12

/ \

5 7

/ \

3 1

Leaves are at same level

12

/ \

5 7

/

3

Leaves are Not at same level

**EX.NO: 2.2(B)**

**DATE:**

**SAME LEAF OF A BINARY TREE**

**AIM:**

To check if all leaves are at same level or not in a binary tree.

**Data Structure used** : Binary tree

**Data Type** : integer

**Routine** : iterative

**Time complexity :** O(nlogn)

**PSEUDOCODE:**

BEGIN

int checkUtil(struct Node \*root, int level, int \*leafLevel)

{

if (root == NULL) do

return 1

if (root->left == NULL && root->right == NULL) do

if (\*leafLevel == 0) do

\*leafLevel = level;

return 1

return (level == \*leafLevel)

return checkUtil(root->left, level+1, leafLevel) and

checkUtil(root->right, level+1, leafLevel);

}

check(struct Node \*root)

return checkUtil(root, level, &leafLevel)

root->left = newNode(5);

root->left->left = newNode(3);

root->left->right = newNode(9);

root->left->left->left = newNode(1);

root->left->right->left = newNode(1);

if (check(root)) do

print("Leaves are at same level\n")

else do

print("Leaves are not at same level\n")

}

END

**SOURCECODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node{

int data;

struct Node \*left, \*right;

};

struct Node\* newNode(int data){

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

node->data = data;

node->left = node->right = NULL;

return node;

}

int checkUtil(struct Node \*root, int level, int \*leafLevel){

if (root == NULL) return 1;

if (root->left == NULL && root->right == NULL){

if (\*leafLevel == 0){

\*leafLevel = level;

return 1;

}

return (level == \*leafLevel);

}

return checkUtil(root->left, level+1, leafLevel) &&

checkUtil(root->right, level+1, leafLevel);

}

int check(struct Node \*root){

int level = 0, leafLevel = 0;

return checkUtil(root, level, &leafLevel);

}

int main(){

struct Node \*root = newNode(12);

root->left = newNode(5);

root->left->left = newNode(3);

root->left->right = newNode(9);

root->left->left->left = newNode(1);

root->left->right->left = newNode(1);

if (check(root))

printf("Leaves are at same level\n");

else

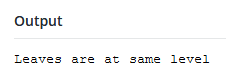
printf("Leaves are not at same level\n");

getchar();

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a function to get the maximum width of the binary tree where width of a tree is maximum of widths of all levels

**EXAMPLE:**

**INPUT:**

15,11,8,7,10,9,12,13

**OUTPUT:**

3

**EX.NO: 2.2(C)**

**DATE:**

**WIDTH OF BINARY TREE**

**AIM:**

To write a function to get the maximum width of the binary tree where width of a tree is maximum of widths of all levels.

**Data Structure used** : Binary tree

**Data Type** : integer

**Routine** : iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

BEGIN

int getMaxWidth(struct node\* root)

{

int maxWidth = 0;

int width;

int h = height(root);

int i;

for i<-1 to h

{

width = getWidth(root, i);

if(width > maxWidth)

maxWidth = width;

}

return maxWidth;

}

int getWidth(struct node\* root, int level)

{

if(root == NULL)

return 0;

if(level == 1)

return 1;

else if (level > 1)

return getWidth(root->left, level-1) +

getWidth(root->right, level-1);

}

END

**SOURCECODE:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node\* left;

struct node\* right;

};

int getWidth(struct node\* root, int level);

int height(struct node\* node);

struct node\* newNode(int data);

int getMaxWidth(struct node\* root)

{

int maxWidth = 0;

int width;

int h = height(root);

int i;

for(i=1; i<=h; i++)

{

width = getWidth(root, i);

if(width > maxWidth)

maxWidth = width;

}

return maxWidth;

}

int getWidth(struct node\* root, int level)

{

if(root == NULL)

return 0;

if(level == 1)

return 1;

else if (level > 1)

return getWidth(root->left, level-1) +

getWidth(root->right, level-1);

}

int height(struct node\* node)

{

if (node==NULL)

return 0;

else

{

int lHeight = height(node->left);

int rHeight = height(node->right);

return (lHeight > rHeight)? (lHeight+1): (rHeight+1);

}

}

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);

}

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);

root->right->right = newNode(8);

root->right->right->left = newNode(6);

root->right->right->right = newNode(7);

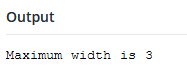
printf("Maximum width is %d \n", getMaxWidth(root));

getchar();

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write Code to Determine if Two Trees are Identical where Two trees are identical when they have same data and arrangement of data is also same.

**EXAMPLE:**

**INPUT:**

1,2,3,4

1,2,3,4

**OUTPUT:**

Identical

**EX.NO: 2.2 (D)**

**DATE:**

**IDENTICAL OF BINARY TREE**

**AIM:**

To write Code to Determine if Two Trees are Identical where Two trees are identical when they have same data and arrangement of data is also same.

**Data Structure used** :Binary tree

**Data Type** : integer

**Routine** : iterative

**Time complexity :** O(n)

**PSEUDOCODE:**

BEGIN

int identicalTrees(struct node\* a, struct node\* b)

{

if (a==NULL && b==NULL)

return 1;

if (a!=NULL && b!=NULL)

{

return

(

a->data == b->data &&

identicalTrees(a->left, b->left) &&

identicalTrees(a->right, b->right)

);

}

return 0;

}

END

**SOURCECODE:**

#include <stdio.h>

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);

}

int identicalTrees(struct node\* a, struct node\* b)

{

if (a==NULL && b==NULL)

return 1;

if (a!=NULL && b!=NULL)

{

return

(

a->data == b->data &&

identicalTrees(a->left, b->left) &&

identicalTrees(a->right, b->right)

);

}

return 0;

}

int main()

{

struct node \*root1 = newNode(1);

struct node \*root2 = newNode(1);

root1->left = newNode(2);

root1->right = newNode(3);

root1->left->left = newNode(4);

root1->left->right = newNode(5);

root2->left = newNode(2);

root2->right = newNode(3);

root2->left->left = newNode(4);

root2->left->right = newNode(5);

if(identicalTrees(root1, root2))

printf("Both tree are identical.");

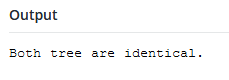
else

printf("Trees are not identical.");

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write a program to Calculate Size of a tree where size of a tree is the number of elements present in the tree.

**EXAMPLE:  
 INPUT:**

1,2,3,4,5

**OUTPUT:**

5

**EX.NO: 2.2(E)**

**DATE:**

**SIZE OF BINARY TREE**

**AIM:**

To write a program to Calculate Size of a tree where size of a tree is the number of elements present in the tree.

**Data Structure used** : Binary tree

**Data Type :** Integer

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

BEGIN

int size(struct node\* node)

{

if (node==NULL)

return 0;

else

return(size(node->left) + 1 + size(node->right));

}

END

**SOURCECODE:**

#include <stdio.h>

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);

}

int size(struct node\* node)

{

if (node==NULL)

return 0;

else

return(size(node->left) + 1 + size(node->right));

}

int main()

{

struct node \*root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

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

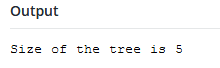
root->left->right = newNode(5);

printf("Size of the tree is %d", size(root));

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

To design and implement to reverse the alternate levels

**EXAMPLE:  
 INPUT:**

a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p

**OUTPUT:**

o,d,n,c,m,e,l,a,k,f,j,b,i,g,h

**EX.NO: 2.2(F)**

**DATE:**

**REVERSE ALTERNATE LEVEL**

**AIM**:

To design and implement to reverse the alternate levels

**Data Structure used** : Binary tree

**Data Type :** Integer

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

void storeAlternate(struct Node \*root, char arr[], int \*index, int l){

    if (root == NULL) return;

    storeAlternate(root->left, arr, index, l+1);

    if (l%2 != 0)

    {

        arr[\*index] = root->data;

        (\*index)++;

    }

    storeAlternate(root->right, arr, index, l+1);

}

void modifyTree(struct Node \*root, char arr[], int \*index, int l){

    if (root == NULL) return;

    modifyTree(root->left, arr, index, l+1);

    if (l%2 != 0) {

        root->data = arr[\*index];

        (\*index)++;

    }

    modifyTree(root->right, arr, index, l+1);

}

void reverse(char arr[], int n){

    int l = 0, r = n-1;

    while (l < r) {

        int temp = arr[l];

        arr[l] = arr[r];

        arr[r] = temp;

        l++; r--;

    }

}

void reverseAlternate(struct Node \*root){

    char \*arr = new char[MAX];

    int index = 0;

    storeAlternate(root, arr, &index, 0);

    reverse(arr, index);

    index = 0;

    modifyTree(root, arr, &index, 0);

}

void printInorder(struct Node \*root){

    if (root == NULL) return;

    printInorder(root->left);

    print (root->data);

    printInorder(root->right);

}

**SOURCECODE:**

#include<stdio.h>

#define MAX 100

struct Node{

    char data;

    struct Node \*left, \*right;

};

struct Node \*newNode(char item){

    struct Node \*temp =  newNode;

    temp->data = item;

    temp->left = temp->right = NULL;

    return temp;

}

void storeAlternate(struct Node \*root, char arr[], int \*index, int l){

    if (root == NULL) return; storeAlternate(root->left, arr, index, l+1);

    if (l%2 != 0){

        arr[\*index] = root->data;

        (\*index)++;

    }

    storeAlternate(root->right, arr, index, l+1);

}

void modifyTree(struct Node \*root, char arr[], int \*index, int l){

    if (root == NULL) return;

    modifyTree(root->left, arr, index, l+1);

    if (l%2 != 0){

        root->data = arr[\*index];

        (\*index)++;

    }

    modifyTree(root->right, arr, index, l+1);

}

void reverse(char arr[], int n){

    int l = 0, r = n-1;

    while (l < r) {

        int temp = arr[l];

        arr[l] = arr[r];

        arr[r] = temp;

        l++; r--;

    }}

void reverseAlternate(struct Node \*root){

    char \*arr = new char[MAX];

    int index = 0;

    storeAlternate(root, arr, &index, 0);

    reverse(arr, index);

    index = 0;

    modifyTree(root, arr, &index, 0);

}

void printInorder(struct Node \*root){

    if (root == NULL) return;

    printInorder(root->left);

    printf("%c",root->data);

    printInorder(root->right);

}

int main(){

    struct Node \*root = newNode('a');

    root->left = newNode('b');

    root->right = newNode('c');

    root->left->left = newNode('d');

    root->left->right = newNode('e');

    root->right->left = newNode('f');

    root->right->right = newNode('g');

    root->left->left->left = newNode('h');

    root->left->left->right = newNode('i');

    root->left->right->left = newNode('j');

    root->left->right->right = newNode('k');

    root->right->left->left = newNode('l');

    root->right->left->right = newNode('m');

    root->right->right->left = newNode('n');

    root->right->right->right = newNode('o');

    printf("Inorder Traversal of given tree\n");

    printInorder(root);

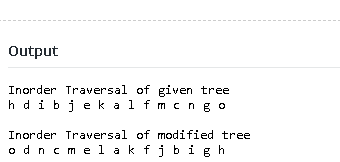
    reverseAlternate(root);

    printf("\n\nInorder Traversal of modified tree\n");

    printInorder(root);

    return 0;}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Construct winner tournament tree for {2, 6, 9, 23, 12, 78, 65, 90, 76 } and analyse the

algorithm for same.

**EXAMPLE:**

**INPUT** : 2, 6, 9, 23, 12, 78, 65, 90, 76

**OUTPUT** : 90

|  |  |
| --- | --- |
| **EXP NO: 2.3(A)** | **FINDING Kth SMALLEST ELEMENT** |
| **DATE :** |

**AIM:**

To find Kth smallest element in a list using suitable data structure and implement it.

**Data Structure used** : Binary tree

**Data Type :** Integer

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

Jflsjd;fjsdj

void minHeapify(int a[], int size, int i){

    int l = 2\*i;

    int r = 2\*i+1;

    int smallest = i;

    if(l<size && a[l]<a[smallest])

        smallest = l;

    if(r<size && a[r]<a[smallest])

        smallest = r;

    if(smallest!=i){

        swap(&a[i],&a[smallest]);

        minHeapify(a,size,smallest);

    }

}

void buildMinHeap(int a[], int size) {

    for(int i=size/2;i>=0;i--)

        minHeapify(a,size,i);

}

int kthLargest(int a[], int size, int k){

    int minHeap[k];

    int i;

    for(i=0;i<k;i++)

        minHeap[i] = a[i];

    buildMinHeap(minHeap,k);

    for(i=k;i<size;i++) {

        if(a[i]>minHeap[0]) {

            minHeap[0]=a[i];

            minHeapify(minHeap,k,0);

        }

|  |
| --- |
|  |

    }

**SOURCE CODE:**

#include<stdio.h>

void swap(int \*a, int \*b){

    \*a = \*a + \*b;

    \*b = \*a - \*b;

    \*a = \*a - \*b;

}

void minHeapify(int a[], int size, int i){

    int l = 2\*i;

    int r = 2\*i+1;

    int smallest = i;

    if(l<size && a[l]<a[smallest]) smallest = l;

    if(r<size && a[r]<a[smallest]) smallest = r;

    if(smallest!=i){

        swap(&a[i],&a[smallest]);

        minHeapify(a,size,smallest);

    }

}

void buildMinHeap(int a[], int size) {

    for(int i=size/2;i>=0;i--)

        minHeapify(a,size,i);

}

int kthLargest(int a[], int size, int k) {

    int minHeap[k];

    int i;

    for(i=0;i<k;i++)

        minHeap[i] = a[i];

    buildMinHeap(minHeap,k);

    for(i=k;i<size;i++){

        if(a[i]>minHeap[0]) {

            minHeap[0]=a[i];

            minHeapify(minHeap,k,0);

        }

    }

    return minHeap[0];

}

int a[] = {16,17,18,4,12,9,5,1};

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

    int k = 3;

    printf("%d ",kthLargest(a,size,k));

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Sort the following elements using tournament tree

{20, 14, 17, 23, 3, 9, 15, 25, 30, 34}

|  |  |
| --- | --- |
| **EXP NO:2.3(B)** | **SORTING ELEMENTS IN TOURNAMENT TREE** |
| **DATE :** |

**AIM:**

To sort the following elements using tournament tree

**Data Structure used :** Binary tree

**Data Type :** Integer

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

int buildtourn(int tourn[], int n){

int min1=0, a;

for (int i=2\*n-2; i>1; i=i-2) {

tourn[i/2] = maxi(tourn[i], tourn[i+1]);

a=mini(tourn[i], tourn[i+1]);

if (min1>a) min1=a;

}

return min1;

}

int getnext(int tourn[], int n, int low){

int i = 2;

while (i <= 2\*n-1){

if (tourn[i]>tourn[i+1]) {

tourn[i]=low;

i=2\*i;

}

else{

tourn[i+1]=low;

i=2\*(i+1);

}

}

for (i = i/2; i>1; i=i/2){

if (i%2==0) tourn[i/2]=maxi(tourn[i],tourn[i+1]); // go to the right of i

else tourn[i/2]=maxi(tourn[i], tourn[i-1]); // to the left of i

}

return 0;

}

**SOURCE CODE:**

#include<stdio.h>

int maxi(int i, int j){

if (i > j) return(i);

else return(j);

}

int mini(int i, int j){

if (i < j) return(i);

else return (j);

}

int buildtourn(int tourn[], int n){

int min1=0, a;

for (int i=2\*n-2; i>1; i=i-2){

tourn[i/2] = maxi(tourn[i], tourn[i+1]);

a=mini(tourn[i], tourn[i+1]);

if (min1>a) min1=a;

}

return min1;

}

int getnext(int tourn[], int n, int low){

int i = 2;

while (i <= 2\*n-1){

if (tourn[i]>tourn[i+1]){

tourn[i]=low;

i=2\*i;

}

else{

tourn[i+1]=low;

i=2\*(i+1);

}

}

for (i = i/2; i>1; i=i/2) {

if (i%2==0) tourn[i/2]=maxi(tourn[i],tourn[i+1]);

else tourn[i/2]=maxi(tourn[i], tourn[i-1]);

}

return 0;

}

int main() {

int tourn[100], n, i, low;

scanf("%d",&n);

for (i=n; i<=2\*n-1; i++)

scanf("%d",&tourn[i]);

low=buildtourn(tourn,n);

for(i=1; i<=n; i++){

printf("%d\t",tourn[i]);

getnext(tourn,n,low);

}

printf("\n");

return 0;

}

**OUTPUT :**



**RESULT:**

**QUESTION:**

Write a C program to find the height of winner Tournament tree and analyze the time complexity of the same.

|  |  |
| --- | --- |
| **EXP NO: 2.3(C)** | **FINDING HEIGHT OF WINNER**  **TOURNAMENT TREE** |
| **DATE :** |

**AIM:**

To find the height of winner Tournament tree and analyze the time complexity.

**Data Structure used :** Binary tree

**Data Type :** Integer

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

int maxDepth(struct node\* node)

{

if (node==NULL)

return 0;

else

{

int lDepth = maxDepth(node->left);

int rDepth = maxDepth(node->right);

if (lDepth > rDepth)

return(lDepth+1);

else return(rDepth+1);

}

}

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

struct node {

int data;

struct node\* left;

struct node\* right;

};

int maxDepth(struct node\* node) {

if (node==NULL)

return 0;

else{

int lDepth = maxDepth(node->left);

int rDepth = maxDepth(node->right);

if (lDepth > rDepth)

return(lDepth+1);

else return(rDepth+1);

}

}

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);

}

int main(){

struct node \*root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

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

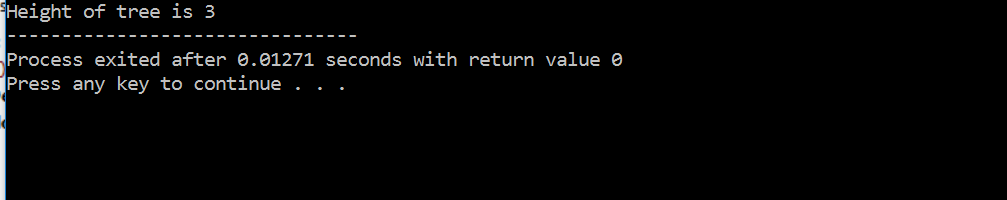
root->left->right = newNode(5);

printf("Height of tree is %d", maxDepth(root));

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Write an algorithm and implement the same to find Kth greatest element in a list. Use suitable data structure to implement it.

|  |  |
| --- | --- |
| **EXP NO: 2.3(D)** | **FINDING Kth LARGEST ELEMENT** |
| **DATE :** |

**AIM:**

To find Kth greatest element in a list using suitable data structure.

**Data Structure used :** Binary tree

**Data Type :** Integer

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

int kthlargest(int arr[], int n, int k) {

sort(arr, arr+n);

return arr[k+1];

}

**SOURCE CODE:**

#include<stdio.h>

int kthlargest(int arr[], int n, int k) {

sort(arr, arr+n);

return arr[k+1];

}

int main() {

int arr[] = {12, 3, 5, 7, 19};

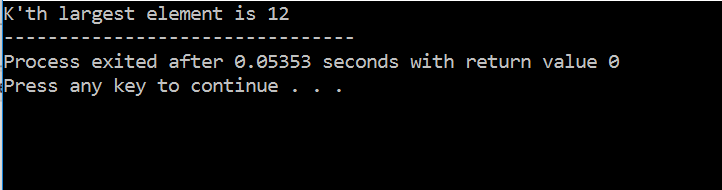
int n = sizeof(arr)/sizeof(arr[0]), k = 2;

printf("K'th largest element is %d", kthLargest(arr, n, k));

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Check if a given Binary Tree is Heap

Given a binary tree we need to check it has heap property or not, Binary tree need to fulfill following two conditions for being a heap –

1. It should be a complete tree (i.e. all levels except last should be full).
2. Every node’s value should be greater than or equal to its child node (considering max-heap)

|  |  |
| --- | --- |
| **EXP NO: 2.3(E)** | **CHECKING BINARY TREE IS HEAP** |
| **DATE :** |

**AIM:**

To check if a given Binary Tree is Heap.

**Data Structure used :** Binary tree

**Data Type :** Integer

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE**

bool isCompleteUtil (struct Node\* root, unsigned int index,

unsigned int number\_nodes)

{

if (root == NULL)

return (true);

if (index >= number\_nodes)

return (false);

return (isCompleteUtil(root->left, 2\*index + 1, number\_nodes) &&

isCompleteUtil(root->right, 2\*index + 2, number\_nodes));

}

bool isHeapUtil(struct Node\* root){

if (root->left == NULL && root->right == NULL)

return (true);

if (root->right == NULL)

return (root->key >= root->left->key);

else

if (root->key >= root->left->key &&

root->key >= root->right->key)

return ((isHeapUtil(root->left)) &&

(isHeapUtil(root->right)));

else

return (false);

}

bool isHeap(struct Node\* root){

unsigned int node\_count = countNodes(root);

unsigned int index = 0;

if (isCompleteUtil(root, index, node\_count) && isHeapUtil(root)) return true;

return false;

}

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct Node{

int key;

struct Node \*left;

struct Node \*right;

};

struct Node \*newNode(int k){

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

node->key = k;

node->right = node->left = NULL;

return node; }

unsigned int countNodes(struct Node\* root){

if (root == NULL) return (0);

return (1 + countNodes(root->left) + countNodes(root->right)); }

bool isCompleteUtil (struct Node\* root, unsigned int index,

unsigned int number\_nodes){

if (root == NULL) return (true);

if (index >= number\_nodes) return (false);

return (isCompleteUtil(root->left, 2\*index + 1, number\_nodes) &&isCompleteUtil(root->right, 2\*index + 2, number\_nodes));

}

bool isHeapUtil(struct Node\* root){

if (root->left == NULL && root->right == NULL) return (true);

if (root->right == NULL) return (root->key >= root->left->key);

else{

if (root->key >= root->left->key &&

root->key >= root->right->key)

return ((isHeapUtil(root->left)) &&

(isHeapUtil(root->right)));

else return (false);

}}

bool isHeap(struct Node\* root){

unsigned int node\_count = countNodes(root);

unsigned int index = 0;

if (isCompleteUtil(root, index, node\_count) && isHeapUtil(root)) return true;

return false; }

int main(){

struct Node\* root = NULL;

root = newNode(10);

root->left = newNode(9);

root->right = newNode(8);

root->left->left = newNode(7);

root->left->right = newNode(6);

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

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

root->left->left->left = newNode(3);

root->left->left->right = newNode(2);

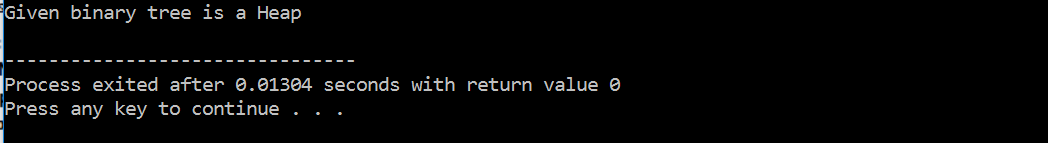
root->left->right->left = newNode(1);

if (isHeap(root)) printf("Given binary tree is a Heap\n");

else printf("Given binary tree is not a Heap\n");

return 0; }

**OUTPUT:**

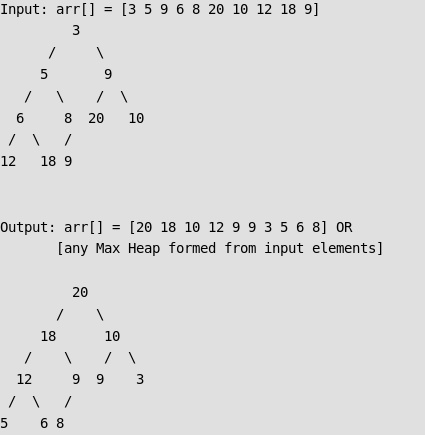


**RESULT:**

**QUESTION:**

Given array representation of min Heap, convert it to max Heap in O(n) time.

**EXAMPLES:**

****

|  |  |
| --- | --- |
| **EXP NO: 2.3(F)** | **CONVERTING MIN HEAP TO MAX HEAP** |
| **DATE :** |

**AIM:**

To convert min heap to max Heap in O(n) time.

**Data Structure used :** Binary tree

**Data Type :** Integer

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

void MaxHeapify(int arr[], int i, int n){

int l = 2\*i + 1;

int r = 2\*i + 2;

int largest = i;

if (l < n && arr[l] > arr[i]) largest = l;

if (r < n && arr[r] > arr[largest]) largest = r;

if (largest != i) {

swap(arr[i], arr[largest]);

MaxHeapify(arr, largest, n);

}}

void convertMaxHeap(int arr[], int n){

for (int i = (n-2)/2; i >= 0; --i)

MaxHeapify(arr, i, n);

}

void printArray(int\* arr, int size){

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

printf("%d ", arr[i]);

}

**SOURCE CODE:**

#include<include.h>

void MaxHeapify(int arr[], int i, int n){

int l = 2\*i + 1;

int r = 2\*i + 2;

int largest = i;

if (l < n && arr[l] > arr[i]) largest = l;

if (r < n && arr[r] > arr[largest]) largest = r;

if (largest != i) {

swap(arr[i], arr[largest]);

MaxHeapify(arr, largest, n);

}

}

void convertMaxHeap(int arr[], int n) {

for (int i = (n-2)/2; i >= 0; --i)

MaxHeapify(arr, i, n);

}

void printArray(int\* arr, int size) {

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

printf("%d ", arr[i]);

}

int main(){

int arr[] = {3, 5, 9, 6, 8, 20, 10, 12, 18, 9};

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

printf("Min Heap array : ");

printArray(arr, n);

convertMaxHeap(arr, n);

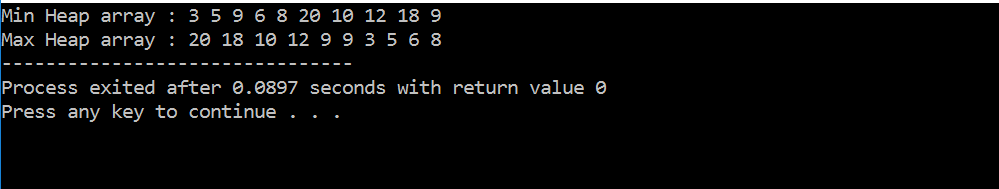
printf("\nMax Heap array : ");

printArray(arr, n);

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]. You may assume that n > m.

|  |  |
| --- | --- |
| **EXP NO: 2.4(A)** | **SEARCH FUNCTION** |
| **DATE :** |

**AIM:**

To write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]

**Data Structure used :** Array, Binary tree

**Data Type :** Integer, character

**Routine :** iterative

**Time complexity :** O(nlog n)

.

**PSEUDOCODE:**

void Search(char \*pat, char \*txt){

int M = strlen(pat);

int N = strlen(txt);

int lps[M];

computeLPSArray(pat, M, lps);

int i = 0;

int j = 0;

while (i < N) {

if (pat[j] == txt[i])

{

j++;

i++;

}

if (j == M) {

printf("Found pattern at index %d \n", (i-j)+1);

j = lps[j-1];

}

else if (i < N && pat[j] != txt[i]) {

if (j != 0)

j = lps[j-1];

else

i = i+1;

}

}

}

void computeLPSArray(char \*pat, int M, int \*lps){

int len = 0;

lps[0] = 0;

int i = 1;

while (i < M){

if (pat[i] == pat[len]) {

len++;

lps[i] = len;

i++;

}

else {

if (len != 0) {

len = lps[len-1];

}

else {

lps[i] = 0;

i++;}}

}

}

**SOURCE CODE:**

#include<string.h>

#include<stdio.h>

void computeLPSArray(char \*pat, int M, int \*lps);

void Search(char \*pat, char \*txt){

int M = strlen(pat);

int N = strlen(txt);

int lps[M];

computeLPSArray(pat, M, lps);

int i = 0;

int j = 0;

while (i < N){

if (pat[j] == txt[i]){

j++;

i++;

}

if (j == M){

printf("Found pattern at index %d \n", (i-j)+1);

j = lps[j-1];

}

else if (i < N && pat[j] != txt[i]){

if (j != 0)

j = lps[j-1];

else

i = i+1;

}

}

}

void computeLPSArray(char \*pat, int M, int \*lps){

int len = 0;

lps[0] = 0;

int i = 1;

while (i < M){

if (pat[i] == pat[len]){

len++;

lps[i] = len;

i++;

}

else

{

if (len != 0)

{

len = lps[len-1];

}

else

{

lps[i] = 0;

i++;}}

}

}

int main()

{

char \*txt = "abcdabcdabcd";

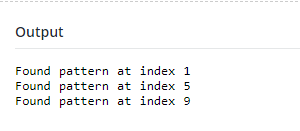
char \*pat = "a";

Search(pat, txt);

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Given a set of strings, find the longest common prefix.

|  |  |
| --- | --- |
| **EXP NO: 2.4(B)** | **FINDING LONGEST COMMON PREFIX** |
| **DATE :** |

**AIM:**

To find the longest common prefix.

**Data Structure used :** Binary tree , array

**Data Type :** Integer, character

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

string commonPrefixUtil(string str1, string str2){

string result;

int n1 = str1.length(), n2 = str2.length();

for (int i=0, j=0; i<=n1-1&&j<=n2-1; i++,j++){

if (str1[i] != str2[j])

break;

result.push\_back(str1[i]);

}

return (result);

}

string commonPrefix (string arr[], int n){

string prefix = arr[0];

for (int i=1; i<=n-1; i++)

prefix = commonPrefixUtil(prefix, arr[i]);

return (prefix);

}

**SOURCE CODE:**

#include<stdio.h>

string commonPrefixUtil(string str1, string str2){

string result;

int n1 = str1.length(), n2 = str2.length();

for (int i=0, j=0; i<=n1-1&&j<=n2-1; i++,j++){

if (str1[i] != str2[j])

break;

result.push\_back(str1[i]);

}

return (result);

}

string commonPrefix (string arr[], int n){

string prefix = arr[0];

for (int i=1; i<=n-1; i++)

prefix = commonPrefixUtil(prefix, arr[i]);

return (prefix);

}

int main(){

string arr[] = {"geeksforgeeks", "geeks",

"geek", "geezer"};

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

string ans = commonPrefix(arr, n);

if (ans.length())

printf ("The longest common prefix is : %s",

ans.c\_str());

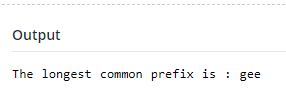
else

printf("There is no common prefix");

return (0);

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

For a given set of positive integers S let's call X a prime if there are no elements in S which are divisors of X (except X itself). You are given a set S. Find elements in it which are prime numbers for this set.

|  |  |
| --- | --- |
| **EXP NO: 2.4(C)** | **FINDING PRIME NUMBERS** |
| **DATE :** |

**AIM:**

To find elements in it which are prime numbers for this set.

**Data Structure Used** : Array

**Data Type**  : integer

**Routine** : Iterative

**Time Complexity** : O(n2)

**PSEUDOCODE:**

for(i=0;i<n;i++)

{

flag = 1;

for(j=0;j<n;j++)

{

if(i!=j && a[i]%a[j]==0)

{

flag = 0;

break;

}

}

if(flag)

printf("%d ",a[i]);

}

return 0;

}

**SOURCE CODE:**

#include <stdio.h>

int main()

{

int n;

int a[1000000];

int i;

int flag;

int j;

scanf("%d",&n);

for(i=0;i<n;i++)

scanf("%d",&a[i]);

for(i=0;i<n;i++)

{

flag = 1;

for(j=0;j<n;j++)

{

if(i!=j && a[i]%a[j]==0)

{

flag = 0;

break;

}

}

if(flag)

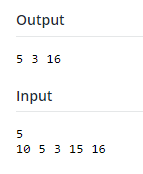
printf("%d ",a[i]);

}

return 0;

}

**OUTPUT:**



**RESULT:**

**QUESTION:**

Given a very long list of URLs find out last unique URL. Only one traversal of all URLs is allowed.

Example



|  |  |
| --- | --- |
| **EXP NO: 2.4(D)** | **FINDING OUT LAST UNQUIE URL.** |
| **DATE :** |

**AIM:**

To find out last unique URL.

**Data Structure used :** Binary tree

**Data Type :** Integer

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

void push(DLLNode\*& head\_ref, string new\_data){

DLLNode\* new\_node = new DLLNode;

new\_node->data = new\_data;

new\_node->next = (head\_ref);

new\_node->prev = NULL;

if(head\_ref != NULL) head\_ref->prev = new\_node;

head\_ref = new\_node;

}

void deleteNode(DLLNode\*& head\_ref, DLLNode\* del){

if (head\_ref == NULL || del == NULL) return;

if (head\_ref == del) head\_ref = del->next;

if (del->next != NULL) del->next->prev = del->prev;

if (del->prev != NULL) del->prev->next = del->next;

free(del);

return;

}

TrieNode\* getNewTrieNode(void){

TrieNode\* pNode = new TrieNode;

if (pNode){

pNode->isLeaf = false;

for (int i = 0; i < ALPHABET\_SIZE; i++)

pNode->children[i] = NULL;

pNode->LLptr = NULL;

}

return pNode;

}

void insert(TrieNode\* root, string key, DLLNode\*& head){

int index;

TrieNode\* pCrawl = root;

for (int level = 0; level < key.length(); level++){

index = int(key[level]);

if (!pCrawl->children[index])

pCrawl->children[index] = getNewTrieNode();

pCrawl = pCrawl->children[index];

}

if (pCrawl->isLeaf){

if (pCrawl->LLptr)

deleteNode(head, pCrawl->LLptr);

pCrawl->LLptr = NULL;

}

else{

pCrawl->isLeaf = true;

push(head, key);

pCrawl->LLptr = head;

}

}

**SOURCE CODE:**

#include <stdio.h>

const int ALPHABET\_SIZE = 256;

struct DLLNode{

string data;

DLLNode\* next, \* prev;

};

struct TrieNode{

TrieNode\* children[ALPHABET\_SIZE];

bool isLeaf;

DLLNode\* LLptr;

};

void push(DLLNode\*& head\_ref, string new\_data){

DLLNode\* new\_node = new DLLNode;

new\_node->data = new\_data;

new\_node->next = (head\_ref);

new\_node->prev = NULL;

if(head\_ref != NULL)

head\_ref->prev = new\_node;

head\_ref = new\_node;

}

void deleteNode(DLLNode\*& head\_ref, DLLNode\* del){

if (head\_ref == NULL || del == NULL) return;

if (head\_ref == del) head\_ref = del->next;

if (del->next != NULL) del->next->prev = del->prev;

if (del->prev != NULL) del->prev->next = del->next;

free(del);

return;

}

TrieNode\* getNewTrieNode(void)

{

TrieNode\* pNode = new TrieNode;

if (pNode){

pNode->isLeaf = false;

for (int i = 0; i < ALPHABET\_SIZE; i++)

pNode->children[i] = NULL;

pNode->LLptr = NULL;

}

return pNode;

}

void insert(TrieNode\* root, string key, DLLNode\*& head){

int index;

TrieNode\* pCrawl = root;

for (int level = 0; level < key.length(); level++){

index = int(key[level]);

if (!pCrawl->children[index])

pCrawl->children[index] = getNewTrieNode();

pCrawl = pCrawl->children[index];

}

if (pCrawl->isLeaf){

if (pCrawl->LLptr)

deleteNode(head, pCrawl->LLptr);

pCrawl->LLptr = NULL;

}

else{

pCrawl->isLeaf = true;

push(head, key);

pCrawl->LLptr = head;

}}

int main(){

string urls[] = {

"http://www.geeksforgeeks.org",

"http://www.contribute.geeksforgeeks.org",

"http://quiz.geeksforgeeks.org",

"http://qa.geeksforgeeks.org",

"http://practice.geeksforgeeks.org",

"http://code.geeksforgeeks.org",

"http://quiz.geeksforgeeks.org",

"http://practice.geeksforgeeks.org",

"http://code.geeksforgeeks.org",

"http://quiz.geeksforgeeks.org",

"http://qa.geeksforgeeks.org",

"http://practice.geeksforgeeks.org"

};

TrieNode\* root = getNewTrieNode();

DLLNode\* head = NULL;

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

for (int i = 0; i < n; i++)

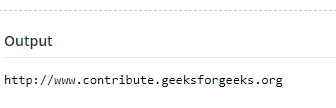
insert(root, urls[i], head);

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

return 0;

}

**OUTPUT:**

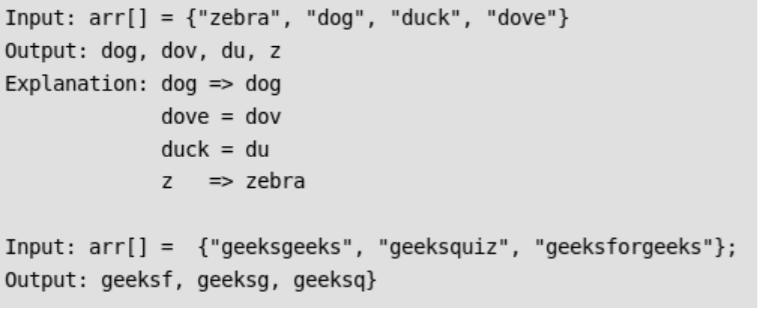


**RESULT:**

**QUESTION:**

Given an array of words, find all shortest unique prefixes to represent each word in the given array. Assume that no word is prefix of another.

Example:



|  |  |
| --- | --- |
| **EXP NO: 2.4(E)** | **FINDING SHORTEST UNIQUE PREFIXES** |
| **DATE :** |

**AIM:**

To find all shortest unique prefixes to represent each word in the given

**Data Structure used :** Binary tree ,array

**Data Type :** Integer,character

**Routine :** iterative

**Time complexity :** O(log n)

**PSEUDOCODE:**

void insert(struct trieNode \*root, string str)

{

int len = str.length();

struct trieNode \*pCrawl = root;

for (int level = 0; level<len; level++)

{

int index = str[level];

if (!pCrawl->child[index])

pCrawl->child[index] = newTrieNode();

else

(pCrawl->child[index]->freq)++;

pCrawl = pCrawl->child[index];

}

}

void findPrefixesUtil(struct trieNode \*root, char prefix[],

int ind)

{

if (root == NULL)

return;

if (root->freq == 1)

{

prefix[ind] = '\0';

printf("%s\t",prefix);

return;

}

for (int i=0; i<MAX; i++)

{

if (root->child[i] != NULL)

{

prefix[ind] = i;

findPrefixesUtil(root->child[i], prefix, ind+1);

}

}

}

void findPrefixes(string arr[], int n)

{

struct trieNode \*root = newTrieNode();

root->freq = 0;

for (int i = 0; i<n; i++)

insert(root, arr[i]);

char prefix[MAX\_WORD\_LEN];

findPrefixesUtil(root, prefix, 0);

}

**SOURCE CODE:**

#include<stdio.h>

#define MAX 256

#define MAX\_WORD\_LEN 500

struct trieNode

{

struct trieNode \*child[MAX];

int freq;

};

struct trieNode \*newTrieNode(void)

{

struct trieNode \*newNode = new trieNode;

newNode->freq = 1;

for (int i = 0; i<MAX; i++)

newNode->child[i] = NULL;

return newNode;

}

void insert(struct trieNode \*root, string str)

{

int len = str.length();

struct trieNode \*pCrawl = root;

for (int level = 0; level<len; level++)

{

int index = str[level];

if (!pCrawl->child[index])

pCrawl->child[index] = newTrieNode();

else

(pCrawl->child[index]->freq)++;

pCrawl = pCrawl->child[index];

}

}

void findPrefixesUtil(struct trieNode \*root, char prefix[],

int ind)

{

if (root == NULL)

return;

if (root->freq == 1)

{

prefix[ind] = '\0';

printf("%s\t",prefix);

return;

}

for (int i=0; i<MAX; i++)

{

if (root->child[i] != NULL)

{

prefix[ind] = i;

findPrefixesUtil(root->child[i], prefix, ind+1);

}

}

}

void findPrefixes(string arr[], int n)

{

struct trieNode \*root = newTrieNode();

root->freq = 0;

for (int i = 0; i<n; i++)

insert(root, arr[i]);

char prefix[MAX\_WORD\_LEN];

findPrefixesUtil(root, prefix, 0);

}

int main()

{

string arr[] = {"zebra", "dog", "duck", "dove"};

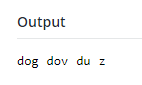
int n = sizeof(arr)/sizeof(arr[0]);

findPrefixes(arr, n);

return 0;

}

**OUTPUT:**



**RESULT:**