Factorial

#include<iostream>

using namespace std;

int fact(int n)

{

//base case

if(n==0)

return 1;

//recursive call

int small\_ans=fact(n-1);

return n\*small\_ans;

}

int main()

{

int n;

cin>>n;

cout<<fact(n);

}

2.fibonaccin

#include<iostream>

using namespace std;

int fib(int n)

{

//base case

if(n==0)

return 0;

if (n==1)

return 1;

//recursion call

return fib(n-1)+fib(n-2);

}

int main()

{

int n;

cin>>n;

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

{

cout<<fib(i)<<" ";

}

}

Check array is Sorted or not

#include<iostream>

using namespace std;

bool isSorted(int a[],int n)

{

//base case

if(n==0 || n==1)

return true;

//recursive case

if(a[0]<a[1] and isSorted(a+1,n-1))

{

return true;

}

return false;

}

int main()

{

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

int n=sizeof(a)/sizeof(int);

cout<<isSorted(a,n);

}

Calculate power

#include<iostream>

using namespace std;

int power(int a,int b)

{

if(b==0)

return 1;

return a\*power(a,b-1);

}

int main()

{

int a,b;

cin>>a>>b;

cout<<power(a,b);

}

Fast Power

#include<iostream>

using namespace std;

int power(int a,int b)

{

if(b==0)

return 1;

return a\*power(a,b-1);

}

int fast\_pow(int a,int b)

{

if(b==0)

return 1;

int smaller\_ans=fast\_pow(a,b/2);

smaller\_ans\*=smaller\_ans;

if(b&1)

return smaller\_ans\*a;

return smaller\_ans;

}

int main()

{

int a,b;

cin>>a>>b;

cout<<fast\_pow(a,b);

}

Increasing and decreasing order

#include<iostream>

using namespace std;

void inc(int n)

{

//Base case

if(n==0)

return;

inc(n-1);

cout<<n<<" ";

}

void dec(int n)

{

if(n==0)

return;

cout<<n<<" ";

dec(n-1);

}

int main()

{

int n;

cin>>n;

inc(n);

dec(n);

}

Binary Search

#include<iostream>

using namespace std;

int binarySearch(int a[],int l,int r,int x )

{

while(l<r)

{

int mid=r-l/2;

if(a[mid]==x)

return mid;

if(a[mid]>x)

return binarySearch(a,l,mid-1,x);

return binarySearch(a,mid+1,r,x);

}

return -1;

}

int main()

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 10;

int n = sizeof(arr) / sizeof(int);

int result = binarySearch(arr, 0, n - 1, x);

(result == -1) ? cout << "Element is not present in array"

: cout << "Element is present at index " << result;

return 0;

}

Tioling

#include<iostream>

using namespace std;

int tile(int n)

{

if(n==0 )

return 0;

if(n==1)

return 1;

return tile(n-1)+tile(n-2);

}

int main()

{

cout<<tile(12);

}

Number to Spelling

#include<iostream>

using namespace std;

char words[][10]={"Zero","One","Two","Three","Four","Five","Six","Seven","Eight","Nine"};

void printLetter(int n)

{

//base case

if(n==0)

return;

//recursive call first print the spelling 204

printLetter(n/10);

int digit=n%10;

cout<<words[digit]<<" ";

}

int main(){

int n;

cin>>n;

printLetter(n);

}

First occurance linear Search

#include<iostream>

using namespace std;

int linearSearch(int \*a,int n,int key)

{

if(n==0)

return -1;

//rec case

if(a[0]==key)

return 0;

int i=linearSearch(a+1,n-1,key);

if(i==-1)

return -1;

return i+1;

}

int main()

{

int a[]={1,2,3,4,79,8,10,7};

int n=sizeof(a)/sizeof(int);

int key=7;

cout<<linearSearch(a,n,key);

}

Bubble Sort

#include<iostream>

using namespace std;

int bubble(int a[],int n)

{

if(n==1)

return;

for(int j=0;j<n-1;j++)

{

if(a[j]>a[j+1])

swap(a[j],a[j+1]);

}

bubble(a,n-1);

}

int main()

{

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

int n=sizeof(a)/sizeof(int);

int ans=bubble(a,n);

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

{

cout<<ans(i);

}

}

Quiz questions

0. Title: Recursion output 1

Description: What does the following function do?<br>

<pre>

int fun(unsigned int n)

{

if (n == 0 || n == 1)

return n;

if (n%3 != 0)

return 0;

return fun(n/3);

}</pre>

0.It returns 1 when n is a multiple of 3, otherwise returns 0.

1.It returns 1 when n is a power of 3, otherwise returns 0.

2.It returns 0 when n is a multiple of 3, otherwise returns 1.

3.It returns 0 when n is a power of 3, otherwise returns 1.

1. Title: Recursion output 2

Description: Predict the output of following program<br><pre><code="lang-cpp">

#include <stdio.h>

int f(int n)

{

if(n <= 1)

return 1;

if(n%2 == 0)

return f(n/2);

return f(n/2) + f(n/2+1);

}

int main()

{

printf("%d", f(11));

return 0;

}</code></pre>

0.Stack Overflow

1.3

2.4

3.5

2. Title: Recursion 1

Description: 5. Consider the following code snippet:<pre><code="lang-cpp">

void my\_recursive\_function()

{

my\_recursive\_function();

}

int main()

{

my\_recursive\_function();

return 0;

}</code></pre>

0.The code will be executed successfully and no output will be generated

1.The code will be executed successfully and random output will be generated

2.The code will show a compile time error

3.The code will run for some time and stop when the stack overflows

3. Title: Recursion 2

Description: How many times is the recursive function called, when the following code is executed?<pre>

void my\_recursive\_function(int n)

{

if(n == 0)

return;

printf("%d ",n);

my\_recursive\_function(n-1);

}

int main()

{

my\_recursive\_function(10);

return 0;

}

</pre>

0.9

1.10

2.11

3.12

4. Title: Recursion 3

Description: Which of the following statements is true?

0.Recursion is always better than iteration

1.Recursion uses more memory compared to iteration

2.Recursion uses less memory compared to iteration

3.Iteration is always better and simpler than recursion

5. Title: Recursion 4

Description: Iteration uses a repetition structure whereas recursion uses

0.Sorting structure

1.Selection structure

2.Controlling structure

3.All of them

6. Title: Recursion 5

Description: What does the following function do?<pre>

int fun(int x, int y)

{

if (y == 0) return 0;

return (x + fun(x, y-1));

}

</pre>

0.x + y

1.x + x\*y

2.x<sup>y</sup>

3.x\*y

7. Title: Recursion 6

Description: Minimum number of moves required to solve a Tower of Hanoi puzzle is

0.2n<sup>2</sup>

1.2<sup>n-1</sup>

2.2<sup>n</sup>-1

3.2n-1

8. Title: Recursion 7

Description: Recursion is memory-intensive because:

0.Recursive functions tend to declare many local variables.

1.Previous function calls are still open when the function calls itself and the activation records of these previous calls still occupy space on the call stack.

2.Many copies of the function code are created.

3.It requires large data values.

9. Title: Recursion 8

Description: What will be the output of the following code?<pre><code="lang-cpp">

int cnt=0;

void my\_recursive\_function(int n)

{

if(n == 0)

return;

cnt++;

my\_recursive\_function(n/10);

}

int main()

{

my\_recursive\_function(123456789);

printf("%d",cnt);

return 0;

}

</code></pre>

0.10

1.123456789

2.0

3.9

Last Occurance

#include<iostream>

using namespace std;

int lastOccurance(int \*a,int n,int key)

{

//Base case

if(n==0)

return -1;

//Rec Case

int i=lastOccurance(a+1,n-1,key);

if(i==-1)

{

if(a[0]==key)

return 0;

else return -1;

}

//otherwise if i returned by subproblem is not -1

return i+1;

}

int linearSearch(int \*a,int n,int key)

{

if(n==0)

return -1;

//rec case

if(a[0]==key)

return 0;

int i=linearSearch(a+1,n-1,key);

if(i==-1)

return -1;

return i+1;

}

int main()

{

int a[]={1,2,3,4,7,8,10,7};

int n=sizeof(a)/sizeof(int);

int key=7;

cout<<linearSearch(a,n,key)<<endl;

cout<<lastOccurance(a,n,key);

}

All Occuranmce

#include<iostream>

using namespace std;

int storeOccurance(int \*a,int i,int n,int key ,int \*out,int j)

{

//base case

if(i==n)

{

return j;

}

if(a[i]==key)

{

out[j]=i;

//increament j to accomodate current occurance

return storeOccurance(a,i+1,n,key,out,j+1);

}

//j remains unchanged

return storeOccurance(a,i+1,n,key,out,j);

}

void alloccurance(int \*a,int i,int n,int key)

{

//

if(i==n)

return;

if(a[i]==key)

cout<<i<<",";

alloccurance(a,i+1,n,key);

}

int lastOccurance(int \*a,int n,int key)

{

//Base case

if(n==0)

return -1;

//Rec Case

int i=lastOccurance(a+1,n-1,key);

if(i==-1)

{

if(a[0]==key)

return 0;

else return -1;

}

//otherwise if i returned by subproblem is not -1

return i+1;

}

int linearSearch(int \*a,int n,int key)

{

if(n==0)

return -1;

//rec case

if(a[0]==key)

return 0;

int i=linearSearch(a+1,n-1,key);

if(i==-1)

return -1;

return i+1;

}

int main()

{

int a[]={1,2,3,4,7,8,10,7};

int n=sizeof(a)/sizeof(int);

int key=7;

cout<<linearSearch(a,n,key)<<endl;

cout<<lastOccurance(a,n,key)<<endl;

alloccurance(a,0,n,key);

int out[1000];

int cnt=storeOccurance(a,0,n,key,out,0);

cout<<"Count"<<cnt<<endl;

for(int k=0;k<cnt;k++)

{

cout<<out[k]<<",";

}

}

Merge Sort Problem

#include<iostream>

using namespace std;

void merge(int a[],int s,int e)

{

int mid=(s+e)/2;

int i=s;

int j=mid+1;

int k=s;

int temp[100];

while(i<=mid && j<=e)

{

if(a[i]<a[j])

{

temp[k++]=a[i++];

}

else {

temp[k++]=a[j++];

}

}

while(i<=mid)

{

temp[k++]=a[i++];

}

while(j<=e)

{

temp[k++]=a[j++];

}

for(int i=s;i<=e;i++)

{

a[i]=temp[i];

}

}

void mergeSort(int a[],int s,int e)

{

if(s>=e)

{

return;

}

int mid=(s+e)/2;

mergeSort(a,s,mid);

mergeSort(a,mid+1,e);

merge(a,s,e);

}

int main()

{

int a[100];

int n;

cin>>n;

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

{

cin>>a[i];

}

mergeSort(a,0,n-1);

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

{

cout<<a[i]<<" ";

}

}

Inversion Count

#include<iostream>

using namespace std;

int merge(int a[],int s,int e)

{

int mid=(s+e)/2;

int i=s;

int j=mid+1;

int k=s;

int cnt=0;

int temp[1000];

while(i<=mid && j<=e)

{

if(a[i]<a[j])

{

temp[k++]=a[i++];

}

else{

temp[k++]=a[j++];

cnt+=mid-i+1;

}

}

// fill the array if some elements are left in one of the arrays

while(i<=mid)

{

temp[k++]=a[i++];

}

while(j<=mid)

{

temp[k++]=a[j++];

}

//copy all the eement on the origina array

for(int i=s;i<=e;i++)

{

a[i]=temp[i];

}

return cnt;

}

int inversion\_count(int a[],int s,int e)

{

//base case

if(s>=e)

{

return 0;

}

//merge sort

int mid=(s+e)/2;

int x=inversion\_count(a,s,mid);

int y=inversion\_count(a,mid+1,e);

int z=merge(a,s,e);

return x+y+z;

}

int main()

{

int a[]={1,5,2,6,3,0};

int n=sizeof(a)/sizeof(int);

cout<<inversion\_count(a,0,n-1)<<endl;

return 0;

}

Quick SORT

#include<iostream>

using namespace std;

int partion(int \*a,int s ,int e )

{

int i=s-1; //one step behind the start

int j=s;

int pivot=a[e];

for(int j=s;j<=e-1;j++)

{

if(a[j]<=pivot) //merge the smaller element into the region

{

i=i+1;

swap(a[i],a[j]);

}

j=j+1;

}

swap(a[i+1],a[e]);

return i+1;

}

void quickSort(int \*a,int s,int e )

{

//base case

if(s>=e)

return;

int p=partion(a,s,e); //Recursive Call

quickSort(a,s,p-1); //left

quickSort(a,p+1,e); //right

}

int main()

{

int a[]={2,7,8,6,1,5,4};

int n=sizeof(a)/sizeof(int);

quickSort(a,0,n-1);

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

{

cout<<a[i]<<" ";

}

}

String TO INTEGER

#include<iostream>

#include<cstring>

using namespace std;

int stringToIInt(char \*a,int n)

{

if(n==0)

{

return 0;

}

int digit=a[n-1]-'0';

int small\_ans=stringToIInt(a,n-1);

return small\_ans\*10+digit;

}

int main()

{

//given a string convert it into integer recursively

char a[]="1234";

int len=strlen(a);

int x=stringToIInt(a,len);

cout<<stringToIInt(a,len)<<endl;

cout<<x+1<<endl;

}

Replace Pi to 3.14

#include<iostream>

using namespace std;

void ReplacePi(char a[],int i)

{

//base case

if(a[i]=='\0' or a[i+1]=='\0')

{

return;

}

//look for pi for currenr location

if(a[i]=='p' and a[i+1]=='i')

{

//shifting + replacement with 3.14

int j=i+2;

//take j to the end of the array

while(a[j]!='\0')

{

j++;

}

//shifting (right to left)

while(j>=i+2)

{

a[j+2]=a[j];

j--;

}

//replacement + recursion call from rem part

a[i]='3';

a[i+1]='.';

a[i+2]='1';

a[i+3]='4';

ReplacePi(a,i+4);

}

else{

//go to the next position

ReplacePi(a,i+1);

}

return;

}

int main()

{

char a[1000];

cin>>a;

ReplacePi(a,0);

cout<<a<<endl;

}

Ladder Problem

#include<iostream>

using namespace std;

int f(int n,int k)

{

if(n==0)

{

return 1;

}

if(n<0)

{

return 0;

}

int ans=0;

for(int i=1;i<=k;i++)

{

ans += f(n-i,k);

}

return ans;

}

int main()

{

int n,k;

cin>>n>>k;

cout<<f(n,k)<<endl;

return 0;

}

Friends Problem

#include<iostream>

using namespace std;

int pairf(int n)

{

if(n==0 || n==1)

{

return 1;

}

return pairf(n-1)+(n-1)\*pairf(n-2);

}

int main()

{

int n;

cin>>n;

cout<<pairf(n);

}

Count the Binary String

#include<iostream>

using namespace std;

int countSt(int n)

{

int a[n],b[n];

a[0]=b[0]=1;

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

{

a[i]=a[i-1]+b[i-1];

b[i]=a[i-1];

}

return a[n-1]+b[n-1];

}

int main()

{

int n;

cin>>n;

cout<<countSt(n);

}

Tower of Hanoi Problem

#include<iostream>

using namespace std;

void move(int n,char src,char helper,char des)

{

if(n==0)

{

return;

}

move(n-1,src,des,helper);

cout<<" shift disk "<<n<<" from "<<src<<" to "<<des<<endl;

move(n-1,helper,src,des);

}

int main()

{

int n;

cin>>n;

move(n,'A','B','C');

}

Subsequence Generation

#include<iostream>

using namespace std;

void generate\_sequence(char \*in,char \*out,int i, int j)

{

//base case

if(in[i]=='\0')

{

out[j]='\0';

cout<<out<<endl;

return;

}

//Rec Case

//Include the current char

out[j]=in[i];

generate\_sequence(in,out,i+1,j+1);

//exclude the current char

generate\_sequence(in,out,i+1,j);

}

int main()

{

char input[]="abc";

char output[10];

generate\_sequence(input,output,0,0);

}

Generate Brackets

#include<iostream>

using namespace std;

void generate\_brackets(char \*out,int n,int idx,int open,int close)

{

//Base Case

if(idx==2\*n)

{

out[idx]='\0';

cout<<out<<endl;

return;

}

//recursive call

//2 option

if(open<n)

{

out[idx]='(';

generate\_brackets(out,n,idx+1,open+1,close);

}

if(close<open)

{

out[idx]=')';

generate\_brackets(out,n,idx+1,open,close+1);

}

return;

}

int main()

{

int n;

cin>>n;

char out[1000];

int idx=0;

generate\_brackets(out,n,0,0,0);

}

Knapsack Recursion

#include<iostream>

using namespace std;

int profit(int n,int C,int \*wt,int \*prices)

{

//base case

if(n==0 || C==0)

{

return 0;

}

//Recursive Case

int ans=0;

int inc,exc;

inc=exc=INT\_MIN;

if(wt[n-1]<=C)

{

inc=prices[n-1]+profit(n-1,C-wt[n-1],wt,prices);

}

//exclude

exc=profit(n-1,C,wt,prices);

ans=max(inc,exc);

return ans;

}

int main()

{

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

int prices[]={40,20,30,100};

int N=4;

int C=7;

cout<<profit(N,C,weights,prices);

}

Phone KeyPad Problem

#include<iostream>

using namespace std;

char keypad[][10]={"","","ABC","DEF","GHI","JKL","MNO","PQRS","TUV","WXYZ"};

void generate\_name(char \*in,char \*out,int i,int j)

{

//base case

if(in[i]=='\0')

{

out[j]='\0';

cout<<out<<" ";

return;

}

//Recursive Call

int digit=in[i]-'0';

for(int k=0;keypad[digit][k]!='\0';k++)

{

out[i]=keypad[digit][k];

//fill the remaining Part

generate\_name(in,out,i+1,j+1);

}

return;

}

int main()

{

char in[100];

cin>>in;

char out[100];

generate\_name(in,out,0,0);

return 0;

}

Rat In a Maze

#include<iostream>

using namespace std;

bool ratMaze(char maze[10][10],int soln[][10],int i,int j,int m,int n)

{

if(i==m && j==n)

{

soln[m][n]=1;

//print the path

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

{

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

{

cout<<soln[i][j]<<" ";

}

cout<<endl;

}

cout<<endl;

return true;

}

//Rat should be inside gride

if(i>m || j>n)

{

return false;

}

if(maze[i][j]=='X')

{

return false;

}

//Assume solution exists through current cell

soln[i][j]=1;

bool rightSuccess=ratMaze(maze,soln,i,j+1,m,n);

bool downSuccess=ratMaze(maze,soln,i+1,j,m,n);

//backtraking

soln[i][j]=0;

if(rightSuccess || downSuccess)

{

return true;

}

return false;

}

int main()

{

char maze[10][10]={

"0000",

"00X0",

"000X",

"0X00",

};

int soln[10][10]={0};

int m=4,n=4;

bool ans=ratMaze(maze,soln,0,0,m-1,n-1);

if(ans==false)

{

cout<<"pATH NOT FOUND";

}

}

Permutation Problem

#include<iostream>

using namespace std;

void permute(char \*in,int i)

{

//base case

if(in[i]=='\0')

{

cout<<in<<",";

return;

}

//recursive Case

for(int j=i;in[j]!='\0';j++)

{

swap(in[i],in[j]);

permute(in,i+1);

// Backtracking -To Restore the Original Array

swap(in[i],in[j]);

}

}

int main()

{

char in[100];

cin>>in;

permute(in,0);

}

N QUEEN

#include<iostream>

using namespace std;

bool isSafe(int board[][10],int i,int j,int n)

{

//You can check for the column

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

{

if(board[row][j]==1)

{

return false;

}

}

//You can check for the left diogonal

int x=i;

int y=j;

while(x>=0 && y>=0)

{

if(board[i][j]==1)

{

return false;

}

x--;

y--;

}

//you have to check for right diagonal

x=i;

y=j;

while(x>=0 && y<n)

{

if(board[i][j]==1)

{

return false;

}

x--;

y++;

}

//the position is now safe col and diagonals.

return true;

}

bool solveNQueen(int board[][10],int i,int n)

{

//Base case

if(i==n)

{

//you have successfully place queens in n rows(0..n-1)

//print the board

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

{

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

{

if(board[i][j]==1)

{

cout<<"Q";

}

else

{

cout<<"\_ ";

}

}

cout<<endl;

}

return true;

}

//Recursive Case

//Try to place the queen in the current row and call on the Remaining part

//which is solve by recursion

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

{

//I have to check if i,jth place is safe to place the queen or not

if(isSafe(board,i,j,n))

{

//place the queen Assuming i,j is the right position

board[i][j]=1;

bool nextQueenRakhPaye=solveNQueen(board,i+1,n);

if(nextQueenRakhPaye)

{

return true;

}

//i,,j is not right position

board[i][j]=0; //Backtracking

}

}

//You have tried for all position in the current row but could'nt place a queen

return false;

}

int main()

{

int n;

cin>>n;

int board[10][10]={0};

solveNQueen(board,0,n);

return 0;

}