**All code are written in c++.**

1. **Tower of Hanoi**

#include<iostream>

using namespace std;

void towerHanoi(int n,char Source, char Aux,char Desti)

{

if(n==1)

{

cout<<"Move Disk "<<n<<" from "<<Source<<" to "<<Desti<<endl;

return;

}

towerHanoi(n-1,Source,Desti,Aux);

cout<<"Move Disk "<<n<<" from "<<Source<<" to "<<Desti<<endl;

towerHanoi(n-1,Aux,Source,Desti);

}

int main()

{

int n;

cout<<"Enter no. of disks:";

cin>>n;

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

return 0;

}

1. **All Permutations of a string taken from user.**

#include <bits/stdc++.h>

using namespace std;

void permutation(string& a, int l, int r)

{

if (l == r)

cout << a << endl;

else {

for (int i = l; i <= r; i++) {

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

permutation(a, l + 1, r);

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

}

}

}

int main()

{

string str;

cin>>str;

int n = str.size();

permutation(str, 0, n - 1);

return 0;

}

1. **Printing all the binary strings of length n.**

#include <bits/stdc++.h>

using namespace std;

void print(int arr[], int n)

{

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

cout << arr[i] << " ";

}

cout << endl;

}

void BinaryStrings(int n, int arr[], int i)

{

if (i == n) {

print(arr, n);

return;

}

arr[i] = 0;

BinaryStrings(n, arr, i + 1);

arr[i] = 1;

BinaryStrings(n, arr, i + 1);

}

int main()

{

int n ;

cout<<"Enter the length:";

cin>>n;

int arr[n];

BinaryStrings(n, arr, 0);

return 0;

}

1. **Calculate Greatest Common Division GCD using Euclid’s algorithm**

#include <bits/stdc++.h>

using namespace std;

int gcd(int a, int b)

{

if (a == 0)

return b;

return gcd(b % a, a);

}

int main()

{

int a, b;

cin>>a>>b;

cout << gcd(a, b)<< endl;

}

1. **Implement merge sort algorithm using recursion**

#include<iostream>

using namespace std;

void merge(int arr[], int s, int e) {

int mid = (s+e)/2;

int leftLength = mid-s+1;

int rightLength = e-mid;

int \*left = new int[leftLength];

int \*right = new int[rightLength];

int originalIndex = s;

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

left[i] = arr[originalIndex++];

}

originalIndex = mid+1;

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

{

right[i] = arr[originalIndex++];

}

int leftIndex = 0;

int rightIndex = 0;

originalIndex = s;

while(leftIndex < leftLength && rightIndex < rightLength) {

if(left[leftIndex] <= right[rightIndex]){

arr[originalIndex++] = left[leftIndex++];

}

else{

arr[originalIndex++] = right[rightIndex++];

}

}

while(leftIndex < leftLength) {

arr[originalIndex++] = left[leftIndex++];

}

while(rightIndex < rightLength) {

arr[originalIndex++] = right[rightIndex++];

}

}

void mergeSort(int arr[], int s, int e) {

if(s >= e)

{

return ;

}

int mid = (s+e)/2;

mergeSort(arr, s, mid);

mergeSort(arr, mid+1, e);

merge(arr, s, e);

}

int main() {

int arr[] = {12, 11, 13, 5, 6, 7};

int size = 6;

mergeSort(arr, 0, size-1);

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

cout << arr[i] << " ";

}cout << endl;

return 0;

}

1. **Solve the N Queen problem in chess using Recursion**

void addSolution(vector<vector<int> > &ans, vector<vector<int> > &board, int n) {

vector<int> temp;

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

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

temp.push\_back(board[i][j]);

}

}

ans.push\_back(temp);

}

bool isSafe(int row, int col, vector<vector<int> > &board, int n) {

int x = row;

int y = col;

while(y>=0) {

if(board[x][y] == 1)

return false;

y--;

}

x = row;

y = col;

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

if(board[x][y] == 1)

return false;

y--;

x--;

}

x = row;

y = col;

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

if(board[x][y] == 1)

return false;

y--;

x++;

}

return true;

}

void solve(int col, vector<vector<int> > &ans, vector<vector<int> > &board, int n) {

if(col == n) {

ans = board;

return;

}

for(int row = 0; row <n; row++) {

if(isSafe(row,col, board,n)) {

board[row][col] = 1;

solve(col+1, ans, board, n);

board[row][col] = 0;

}

}

}

vector<vector<int>> nQueens(int n)

{

vector<vector<int> > board(n, vector<int>(n,0));

vector<vector<int> > ans;

solve(0, ans, board, n);

return ans;

}

**7. Remove a given element form a linked list using recursion**

class Solution {

public:

ListNode\* removeElements(ListNode\* head, int val) {

ListNode \*answer = new ListNode(0);

ListNode \*curr = answer;

while(head) {

if(head -> val != val) {

curr -> next = head;

curr = head;

}

head = head -> next;

}

curr -> next = nullptr;

return answer -> next;

}

};

1. **Solve egg dropping problem using recursion.**

class Solution {

public:

int EggDrop(int e, int f) {

if(e==1)return f;

if(f==0||f==1)return f;

int ans = INT\_MAX;

for(int i =1;i<=f;i++){

int temp = 1+max(EggDrop(e-1,i-1),EggDrop(e,f-i));

ans = min(ans,temp);

}

return ans;

}

};