**Problem 4 : 8 queen’s probles**

***Code***

#include <stdio.h>

#include <stdlib.h>

void queens(int);

int promising(int);

int n;

int col[255]={0,};

int cnt=0;

int main() {

//Receive number of queen

printf("Input n (Quenn number) : ");

scanf("%d", &n);

//Find the chessboard so that no two queens attack each other.

queens(0);

//Display number of ways there are to place n mutually non-attacking queens on an 8X8chessboard

printf("\n\n\*\*Total number of ways = %d \n",cnt);

}

void printResult() { //Dispaly each position

int i;

printf("\n-------result : ");

for(i = 1; i<=n; i++) //Print the each position

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

cnt++; //check the number of way

}

void queens(int i) {

int j;

if(promising(i)) { //Check the promising and excute prunning or continue

if(i == n)

printResult(); //Display correct position

else

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

col[i+1] = j;

queens(i+1);

}

}

}

int promising(int i) { // Check two queens share the same row, column, or diagonal.

int k = 1, promising = 1;

while(k < i && promising) {

if(col[i] == col[k] || abs(col[i]-col[k]) == abs(i-k))

promising = 0;

k++;

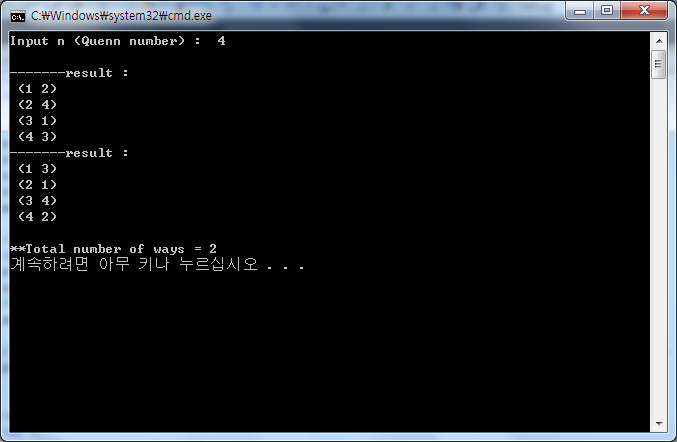
}

return promising;

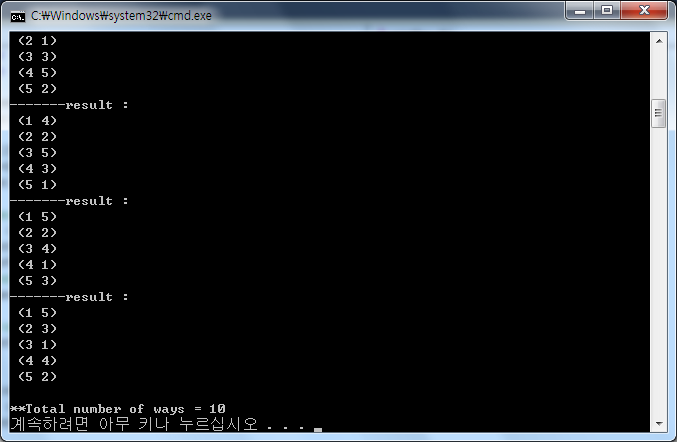
}

***Print***

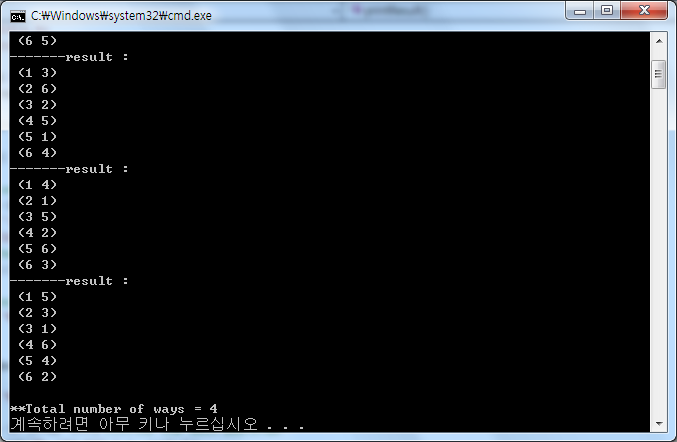
N = 4 🡪 2 ways



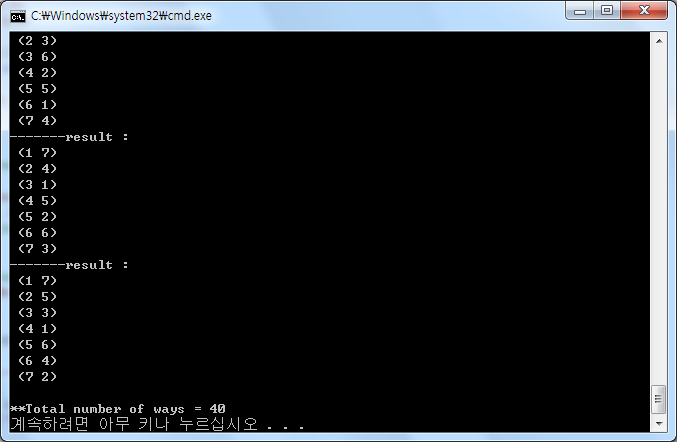
N = 5 🡪 10 ways



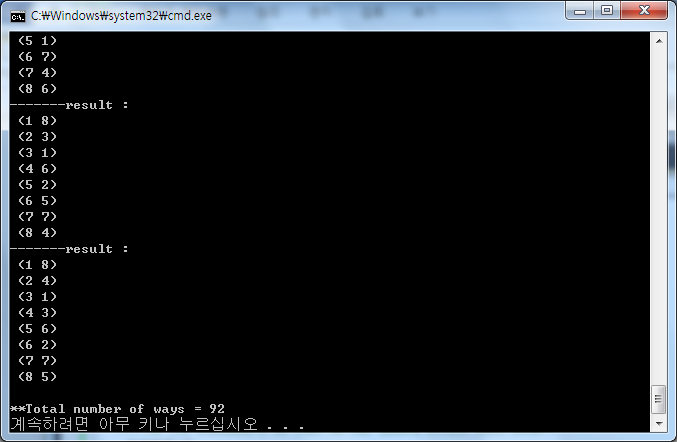
N = 6 🡪 4 ways



N = 7 🡪 40 ways



N = 8 🡪 92 ways



**Problem 5 : Servicing Stations**

***Code***

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

typedef struct point{ //struct of point

float x;

float y;

}Point;

void initializePoint(Point p[], int length){ //Receive points from user

int i;

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

scanf("%f %f", &p[i].x, &p[i].y);

//All coordinates will have values less than 40,000 and be non-negative

while(p[i].x > 40000 || p[i].x < 0 || p[i].y > 40000 || p[i].y < 0){

printf("Too Big number !! Re input \n");

scanf("%f %f", &p[i].x, &p[i].y);

}

}

}

float calculate(Point A, Point B){ //Calculate distance of two points

float res;

res = sqrt((float)(((A.x - B.x) \* (A.x - B.x)) + ((A.y - B.y) \* (A.y - B.y))));

return res;

}

void sort(Point p[], int left, int right, int ver){ //Sorting each distance

int i, j;

Point temp;

if (ver == 1)

{

for (i = left; i < right; i++)

{

for (j = i; j < right; j++)

{

if (p[i].x > p[j].x)

{

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

}

else

{

for (i = left; i < right; i++)

{

for (j = i; j < right; j++)

{

if (p[i].y > p[j].y)

{

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

}

}

float Brute\_force(Point p[], int left, int right){ //Calculate distance of each section

int i,j;

float res = 0;

for (i = left; i < right; i++)

{

if (i == left)

res = calculate(p[i], p[i + 1]);

else if (res > calculate(p[i], p[i + 1]))

res = calculate(p[i], p[i + 1]);

}

return res;

}

float Closset\_pair(Point p[], int left, int right){ //Calculate closset\_pair

Point \* stack;

int count = 0;

int q, i;

float dl;

float dr;

float d;

stack = (Point\*)malloc(sizeof(Point)\* (right - left));

sort(p, left, right, 1); // Sort of X line

if (right - left < 3) // Number is less than 3

return Brute\_force(p, left, right);

q = (left + right) / 2;

dl = Closset\_pair(p, left, q);

dr = Closset\_pair(p, q, right);

d = ((dl > dr) ? dr : dl);

for (i = left; i < right; i++) //Push

{

if (p[q].x - d <= p[i].x && p[q].x + d >= p[i].x)

stack[count++] = p[i];

}

sort(stack, 0, count, 2); //Sorting part

for (i = 0; i < count - 1; i++)

{ //calculate distance

if (calculate(stack[i], stack[i + 1]) < d)

d = calculate(stack[i], stack[i + 1]);

}

return d;

}

int main(void){

Point \* p; //Declear of Points

Point \* stack; //Stor the stack

float distance; //Last distance

while(1){

int length=31; // first Length

//The input set starts with an integer N(0<=N<=30), which denotes the number of points in this set.

while(length>30 || length<0){

printf("Enter the number of points : ");

scanf("%d", &length);

if(length==0)

exit(0);

}

//Allocate space

p = (Point\*)malloc(sizeof(Point)\* length);

stack = (Point\*)malloc(sizeof(Point)\* length);

//initialize points

initializePoint(p, length);

//Find closset points

distance = Closset\_pair(p,0,length);

//Check distance

if(distance > 10000)

printf("It is Infinity ! \n");

else

printf("Distance result = %.2f\n",distance);

}

}

***Print***

