BLG 381E

Advanced Data Structure

2011 Summer School

Report of Homework 1

Date of Submission : 05.07.2011

Student Name : Ozan Arkan Can

Student Number : 040090573

Instructor : Arzucan Özgür Türkmen

Teaching assistant: Ahmet Aycan Atak

CRN: 30621

INDEX

Introduction

The purpose of this homework is calculating running times and representing with asymptotic notations for given pseudo codes. The other work is implementing a recursive algorithm for solution of n-queen problem and calculating running times and finding this algorithm big-Oh notation.

Defining Function For Running Time (Q_1)

Note that for analysis C_i is cost of line.

a)First code

```
1) for i=1 to N1
2)      m <- i*i
3)       for j=1 to N2
4)              m<- m+j
5)      endfor
6)      for k=i to N3
7)              m <- m-1
8)      endfor
9) endfor
```

1) C_1*N_1

2) $C_2*(N_1-1)$

3) $C_3*(N_1-1)*N_2$

4) $C_4*(N_1-1)*(N_2-1)$

6) $C_5*(N_1-1)*N_3$

7) $C_6*(N_1-1)*(N_3-1)$

$$T(N_1, N_2, N_3) = N_1(AN_2 + BN_3 + C) - DN_2 - EN_3 + F$$

b)Second code

Note that for this code p is probability of executing if block. (1-p) is probability of executing else block.

```
1) sum <- 0
2) if N1>N2
3)       for i=N2 to N1
4)              sum <- sum + i^3
5)       endfor
6) else
7)       for i=N2 down to N1
8)              sum <- sum + i^3
9)              sum <-sum – i
10)      endfor
11) endif
```

1) C_1

2) C_2

3) $C_3*(N_1- N_2+1)$

4) $C_4*(N_1- N_2)$

7) $C_5*(N_2- N_1)$

8) $C_3*(N_2- N_1)$

9) $C_6*(N_2- N_1)$

$$T(N_1, N_2) = \begin{cases} A + B(N1 - N2) , & p \\ C + D(N2 - N1) , & 1 - p \end{cases}$$

c)Third code

Note that S is number of iteration of inner loop (line 7)

1) N1 <- size of integer array ‘arr’

2) vals: empty integer array

3) inds: empty integer array

4) foreach element of arr

5) N2 <- size of integer array ‘vals’

6) i <- 1

7) for i=1 to N2

8) if vals[i]<arr[i]

9) break

10) endfor

11) insert arr[i] at vals[i]

12) insert i at inds[i]

13) endforeach

1) C_1

4) $C_2*(N_1+1)$

5) $C_3*(N_1)$

6) $C_4*(N_1)$

7) $C_5*(N_1*S)$

8) $C_6*(N_1*S)$

11) $C_7*(N_1)$

12) $C_8*(N_1)$

$$T(N_1, N_2) = N_1(A + BS) + C$$

Big-Oh Notation Of Functions In Q_1 (Q_2)

a)Analysis for first code

$$T(N_1, N_2, N_3) = N_1(AN_2 + BN_3 + C) - DN_2 - EN_3 + F$$

In this function $AN_1N_2 + BN_1N_3$ is the element of function that grow faster than the others. So this function can represent with

$$O\ (N_1N_2 + N_1N_3).\ T(N_1, N_2, N_3) = O\ (N_1N_2\ +\ N_1N_3)$$

Definition of big-oh:

$$0 \leq f(n) \leq cg(n)$$

$$0 \leq N_1(AN_2 + BN_3 + C) - DN_2 - EN_3 + F \leq c * (N_1N_2\ +\ N_1N_3)$$

$$N_1 \geq 1, N_2 \geq \frac{D}{A}, N_3 \geq \frac{E}{B} \Rightarrow\ c = A + B + F$$

b) Analysis for second code

$$T(N_1, N_2) = \begin{cases} A + B(N_1 - N_2)\ , & p \\ C + D(N_2 - N_1)\ , & 1 - p \end{cases}$$

For worst-case analysis p must be defined. I assume that $(N_1 - N_2)$ and $(N_2 - N_1)$ abstract values are equal. And if we compare if and else block , we can see that else block include if block. So, else block runs slower than if block. In conclusion, p must be zero and for all inputs else block is executed. This reason this function can represent with $O\ (N_2 - N_1)$.

$$0 \leq C + D(N_2 - N_1) \leq c * (N_2 - N_1)$$

$$N_1 \geq 1, N_2 \geq N_1 \Rightarrow\ c = C + D$$

c) Analysis for third code

$$T(N_1, N_2) = N_1(A + BS) + C$$

In this code, inner loop that in line 7 runs different times for every iterations of outer loop. S represents this running time. In worst-case inner loop execute N_2 times, so S = N_2.

$$T(N_1, N_2) = N_1(A + BN_2) + C$$

In this function BN_1N_2 is element that grows faster than the others.

$$T(N_1, N_2) = O(N_1N_2)$$

$$0 \leq N_1(A + BN_2) + C \leq c * N_1N_2$$

$$N_1 \geq 1, N_2 \geq 1 \Rightarrow c = A + B + C$$

Best, Worst and Average Running Times (Q_3)

Note that for this code p is probability of executing if block. (1-p) is probability of executing else block.

```
1) var flag: boolean
2) …
3) …  // some code here
4) …  // flag is modified several times
5) …
6) var sum: integer
7) sum <- 0
8) if flag is true
9)       for i:1 to N
10)              sum <- sum + i^3
11)      endfor
12)      flag <- true
13) else
14)      for i:1 to N
15)              for j:i to N
16)                      sum <- sum + i^3
17)              endfor
18)      endfor
19)      flag <- false
20) endif
```

7) C_1

8) C_2

9) $C_3*(N)$

10) $C_4*(N-1)$

12) C_5

14) $C_3*(N)$

15) $C_6*(N^2+N-2)/2$

16) $C_7*(N^2-N)/2$

19) C_8

$$T(N) = \begin{cases} AN + B & , \quad p \\ CN^2 + DN + E & , \quad 1 - p \end{cases}$$

We can see that if block runs faster than else block for this code from T(N). Value of p defined for this situation while doing asymptotic analysis.

In best case, p must be one, so if block will execute.

$$T(N) = AN + B = \Omega(N)$$

In average case, we assume that probability of executing if block and else block are equal, because there is no information about distribution. p must be 0.5 by this reason.

$$T(N) = \frac{AN+B+CN^2+DN+E}{2} = \Theta(N^2)$$

In worst case, else block must be executed for every inputs and this will occur when p is zero.

$$T(N) = CN^2 + DN + E = O(N^2)$$

N-Queen Problem Solution Implementation (Q_4)

In chess, a queen can move as far as she pleases, horizontally, vertically, or diagonally. The n Queen's problem asks how to place n queens on nxn chess board so that none of them can hit any other in one move and what is the number of total solutions.

Recursive Algorithm For Solution

On chess board, there can be only one queen in a row. Potential solutions are just permutation of columns numbers. All permutations are not solution but solution is a permutation of columns number.

Algorithm steps are just like following:

1) Get one of permutation of columns number.

2)Set first queen at a column that number from permutation.

3)Save attacked places on board.

4)For next queen look is board proper for column by next number from permutation.

If it is proper and it is nth queen. This is a solution and increase number of solution. It is not last queen set queen and update attacked places and return 4.

Else, clean board and return 1.

5)Repeat this algorithm n! times.

Permutation Function And Its Analysis

```
void nQueen::permutation(short int *numbers, short int k, short int m)
{
   if(k == m)
   {
      bool breakControl = false;
      for(short int i = 0; i <= m; i++)
      {
         if(isProper(numbers[i], i))
         {
            updateThreatenedPlaces(numbers[i], i);
         }
         else
         {
            breakControl = true;
            break;
         }
      }
      if(!breakControl)
      {
         numberOfSolutions++;
      }
      cleanboard();
   }
   else
   {
      for(short int i = k; i <= m; i++)
      {
         swap(numbers[k], numbers[i]);
         permutation(numbers, k+1, m);
         swap(numbers[k], numbers[i]);
      }
   }
}
```

This recursive function execute recursively until parameters k and m equal. If block condition is a function and its running time is O(1).

```
bool nQueen::isProper(short int row, short int column)
{
    return !board[row][column];
}
```

For loop that in if block execute n times (in worst-case). And function updateThreatenedPlaces(numbers[i], i) execute n times. This function running time is O(n). (5 for loops with n iterations.)

```
void nQueen::updateThreatenedPlaces(short int row, short int column)
{
    for(short int i = 0; i < numberOfQueens; i++)
    {

    //to left upper corner
    for(short int i = row - 1, j = column - 1; i >= 0 && j >= 0; i--, j--)
    {

    //to right bottom corner
    for(short int i = row + 1, j = column + 1; (i < numberOfQueens) && (j < numberOfQueens); i++, j++)
    {

    // to left bottom corner
    for(short int i = row + 1, j = column - 1; i < numberOfQueens && j >= 0; i++, j--)
    {

    //to right upper corner
    for(short int i = row - 1, j = column + 1; i >= 0 && j < numberOfQueens; i--, j++)
    {
}
```

Else block has a for loop with m(n) – k iterations. And it calls every iteration two times swap function and one time itself(permutation). Running time of swap function is O(1).

```
void nQueen::swap(short int& x, short int& y)
{
    if(x != y)
    {
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;
    }
}
```

So total running time of permutation function:

$T(n) = nT(n-1) + O(n^2)$ (updateThreatenedPlaces)+ $O(1)$

$T(n-1) = (n-1)T(n-2) + O(n^2) + O(1)$

$T(n-2) = (n-2)T(n-3) + O(n^2) + O(1)$

$T(n-3) = (n-3)T(n-4) + O(n^2) + O(1)$

…..

….

…

$T(n-(n-1)) = (n-(n-1))T(n-n) + O(n^2) + O(1)$

$T(1) = T(0) + O(n^2) + O(1) = O(n^2) + O(1)$

$T(n) = 1*2…*(n-1)*n*T(1) + O(n^2) + O(1)$

$T(n) = n!*T(1) + O(n^2) + O(1)$

$T(n) = n!*(O(n^2) + O(1)) + O(n^2) + O(1)$

$T(n) = O(n!* n^2)$

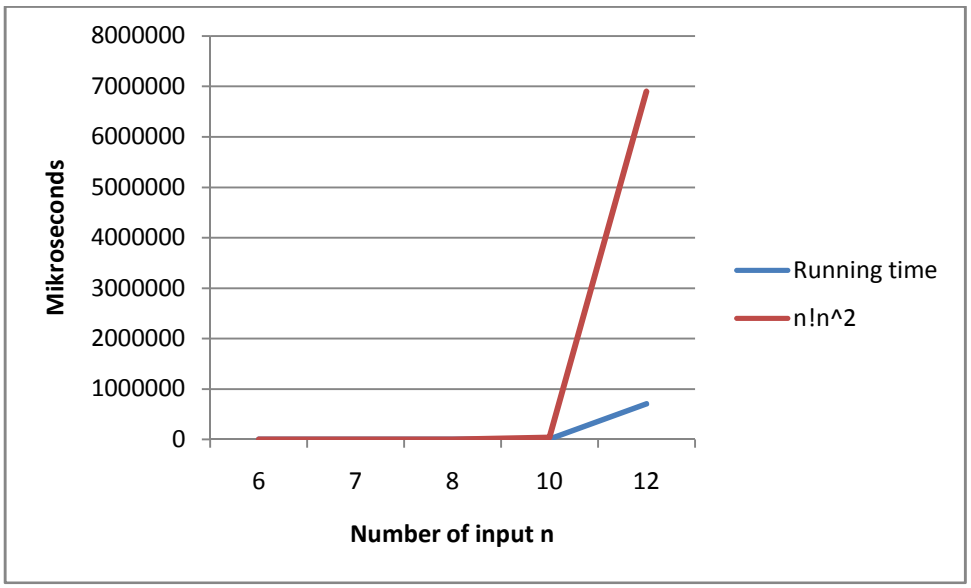
Running time for number of input n:

```
C:\Users\OZAN\Desktop>homework1.exe 8
Number of inputs: 8
Running time: 31 miliseconds
Number of solutions: 92
```

```
C:\Users\OZAN\Desktop>homework1.exe 10
Number of inputs: 10
Running time: 4009 miliseconds
Number of solutions: 724
```

```
C:\Users\OZAN\Desktop>homework1.exe 12
Number of inputs: 12
Running time: 700631 miliseconds
Number of solutions: 14200
```

For n = 14,16 and 18 program did not execute, because running time is very big for this input size. Here is the graphic of comparison between running time of program and running time function (T(n)).



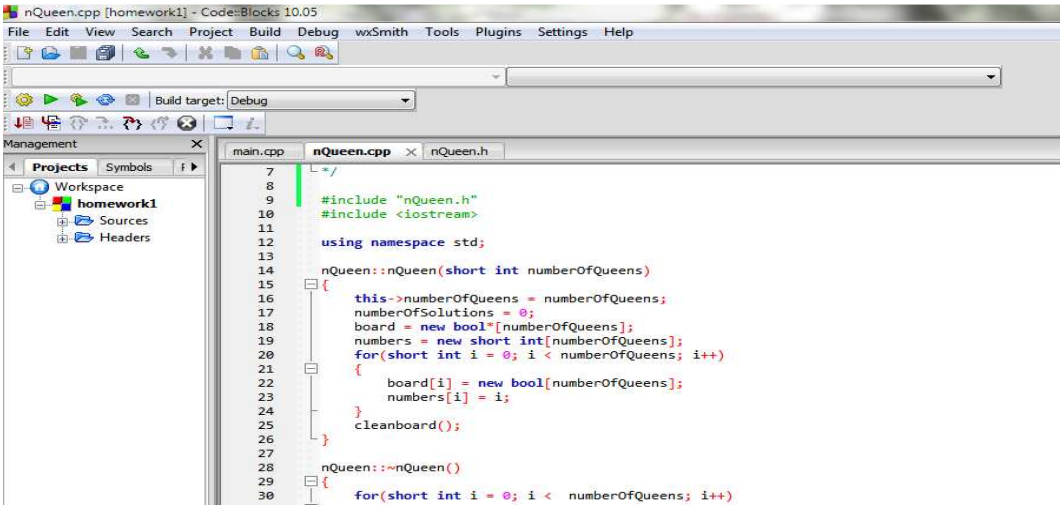
Class UML Diagram

nQueen
-numberOfQueens : short int -numberOfSolutions : unsigned int -board : bool** -numbers : short int*
+nQueen() +~nQueen() +cleanboard() : void +permutation(içeri numbers : short int*, içeri k : short int, içeri m : short int) : void +updateThreatenedPlaces(içeri row : short int, içeri column : short int) : void +isProper(içeri row : short int, içeri column : short int) : bool +swap(içerivedışarı x : short int, içerivedışarı y : short int) : void +showResult() : void +getList() : short int*

Development and Operating Environments

MS Windows

The Code::Blocks IDE has been used to write source code, compile and run the code.



Unix

The source code has been also copied to Unix, then compiled and tested with the
GNU C++ Compiler. The following is the commands used:
To compile : g++ main.cpp nQueen.cpp nQueen.h –o homework1
To run : ./homework1 8

```
[cano@ssh ~]$ g++ main.cpp nQueen.cpp nQueen.h -o homework1
[cano@ssh ~]$ ./homework1 8
```

Conclusion

In this homework , calculating running time a code, making asymptotic analysis , big-Oh, big-Theta, big-Omega notations are learned.

How important implementing a fast algorithm for running times.

The growth of running times function is learned.