#include <iostream>

Using namespace std;

Int f(int n)

{

If(n==1) return 3;

Else return n+f(n-1)

}

Int main()

{

Cout<<f(5)<<endl;

system(“pause”);

return 0;

}//end main

for hw due wed oct 7

THE HEADER

#pragma once

#ifndef BINARYSEARCHTREE

#define BINARYSEARCHTREE

#include <iostream>

#include <string>

using namespace std;

class BST

{

private:

struct node

{

string name;

int days;

node \*left;

node \*right;

};

node \*root;

public:

BST() { root = NULL; }

void insert(node \* &r, string monthName, int monthDays)

{

node\*p = new node;

p->name = monthName;

p->days = monthDays;

p->left = p->right = NULL;

if (r == NULL)

{

r = p;

}//endif

else

{

if (p->name < r->name) insert(r->left, monthName, monthDays);

if (p->name > r->name) insert(r->right, monthName, monthDays);

}//endelse

}//end insert

void displayInorder(node \*r)

{

//LC P RC

if (r != NULL)

{

displayInorder(r->left);

cout << r->name << " ";

displayInorder(r->right);

}

}//end displayInorder

void search(node \*r, string monthName)

{

if (r == NULL) { cout << "Month name does not exist\n"; }

else if (r->name == monthName) { cout<<"Month name exists with "<<r->days<<" days\n"; }

else if (monthName < r->name) { search(r->left, monthName); }

else if (monthName > r->name) { search(r->right, monthName); }

}//end search

void insert(string monthName, int monthDays)

{

insert(root, monthName, monthDays);

}//end overriden insert

void displayInorder()

{

displayInorder(root);

}//end overriden displayinorder

void search(string monthName)

{

search(root, monthName);

}//end overriden search

};

#endif

//---------------------------------------------------------------------------

#include <iostream>

#include "BINARYSEARCHTREE.h"

using namespace std;

int f(int n)

{

if (n == 1) return 3;

else return n + f(n - 1);

}

int main()

{

cout << f(5) << endl;

struct MONTHDAYS {

string name;

int days;

};

MONTHDAYS a[12] = { {"Jan",31 }, {"Feb",28 }, {"Mar",31 }, {"Apr",30 }, {"May",31}, {"Jun",30},

{"Jul",31}, {"Aug",31}, {"Sep",30}, {"Oct",31}, { "Nov",30}, { "Dec",31 } };

BST b;

for (int i = 0; i < 12; ++i) { b.insert(a[i].name, a[i].days); }

b.displayInorder();

cout << endl;

string toSearch;

cout << "search month to determine the days ";

cin >> toSearch;

b.search(toSearch);

system("pause");

return 0;

}//end main

tHE BIG-Oh is an estimation to compute the runtime of functions or program segments

for recursive functions, find how many times the function calls itself

ex:

int f(int n)

{

if(n==1) return 3;

else return n+f(n-1)

}

//the runtime of this function

if n=5,calls 5, becomes 4, becomes3, becomes 2, becomes 1

this is O(n) because the function calls itself n times

ex:

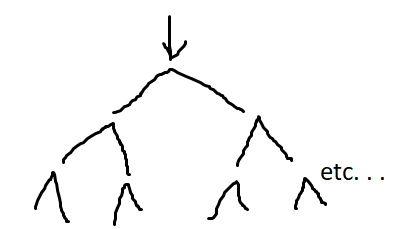
int f(n)

{

if (n<=1) return 0;

else return f(n-1)+f(n-2);

}

goes through 2 calls 

runtime is = 2^0 + 2^1 + 2^2 + . . . . .+ 2^n = 2^(n+1) -1

=2(2^n -1) = O(2^n)

ex:

for(i=1; i<n; ++i) {cout<<i} //this is O(n)

for(int j=0; j<m; ++j){cout<<j} //this is O(m)

runtime = O(n) + O(m)

= O(max(n,m))

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

{ //this outer loop is also O(n)

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

{

cout<<i+j; //this inner loop is O(n)

}

}

runtime = O(n) \* O(n)

= O(n\*n)

=O(n^2)

ex:

f(n) = (n+1)^5 (n-2)^3 + n^7log(n^10 + 10n)

//write stepbystep

=(n+1)^5 becomes O(n^5)

=(n-2)^3 becomes O(n^3)

=n^7 becomes O(n^7).

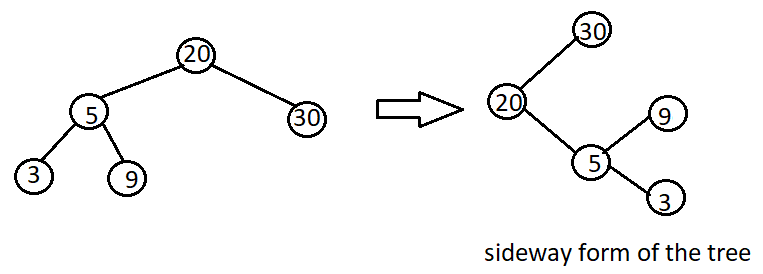
=//with log at some point it will become the coef? so it becomes O(log n)

O(n^5) \* O(n^3) + O(n^7 logn)

O(max(n^8, n^7 log2n))

~~//for log ex n=8 -> 8^8, 8^7logbase2 8 = 8^7logbase2 ^2.~~

O(n^5) \* O(n^3) + O(n^7 logn)=O(n^8)



INSTEAD OF GOING LEFT FIRST, WE GO RIGHT FIRST

//calling statement

int s=0

displaySideway(root, 2);

void displaySideway(node \*r, int s)

{

if(r!=NULL)

{

//we print the right side first

displaySideway(r->right, s+=5);

cout<<setw(s) //skip this many spaces to display r->info<<endl;

displaySideway(r->left, 5)

}//endif

}//end displaySideway

DISPLAYS TREE SIDEWAYS