KATHMANDU UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Dhulikhel, Kavre.

Mini Project of COMP 314,

N_Queen Problem with Backtracking

recursive and iterative

SUBMITTED TO:

Mr. Bal Krishna Bal

SUBMITTED BY:

Sandip Sahani(40), Iksha Gurung(18), Milan Thapa(52)

Input

A N x N chess board and N queens. Queens can move horizontally, vertically and diagonally.

Output

A chessboard with N queens arranged on the board in such a way that no two queens can attack each other.

Process

To solve this problem, we use backtracking each time we got bounded by constraints i.e, when we have no more place valid to place the new queens in the board. Also, before placing a new queen to the board, at first the position in the board should be throughly checked and if the place is valid then we could place a new queen on the board.

To find all the valid solutions we implemented the following pseudo-codes.

Pseudo code

1. Iterative

```
Algorithm N_QUEEN_BACKTRACKING(n)

{

       j:=1;

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

{

       while(j!=0) do

       {

              if(j<n and place(k,j))

              {

                     if(k==n)then write the current co-ordinate

                     j:=j+1;

}
```

```
              else
                     j:=j-1;
}
}
}
```

2. Recursive

```
void NqueenLogic(int qu[], int n){
       int i;
       if n == N
       print the queens on arrya qu[]
       else
              for i  to N{
                     qu[n] = i;
//check the new place if it is safe
                       if(new queen could be placed )
       //recursively call increaing the value of new row
NqueenLogic(qu, n + 1);
              }
       }
}
```

Source Codes

1. Iterative

```
/*-------Backtracking NQueens Problem using Iterative
Approach-------------*/

#include <stdio.h>

#include<time.h>

#define N 4

int a[N][N];

int check(int,int);

FILE *outputFile;

main()

{

 int n=N,row,col,prev,flag,count=0,i,j;

 clock_t start=clock();

 outputFile=fopen("iterOutput.txt","w");

 fprintf(outputFile,"******* %d Queen
Puzzle**********\n\n",N);

 for(row=0;row<N;row++)

 {

  for(prev=0;prev<N;prev++)

  {

   flag=0;

   if(a[row][prev]==1)

   {

    a[row][prev]=0;

    flag=1;

    break;

   }

  }

  if(flag==0)
```

```
  {
   prev=-1;
  }

flag=0;
for(col=prev+1;col<N;col++)
  {
   if(check(row,col))
   {
    a[row][col]=1;
    flag=1;
    break;
   }
  }
  if(row==0&&flag==0)
                     break;
  else if(col==N)
       row=row-2;
  else if(row==N-1&&flag==1)
  {
   count++;

   fprintf(outputFile,"\t \t \t  %dsolution found \n ",count);
  for(i=0;i<N;i++){
                  for(j=0;j<N;j++)
                  {
                                 if(a[i][j]==1)

fprintf(outputFile,"Q ");
                                 else
```

```
fprintf(outputFile,"* ");

                    }

           fprintf(outputFile,"\n");

   }

  row=row-1;



  }

 }

fprintf(outputFile,"\n\nTime elapsed: %f secs\n",
((double)clock() - start)/CLOCKS_PER_SEC);

fprintf(outputFile," \ntotal solutions are %d",count);

// system("pause");

fclose(outputFile);

}



int check(int row,int col)

{

 int i;

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

 {

  if((a[row-1-i][col]==1)||(a[row-1-i][col-1-i]==1&&(col-i-
1)>=0)||(a[row-1-i][col+1+i]==1&&(col+1+i<=N-1)))

  return 0;

  }

  return 1;

}
```

2. Recursive

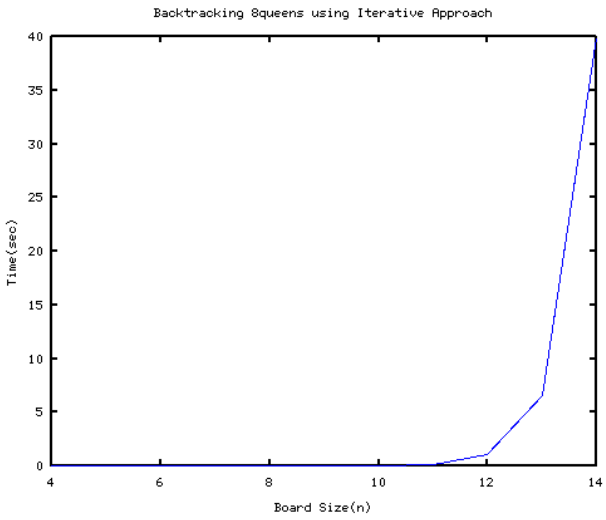
```
/*---Backtracking NQueens Problem using Recursive
Approach-----------*/

#include<stdio.h>

#include<time.h>



#define TRUE 1

#define FALSE 0

#define N 8

int counter=0;

int placeQueen(int q[], int n);

void queenLogic(int q[],int n);

void printQueens(int q[]);



FILE *theFile;

/*-------------------------------------------------------------*/

int main(){

       clock_t start;

       int i;

       int queen[N];



       theFile = fopen("output.txt","w");

       fprintf(theFile,"******* %d Queen
Puzzle**********\n\n",N);
```

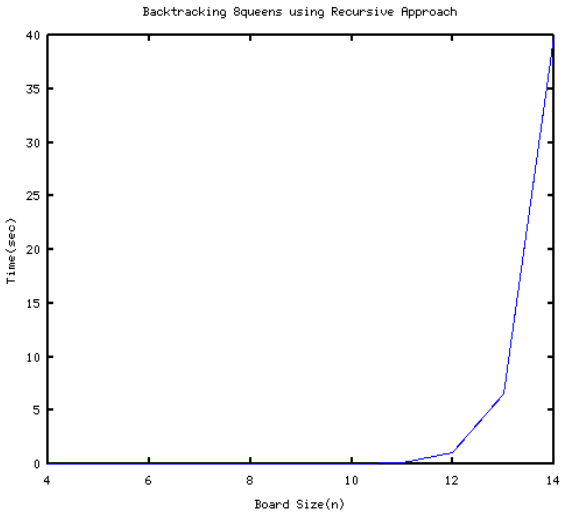
```
	start=clock();

	queenLogic(queen,0);

	fprintf(theFile,"\n\nTime elapsed: %f secs\n",
((double)clock() - start)/CLOCKS_PER_SEC);



	fprintf(theFile,"with %d solutions. :):)",counter);

	fprintf(theFile,"\n");

	fclose(theFile);

	return 1;

}

/*-------------------------------------------------------------*/

void queenLogic(int qu[], int n){



	int i;

	if (n == N){

		printQueens(qu);

		counter++;

		//printf("\n%d solution found...",counter);

		}

	else {

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

			qu[n] = i;

			if (placeQueen(qu,n))

				queenLogic(qu, n + 1);

		}

	}

}

/*-------------------------------------------------------------*/

int placeQueen(int q[], int n){
```

```
	int i;
	for (i = 0; i < n; i++) {
		if (q[i] == q[n])
			return FALSE; // same column
		if ((q[i] - q[n]) == (n - i))
			return FALSE; // same major diagonal
		if ((q[n] - q[i]) == (n - i))
			return FALSE; // same minor diagonal
		}
	return TRUE;
}
/*-----------------------------------------------------------*/
void printQueens(int q[])
{
	int i,j;
	for ( i = 0; i < N; i++) {
		for ( j = 0; j < N; j++) {
			if (q[i] == j)
				fprintf(theFile,"Q ");
			else
				fprintf(theFile,"* ");
		}
		fprintf(theFile,"\n");
	}
	fprintf(theFile,"\n");
}
```

Graphs

The output response of the problem could be visualized using the graph plotted with time vs input size.





Conclusion

After looking to the output response graph above, we concluded that the time complexity obtained from our previous pencil paper analysis i.e, O(p(n)2^n) or O(q(n)n!) (where p(n) and q(n) are the polynomials in n.) is similar to our actual recursive and iterative implementaions.

References

- Horowitz, E., Sahni S., Rajasekaran S., “Fundamentals of Computer Algorithms”, University Press, Second Edition
- http://en.wikipedia.org/wiki/Eight_queens_puzzle
- http://www.c4learn.com/c-progran-to-implement-n-queens-problem.html
- http://ucancode.wordpress.com/2010/12/23/solution-to-n-queens-problem