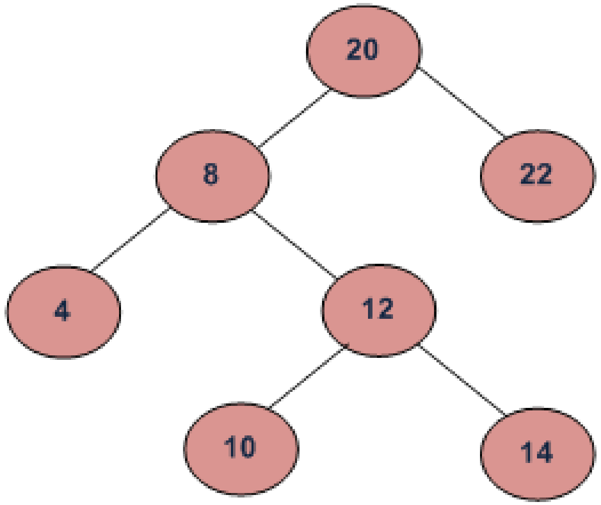
Find k-th smallest element in BST (Order Statistics in BST)

Given root of binary search tree and K as input, find K-th smallest element in BST.

For example, in the following BST, if k = 3, then output should be 10, and if k = 5, then output should be 14.



Method 1: Using Inorder Traversal.

Inorder traversal of BST retrieves elements of tree in the sorted order. The inorder traversal uses stack to store to be explored nodes of tree (threaded tree avoids stack and recursion for traversal, see this post). The idea is to keep track of popped elements which participate in the order statics. Hypothetical algorithm is provided below,

Time complexity: O(n) where n is total nodes in tree..

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion
Geometric Algorithms

Algorithm:

```
/* initialization */
pCrawl = root
set initial stack element as NULL (sentinal)

/* traverse upto left extreme */
while(pCrawl is valid )
   stack.push(pCrawl)
   pCrawl = pCrawl.left

/* process other nodes */
while( pCrawl = stack.pop() is valid )
   stop if sufficient number of elements are popped.
   if( pCrawl.right is valid )
      pCrawl = pCrawl.right
      while( pCrawl is valid )
         stack.push(pCrawl)
         pCrawl = pCrawl.left
```

Implementation:

```
#include <stdio.h>
#include <stdlib.h>

#define ARRAY_SIZE(arr) sizeof(arr)/sizeof(arr[0])

/* just add elements to test */
/* NOTE: A sorted array results in skewed tree */
int ele[] = { 20, 8, 22, 4, 12, 10, 14 };

/* same alias */
typedef struct node_t node_t;

/* Binary tree node */
struct node_t
{
    int data;

    node t* left;
```

Popular Posts

All permutations of a given string

Memory Layout of C Programs

Understanding “extern” keyword in C

Median of two sorted arrays

Tree traversal without recursion and without stack!

Structure Member Alignment, Padding and Data Packing

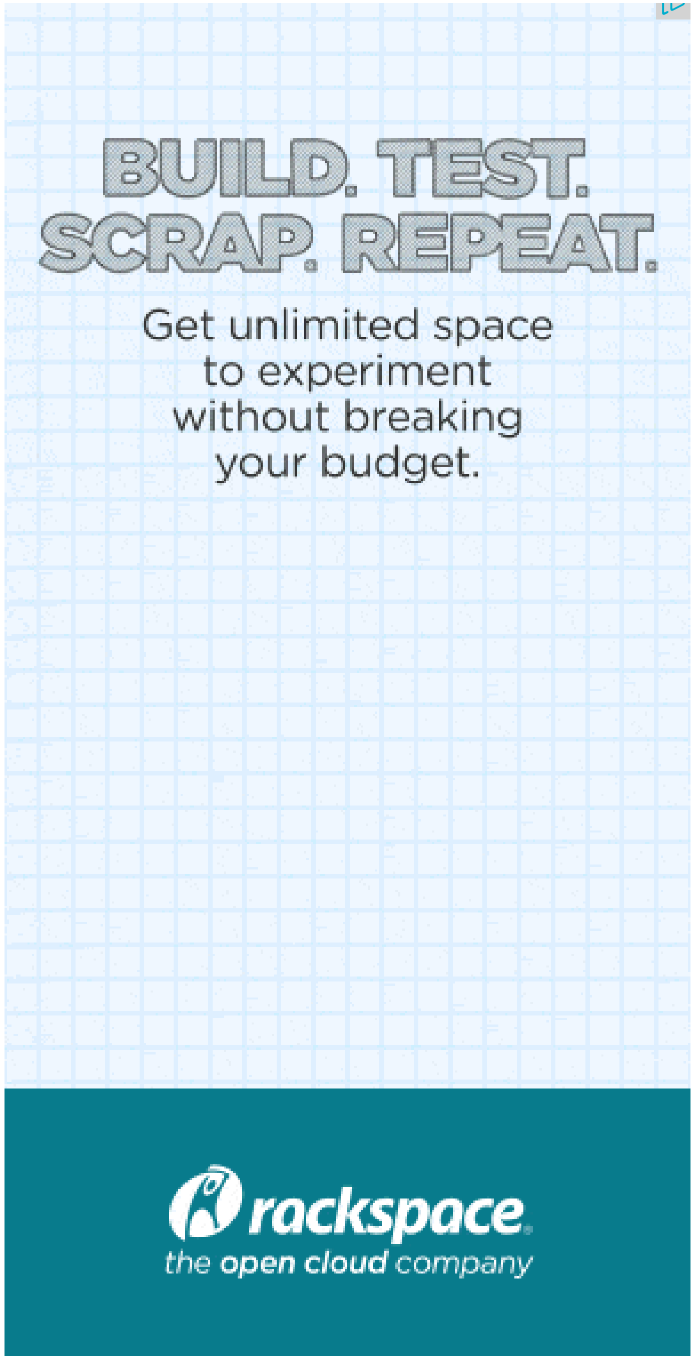
Intersection point of two Linked Lists

Lowest Common Ancestor in a BST.

Check if a binary tree is BST or not

Sorted Linked List to Balanced BST

```
    node_t* right;
};

/* simple stack that stores node addresses */
typedef struct stack_t stack_t;

/* initial element always NULL, uses as sentinal */
struct stack_t
{
    node_t*  base[ARRAY_SIZE(ele) + 1];
    int      stackIndex;
};

/* pop operation of stack */
node_t *pop(stack_t *st)
{
    node_t *ret = NULL;

    if( st && st->stackIndex > 0 )
    {
        ret = st->base[st->stackIndex];
        st->stackIndex--;
    }

    return ret;
}

/* push operation of stack */
void push(stack_t *st, node_t *node)
{
    if( st )
    {
        st->stackIndex++;
        st->base[st->stackIndex] = node;
    }
}

/* Iterative insertion
   Recursion is least preferred unless we gain something
*/
node_t *insert_node(node_t *root, node_t* node)
{
    /* A crawling pointer */
    node_t *pTraverse = root;
    node_t *currentParent = root;

    // Traverse till appropriate node
```



```
    while(pTraverse)
    {
        currentParent = pTraverse;

        if( node->data < pTraverse->data )
        {
            /* left subtree */
            pTraverse = pTraverse->left;
        }
        else
        {
            /* right subtree */
            pTraverse = pTraverse->right;
        }
    }

    /* If the tree is empty, make it as root node */
    if( !root )
    {
        root = node;
    }
    else if( node->data < currentParent->data )
    {
        /* Insert on left side */
        currentParent->left = node;
    }
    else
    {
        /* Insert on right side */
        currentParent->right = node;
    }

    return root;
}

/* Elements are in an array. The function builds binary tree */
node_t* binary_search_tree(node_t *root, int keys[], int const size)
{
    int iterator;
    node_t *new_node = NULL;

    for(iterator = 0; iterator < size; iterator++)
    {
        new_node = (node_t *)malloc( sizeof(node_t) );

        /* initialize */
        new_node->data   = keys[iterator];
```

Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 43 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 1 hour ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 1 hour ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 3 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 3 hours ago

```
        new_node->left   = NULL;
        new_node->right  = NULL;

        /* insert into BST */
        root = insert_node(root, new_node);
    }

    return root;
}

node_t *k_smallest_element_inorder(stack_t *stack, node_t *root, int k
{
    stack_t *st = stack;
    node_t *pCrawl = root;

    /* move to left extremen (minimum) */
    while( pCrawl )
    {
        push(st, pCrawl);
        pCrawl = pCrawl->left;
    }

    /* pop off stack and process each node */
    while( pCrawl = pop(st) )
    {
        /* each pop operation emits one element
           in the order
        */
        if( !--k )
        {
            /* loop testing */
            st->stackIndex = 0;
            break;
        }

        /* there is right subtree */
        if( pCrawl->right )
        {
            /* push the left subtree of right subtree */
            pCrawl = pCrawl->right;
            while( pCrawl )
            {
                push(st, pCrawl);
                pCrawl = pCrawl->left;
            }

            /* pop off stack and repeat */
        }
    }

    /* node having k-th element or NULL node */
    return pCrawl;
}

/* Driver program to test above functions */
int main(void)
{
    node_t* root = NULL;
    stack_t stack = { {0}, 0 };
    node_t *kNode = NULL;

    int k = 5;

    /* Creating the tree given in the above diagram */
    root = binary_search_tree(root, ele, ARRAY_SIZE(ele));

    kNode = k_smallest_element_inorder(&stack, root, k);

    if( kNode )
    {
        printf("kth smallest elment for k = %d is %d", k, kNode->data)
    }
    else
    {
        printf("There is no such element");
    }

    getchar();
    return 0;
}
```

Method 2: Augmented Tree Data Structure.

The idea is to maintain rank of each node. We can keep track of elements in a subtree of any node while building the tree. Since we need K-th smallest element, we can maintain number of elements of left subtree in every node.

Assume that the root is having N nodes in its left subtree. If K = N + 1, root is K-th node. If K < N, we will continue our search (recursion) for the Kth smallest element in the left subtree of root. If K > N + 1, we continue our search in the right subtree for the (K – N – 1)-th smallest element. Note that we need the count of elements in left subtree only.

Time complexity: O(n) where n is total nodes in tree.

Algorithm:

```
start:
if K = root.leftElement + 1
   root node is the K th node.
   goto stop
else if K > root.leftElements
   K = K - (root.leftElements + 1)
   root = root.right
   goto start
else
   root = root.left
   goto srart

stop:
```

Implementation:

```
#include <stdio.h>
#include <stdlib.h>

#define ARRAY_SIZE(arr) sizeof(arr)/sizeof(arr[0])

typedef struct node_t node_t;

/* Binary tree node */
struct node_t
{
    int data;
    int lCount;

    node_t* left;
    node_t* right;
};

/* Iterative insertion
   Recursion is least preferred unless we gain something
*/
node_t *insert_node(node_t *root, node_t* node)
{
```

```c
    /* A crawling pointer */
    node_t *pTraverse = root;
    node_t *currentParent = root;

    // Traverse till appropriate node
    while(pTraverse)
    {
        currentParent = pTraverse;

        if( node->data < pTraverse->data )
        {
            /* We are branching to left subtree
               increment node count */
            pTraverse->lCount++;
            /* left subtree */
            pTraverse = pTraverse->left;
        }
        else
        {
            /* right subtree */
            pTraverse = pTraverse->right;
        }
    }

    /* If the tree is empty, make it as root node */
    if( !root )
    {
        root = node;
    }
    else if( node->data < currentParent->data )
    {
        /* Insert on left side */
        currentParent->left = node;
    }
    else
    {
        /* Insert on right side */
        currentParent->right = node;
    }

    return root;
}

/* Elements are in an array. The function builds binary tree */
node_t* binary_search_tree(node_t *root, int keys[], int const size)
{
    int iterator;
    node_t *new_node = NULL;

    for(iterator = 0; iterator < size; iterator++)
    {
        new_node = (node_t *)malloc( sizeof(node_t) );

        /* initialize */
        new_node->data   = keys[iterator];
        new_node->lCount = 0;
        new_node->left   = NULL;
        new_node->right  = NULL;

        /* insert into BST */
        root = insert_node(root, new_node);
    }

    return root;
}

int k_smallest_element(node_t *root, int k)
{
    int ret = -1;

    if( root )
    {
        /* A crawling pointer */
        node_t *pTraverse = root;

        /* Go to k-th smallest */
        while(pTraverse)
        {
            if( (pTraverse->lCount + 1) == k )
            {
                ret = pTraverse->data;
                break;
            }
            else if( k > pTraverse->lCount )
            {
                /*  There are less nodes on left subtree
                    Go to right subtree */
                k = k - (pTraverse->lCount + 1);
                pTraverse = pTraverse->right;
            }
            else
            {
                /* The node is on left subtree */
                pTraverse = pTraverse->left;
            }
        }
    }

    return ret;
}

/* Driver program to test above functions */
int main(void)
{
    /* just add elements to test */
    /* NOTE: A sorted array results in skewed tree */
    int ele[] = { 20, 8, 22, 4, 12, 10, 14 };
    int i;
    node_t* root = NULL;

    /* Creating the tree given in the above diagram */
    root = binary_search_tree(root, ele, ARRAY_SIZE(ele));

    /*  It should print the sorted array */
    for(i = 1; i <= ARRAY_SIZE(ele); i++)
    {
        printf("\n kth smallest elment for k = %d is %d",
                 i, k_smallest_element(root, i));
    }

    getchar();
    return 0;
}
```

Thanks to **Venki** for providing post. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Tpoics:

- Print a Binary Tree in Vertical Order | Set 2 (Hashmap based Method)
- Print Right View of a Binary Tree
- Red-Black Tree | Set 3 (Delete)
- Construct a tree from Inorder and Level order traversals
- Print all nodes at distance k from a given node
- Print a Binary Tree in Vertical Order | Set 1
- Interval Tree
- Check if a given Binary Tree is height balanced like a Red-Black Tree

1 Tweet 0 1

Writing code in comment? Please use **ideone.com** and share the link here.

79 Comments **GeeksforGeeks**

Sort by Newest

Join the discussion…

AlienOnEarth • 4 days ago

Recursive Method in Java (Inorder Traversal):

```
private static int kthSmallest(BTNode root, int k) {

// base case

if(root == null)

return -1;

// simple inorder traversal

int left = kthSmallest(root.left, k);

count++;

if(count == k)

return root.data;

int right = kthSmallest(root.right, k);

if(left != -1)

return left;

else

return right;
```

^ | ⌄ • Reply • Share ›

opcoder • a month ago

The signature of the function should be

node * kth_smallest(node *root, int &k);

So according to this we should return node pointer of that node

my implementation:

```
node *kth_smallest(node *root, int &k)
{
if (!root)
return NULL;

node *left = kth_smallest(root->left, k);
if (!left){
if (--k == 0)
return root;
}
else
return left;
node *right = kth_smallest(root->right, k);
return right;
}
```

• Reply • Share ›

anon • 2 months ago

```
int getKthSmallest(struct Node *root, int *count)

{

if (root == NULL) return 0;

getKthSmallest (root->left, count);

(*count)++;

if (k == *count)
{

printf("\n %d Smalllest ellement %d", k, root->data);

}

getKthSmallest(root->right, count);

}
```

^ | ˅ • Reply • Share ›

Mohaan • 2 months ago

```
#include <stdio.h>
#include <stdlib.h>

#define ARRAY_SIZE(arr) sizeof(arr)/sizeof(arr[0])

typedef struct node_t node_t;

static bool kthSmallestFound;
static int count = 0;

/* Binary tree node */
struct node_t
{
int data;
int lCount;

node_t* left;
node_t* right;
};
```

see more

Reply · Share ›

Nachiket · 2 months ago

```
public class TreeAllApplication {

public int printKthOrder(Tree root, int k, int temp) {

if (root == null)

return temp>0?temp:0;

temp = printKthOrder(root.getLeft(), k, temp);

temp++;

if (k == temp) {

System.out.println(root.getData());

return temp;

}

temp = printKthOrder(root.getRight(), k, temp);
```

see more

Reply · Share ›

pulikesi · 4 months ago

```
inorder(root,k)
{
static int count=0;
if(root==NULL)
return;
inorder(2*root+1,k);
if(count==k)
print root->data
inorder(2*root+2,k);
}
```

˄ | ˅ • Reply • Share ›

groomnestle • 5 months ago

You can init an array starting at index[1] and traverse the tree in-order and pus
done, you can find kth smallest element at array[k].

˄ | ˅ • Reply • Share ›

Anon • 5 months ago

Why can't we straightaway do an inorder traversal, w/o recursion, using a stac
And keep on traversing till we find the number, as soon as found, break the wh

˄ | ˅ • Reply • Share ›

Gaurav Ambast • 6 months ago

```
#include<stdio.h>
#include<stdlib.h>

/* A binary tree tNode has data, pointer to left child
and a pointer to right child */
struct node
{
int data;
struct node* left;
struct node* right;
};

int getnumber(struct node* root)
{
if(root ==NULL)
```

return 0;
return (getnumber(root->left) + getnumber(root->right) + 1);

}

see more

⌃ | ⌄ • Reply • Share ›

its_dark • 7 months ago

We can follow this approach also :

Keep a static variable and once you reach the minimum element, initialize that element, increment that variable and if it equals k, return the element.

```
int kthsmallest(node *start, int k){

    if(!start->left && !start->right){     //for leaf
        if(c==-1)
            c=1;
        else
            c++;
        if(c==k)
            return start->data;
        return -1;
    }
    if(start->left){                 //if left exists
        int temp = -1;
```

see more

1 ⌃ | ⌄ • Reply • Share ›

Raj • 7 months ago

```
{
static int i=0;
if(root == NULL)
return ;

order(root->left,k);
i++;
if(i==k)
printf("%d",root->data);
order(root->right,k);
}
```

2 ˄ | ˅ • Reply • Share ›

Sumit Monga • 7 months ago

This solution is based upon traversing in inorder and using static variables to k
order:

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node * left, *right;

};

struct node * newNode(int data)

{
```

see more

Reply • Share ›

Rohit Rawat • 7 months ago

GeeksforGeeks.

this solution works. please review it

```
void inorder(struct bst *p, int *n)

{

    if(p == NULL)

        return;

    inorder(p->left, n);

    if(*n == 1)

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

    (*n)--;

    inorder(p->right, n);

}
```

Reply • Share ›

deb • 8 months ago

What about this ?

```
int findk(TNode *root, int k, int *result)
{
   if (root == NULL)
      return 0;
   /* l is the size of left subtree */
   int l = findk(root->left, k, result);
   if (k == l+1)
      *result = root->data;
   /* r is size of right subtree */
   int r = findk(root->right, k-l-1, result);
   return l+r+1;
}
```

Reply • Share ›

JOBBINE JOSEPH • 8 months ago

```
int KthSmallestElement(struct TreeNode *Node,int k)
{
while(1)
{
if(!Node)
{
return 0;
}
if(k == Node->rank)
{
return (Node->data);
}
else if(k>Node->rank)
{
```

```
k = k - Node->rank;
Node = Node->RChild;
}
else
{
Node = Node->LChild;
}
}

}
```

Reply • Share ›

Prakhar Jain • 9 months ago

I think the augmented tree approach has order O(log n) for a balanced BST.
But It is hard to implement a balanced BST with lcount parameter.

```
/* Paste your code here (You may delete these lines if not writing c
```

Reply • Share ›

Soumya • 9 months ago

How about this one?

```
 int kthSmallestUtil(Tree *tree, int &k,bool &found)
{
    if(!tree)return -1;
    int p = kthSmallestUtil(tree->left,k,found);
    if(!found)
    {
        if(!--k)
        {
            found = true;
            return tree->data;
        }
        return kthSmallestUtil(tree->right,k,found);
    }
    return p;
}

int kthSmallest(Tree *tree, int &k)
{
    bool found = false;
    int p = k;
    return kthSmallestUtil(tree,p,found);
}
```

• Reply • Share ›

pranjalgupta • 9 months ago

Awesome implementation @ Venki.

• Reply • Share ›

Pavan • 9 months ago

```
void  KthSmallestinBST(struct node* node , int k , int *kthmin)
{
    static int count =0;
    if (node == NULL)
         return ;
     else
     {

    KthSmallestinBST(node->left,k,kthmin);
    count++;
    if(count==k)
    *kthmin=node->data;
    KthSmallestinBST(node->right,k,kthmin);
    return ;
     }
}
```

see more

^ | ⌄ • Reply • Share ›

Pavan • 9 months ago

```
int  _KthSmallestinBST(struct node* node , int k , int *count)
{
    if (node == NULL)
         return 0;
     else
     {
    int l,m=0,n;
    l=_KthSmallestinBST(node->left,k,count);
    (*count)++;
    if(*count==k)
    m=node->data;
    n=_KthSmallestinBST(node->right,k,count);
    return (l|m|n);
     }
}
int KthSmallestinBST(struct node* node ,int k)
{


    int c =0;
```

```
        return _KthSmallestInBST(node , k , &c);

}
```

⌃ | ⌄ • Reply • Share ›

Pavan • 9 months ago

```
 #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);
```

see more

⌃ | ⌄ • Reply • Share ›

Pavan • 9 months ago

```
[sourcecode language="C"]
#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;
```

see more

^ | ˅ • Reply • Share ›

Pavan • 9 months ago

```
 #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);
```

see more

Reply • Share ›

kritika • 9 months ago

```
#include
#include
#include
struct node
{
struct node * left,*right;
int val;
};
void inorder(struct node * root,int k)
{
static int count=0;
if(root)
{
inorder(root->left,k);
count++;
if(count==k)
printf("%d",root->val);
inorder(root->right,k);
```

see more

Reply • Share ›

Ronny • 10 months ago

@GeeksforGeeks the link to the related post is broken.
Kindly update it

• Reply • Share ›

GeeksforGeeks → Ronny • 10 months ago

@Ronny: Thanks for pointing this out. The linked forum seems to be lc
link.

• Reply • Share ›

geekfreak • 10 months ago

Do reverse inorder traversal. If you have passed through k nodes from last the

```
public static int retreivekth(node n, int k , intp curr){
if(n==null)
return -1;
int a = retreivekth(n.right,k,curr);

if(a!=-1)
return a;
else
curr.set_val(curr.get_val()+1);

if(curr.get_val()==k)
return n.data;

return retreivekth(n.left,k,curr);

}
```

• Reply • Share ›

Himanshu • 10 months ago

```
 /* we can use a static variable and use inorder traversal.First we r
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

int count(struct node *root)
{
    if(root==NULL) return 0;
    else
        return count(root->left)+1+count(root->right);
}
void inorder(struct node *root,int *k)
{
```

see more

^ | ˅ • Reply • Share ›

sonali gupta • 10 months ago

simple approach

#include
#include
#include
typedef struct NODE
{
int info;
struct NODE *left,*right;

```
}node;
node *temp;
node *getnode()
{return((node *)malloc(sizeof(node)));
}
node *newNode(int x)
{
temp=getnode();
temp->info=x;
```

see more

^ | ⌄ • Reply • Share ›

zyzz • 10 months ago

this will take O(logN) on average

```
 int size(node *root)
{
if(root==NULL)
return 0;
else
return (size(root->left)+1+sizeof(root->right));
}

int smallest(int k,node *root)
{
int count;
count=size(root->left)+1;
if(k==count)
return (root->data);
else if(k<count)
return smallest(k,root->left);
else
return smallest(k-count,root->right);
}
```

⌃ | ⌄ • Reply • Share ›

Kunaal • 11 months ago

How about this?
using a static count variable in inorder traversal
O(n) and no extra data structure required.

```
 /* Paste your code here (You may delete these lines if not writing c

#include<stdio.h>
#include<stdlib.h>

struct Node
{
        int val;
        struct Node *left;
        struct Node *right;
};
typedef struct Node node;

node * newnode(int val)
```

see more

2 ⌃ | ⌄ • Reply • Share ›

Jitendra.BITS ➦ Kunaal • 10 months ago

This will have O(n) complexity right?

⌃ | ⌄ • Reply • Share ›

ultimate_coder ➦ Jitendra.BITS • 10 months ago

```
Looks like it is O(k).
```

^ | v • Reply • Share ›

shek8034 • 11 months ago

An alternative could be :

Do inorder traversal and store the result in an array. Print kth element of array.

Space complexity: O(n).

^ | v • Reply • Share ›

Ujjwal • 11 months ago

```
 Cant this work..??
-Build a stack by traversing the node in Inorder.
-Remove (n-k) elements from the stack, where 'n' is the total number 
-Top of the stack will give you 'kth' minimum..
```

^ | v • Reply • Share ›

root • a year ago

```
 int kthsmallest(node *root,int count)
{
static int i=0,val=0;
if(root)
{
kthsmallest(root->left,count);
i++; if(i==count) val=root->data;
kthsmallest(root->right,count);
}
return val;
}
```

• Reply • Share ›

anon_user ➦ root • 11 months ago

I did it in similar way

```
int kthSmallest(struct node* node,int k)
{
    static int count=1;
    int a,c,b;
    if(node==NULL)
        return 0;
    a=kthSmallest(node->left,k);
    if(count==k)
    {
        b= node->data;

    }
    else
        b=0;
    count++;

    c=kthSmallest(node->right,k);
    return (a+b+c);
}
```

• Reply • Share ›

abhishek08aug • a year ago

C++ code:

```
#include <iostream>
#include <stdlib.h>
using namespace std;

class tree_node {
  private:
    int data;
    tree_node * left;
    tree_node * right;
  public:
    tree_node() {
      left=NULL;
      right=NULL;
    }
    void set_data(int data) {
      this->data=data;
```

see more

^ | ⌄ • Reply • Share ›

abhishek08aug ➦ abhishek08aug • a year ago

My solution is recursive.

^ | ⌄ • Reply • Share ›

aman1234 ➦ abhishek08aug • 11 months ago

dude, have you done whole geeksforgeeks ? i am wondering...

^ | ⌄ • Reply • Share ›

naveen.bobbili • a year ago

```
 template<class T>
struct result {
  Tree<T>* node;
  int data;
};

template<class T>
struct result<T> kthInorder(Tree<T>* root, int k) {
  if (root == NULL) {
    struct result<T> a;
    a.data = 0;
    a.node = NULL;
    return a;
  }

  struct result<T> left = kthInorder(root->lchild(), k);
  if (left.node != NULL)
    return left;
```

see more

^ | ˅ • Reply • Share ›

rohit • a year ago

```
int ksmall(struct node*root,int *k)
{
int a;
if(root==NULL)
return(0);
{a=ksmall(root->left,k);
if(a!=0)
return(a);
}
(*k)--;
if(!(*k))
return(root->data);
a=ksmall(root->right,k);
return(a);
}
```

Reply • Share ›

Nirdesh Mani Sharma. • a year ago

```
void findK(Node* p, int& k) {.
if(! p || k < 0) return;.
findK(p->left, k);.
--k;.
if(k == 0) { print p->data;.
return;.
}.
findK(p->right, k);.
}
```

And one more observation:
To find the Nth smallest item, you only need to visit size of the left sub-tree.Yo
tree iif you also wanted to be able to find the Nth largest item.

Reply • Share ›

cyberWolf • a year ago

Kth smallest value in BST using Stack

```
 int findKthSmallest(treeNode* x, int k)
{
        int count = 0;
        stack<treeNode*> s;
        s.push(x);

        while(!s.empty())
        {
                while(x->left != NULL)
                {
                        s.push(x->left);
                        x=x->left;
                }

                x = s.top();
                s.pop();
```

see more

^ | ˅ • Reply • Share ›

doingit • 2 years ago

Isn't it a good solution

```

void Kthsmallest(node* root,int k){
   static int cnt = 0;
   if(root==NULL)
        return;
   Kthsmallest(root->left,k);
   cnt++;
   if(cnt == k)
        cout<<root->key<<" ";
   Kthsmallest(root->right,k);
}


```

^ | ˅ • Reply • Share ›

ashu • 2 years ago

code

```
 void get3(node *root,int* ind,int* val,int k){
        if(root==NULL) return;
        get3(root->left,ind,val,k);
        *ind+=1;
        if(*ind==k) *val=root->data;
        get3(root->right,ind,val,k);
}
void kthSmall(node *root){
        int ind=0;
        int val=-1;
        int k=5;
        get3(root,&ind,&val,k);
        cout<<val<<"\n";
}
```

^ | ⌄ • Reply • Share ›

Ashu • 2 years ago

```
 void get3(node *root,int* ind,int* val,int k){
        if(root==NULL) return;
        get3(root->left,ind,val,k);
        *ind+=1;
        if(*ind==k) *val=root->data;
        get3(root->right,ind,val,k);
}
void KthNode(node *root){
        int ind=0;
        int val=-1;
        int k=5;
        get3(root,&ind,&val,k);
        cout<<val<<"\n";
}
```

• Reply • Share ›

naresh • 2 years ago

Without using any global variable or static variable.
Note : found is used as a indicator.

Using the Inorder tree property.

```
int kth(struct node *root, int k, int *found){
if(root == NULL)
return k;
k = kth(root->left, k, found);
if(k == 1 && *found == 0){
printf("Kth =%d \n", root->data);
*found = 1;
return root->data;
}
if(*found == 0){
k--;
k = kth(root->right, k, found);
return k;
}
}
```

• Reply • Share ›

huha • 2 years ago

```
 struct node *nthorder(struct node* root,int *n)
{
        struct node* ptr=(struct node*)malloc(sizeof(struct node));
        if(!root) return NULL;
        ptr=nthorder(root->left,n);
        if(ptr) return ptr;
        if((*n)==1)
        {
        printf("nth is %d",root->data);
        return root;}
        (*n)--;
        ptr=nthorder(root->right,n);
        return ptr;
}
```

• Reply • Share ›

naresh huha • 2 years ago

It giving segmentation fault.

• Reply • Share ›

Load more comments

Subscribe   Add Disqus to your site