Project Report

CS-5384 Logic for Computer Scientists

Satisfiability Test of Clauses and its Applications

Edwin Sahil Samuel Kathi - R11800320
Vamsi Krishna Muppala - R11770853
Vandana Priya Muppala - R11751885

Problem Definition:

N-Queens Problem:

The problem is based on how a queen can attack in chess. Queen can attack vertically, horizontally and diagonally. The problem is, on an N*N chessboard, we have to place N queens so that no two queens should be in the same row, same column and on the same diagonal. This can be converted to a boolean satisfied expression and is satisfied if there is a queen on the cell which makes it true, and false if the cell is empty. Since no queens can attack each other, we use a to show the position of the N kings binary matrix (N*N).

To solve the n-queens problem, we use the miniSat SAT solver, which takes the CNF file as input and gives an output that satisfies n-queens.

SAT Solver:

A SAT solver is used to determine if a given propositional logical formula is satisfactory Or not. It is a tool that takes a text file with CNF formula as input and outputs successfully if the formula (which is built from variables and connectors) can be made true by assigning truth it is worth its variables or unsatisfactory if it is not.

CNF (Conjunctive Normal Form):

In propositional logic, a formula is called conjunctive normal form (CNF) if it expresses as conjunctions of one or more clauses with AND OR (possibly negated, ¬) variables are called literals.

N-Queens as CNF grammar:

N-Queens problem is approached as a SAT problem. In the N-Queens problem, the queens should be placed in such a way that none of the queens could attack each other. The algorithm generates CNF formatted literals that act as an input to SAT Solver, to decide if the problem is satisfiable or not.

The constraints in n-queens problems are:
Only one queen can be present in each row
Only one queen can be present in each column
At most one queen can be present in the diagonal

Let us consider 4-queens problem:
Assign a 2D matrix for of size 4*4. The matrix would look like:

11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

Every row will have one queen:
$P_{11} \vee P_{12} \vee P_{13} \vee P_{14}$

If the queen is placed in column 3 of row 1, then the propositions would be :
$P_{13} \rightarrow \neg P_{11} \wedge \neg P_{12} \wedge \neg P_{14}$
Since the queen is placed in column 3 of row 1, then the true value is placed in column 3. The
CNF for the above proposition would be :
$P_{13} \rightarrow \neg P_{11}$
$P_{13} \rightarrow \neg P_{12}$
$P_{13} \rightarrow \neg P_{14}$
$\neg P_{11} \rightarrow \neg P_{12}$
$\neg P_{11} \rightarrow \neg P_{14}$
$\neg P_{12} \rightarrow \neg P_{14}$

The propositions could be further be simplified as:
$(P_{13} \vee \neg P_{11}) \wedge (P_{13} \vee \neg P_{12}) \wedge (P_{13} \vee \neg P_{14}) \wedge (\neg P_{11} \vee \neg P_{12}) \wedge (\neg P_{11} \vee \neg P_{14}) \wedge$
$(\neg P_{12} \vee \neg P_{14})$
Similarly the proposition for the positive diagonal would be:
$P_{11} \vee P_{22} \vee P_{33} \vee P_{44}$
If the queen is placed in column 2 of row 2, then the propositions would be :
$P_{22} \rightarrow \neg P_{11} \wedge \neg P_{33} \wedge \neg P_{44}$
$P_{22} \rightarrow \neg P_{11}$
$P_{22} \rightarrow \neg P_{33}$
$P_{22} \rightarrow \neg P_{33}$
$\neg P_{11} \rightarrow \neg P_{33}$
$\neg P_{11} \rightarrow \neg P_{44}$
$\neg P_{33} \rightarrow \neg P_{44}$
The propositions could be further be simplified as:
$(P_{22} \vee \neg P_{11}) \wedge (P_{22} \vee \neg P_{33}) \wedge (P_{22} \vee \neg P_{44}) \wedge (\neg P_{11} \vee P_{33}) \wedge (\neg P_{11} \vee \neg P_{44}) \wedge (\neg P_{33} \vee$
$\neg P_{44})$

The algorithm would generate CNF in the form of integers, so that it could be interpreted by the miniSAT Solver and. Based on the clauses in the CNF file, miniSAT solver tells us whether the given clauses are satisfiable or not. The miniSAT Solver would accept the output of the algorithm and indicate whether the generated output would be satisfactory or not. To be satisfiable , the proposition sequence with positive and negative values where a positive proposition indicates there is a queen in that position otherwise there is no queen.

Pseudo Code:

```
Begin
Let n be the input which is given by the user (n is the number of queens)
Given,chess board size= n*n
If (n<1) // if the input given by the user has no queen
        Return error message and terminate // N cannot be less than 1
For each row from 0 to n // We check that there is only one queen in the row
        for column in range (0,n).
            Here we map the position and get a CNF clause so that only one of the variables is true.
For each column from 0 to n // We check that there is only one queen in the column
        for each row in range (0, n)
            Here we map the position and get a CNF clause so that only one of the variables is true.
        // Now we will check the diagonal constraints.
For each row ranging from n-1 to -1 (with a difference of -1) // One queen constraint negative diagonal from the left
        for x in range (0, n-i)
For each column ranging from(1, n) // One queen constraint in negative diagonal from top
        for x in range(0, n-column)
For each row in from (n-1,- 1,-1) // One queen constraint in positive diagonal from right
        for x in range (0, n-i)
For each column from (n-2, -1,-1) // One queen constraint in positive diagonal from top
        for x in range (0, column+1)
Print and the output is written into a CNF file.
End
```

Steps by Step Process:

1. Install MiniSAT, a minimalistic, open-source SAT solver, from the given website - http://minisat.se/Main.html for executing the given project, we are using macOS.
2. Once we install Minisat on the system using homebrew using the given command: $ brew install minisat
3. We compile the java code in the system terminal using the command javac filename.java
4. We run the code in the system terminal using java filename.
5. The code once run generate input.cnf file
6. We run the command: minisat input.cnf output.txt
7. The output shows for a given number of queens if it is satisfiable or unsatisfiable.The result is also written in the output.txt file.

Output:

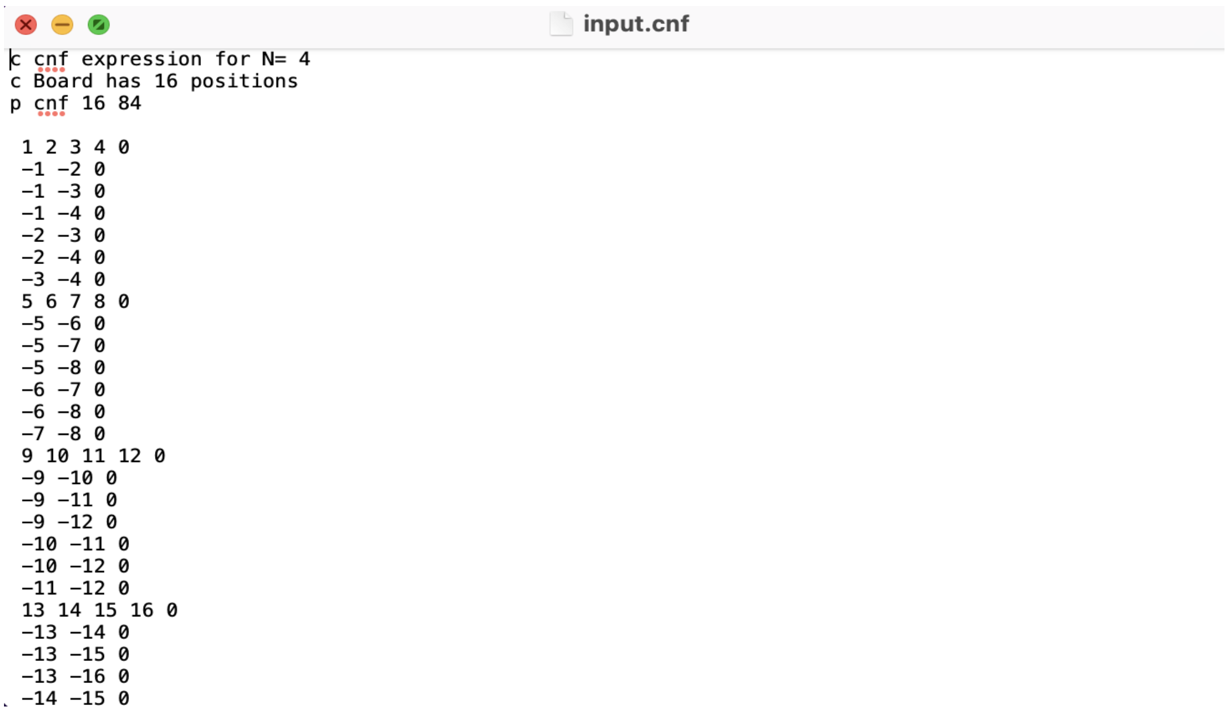
(a) For Satisfiable Output:

(i) Input, n=4 // the number of queens in the Java script which generates an output for 4 Queens in CNF format.

From the input file we could interfere that the board has 16 positions

p cnf 16 84.

Then the Java script generates an input.cnf file with the CNF of the given N-Queens(Given n=4)



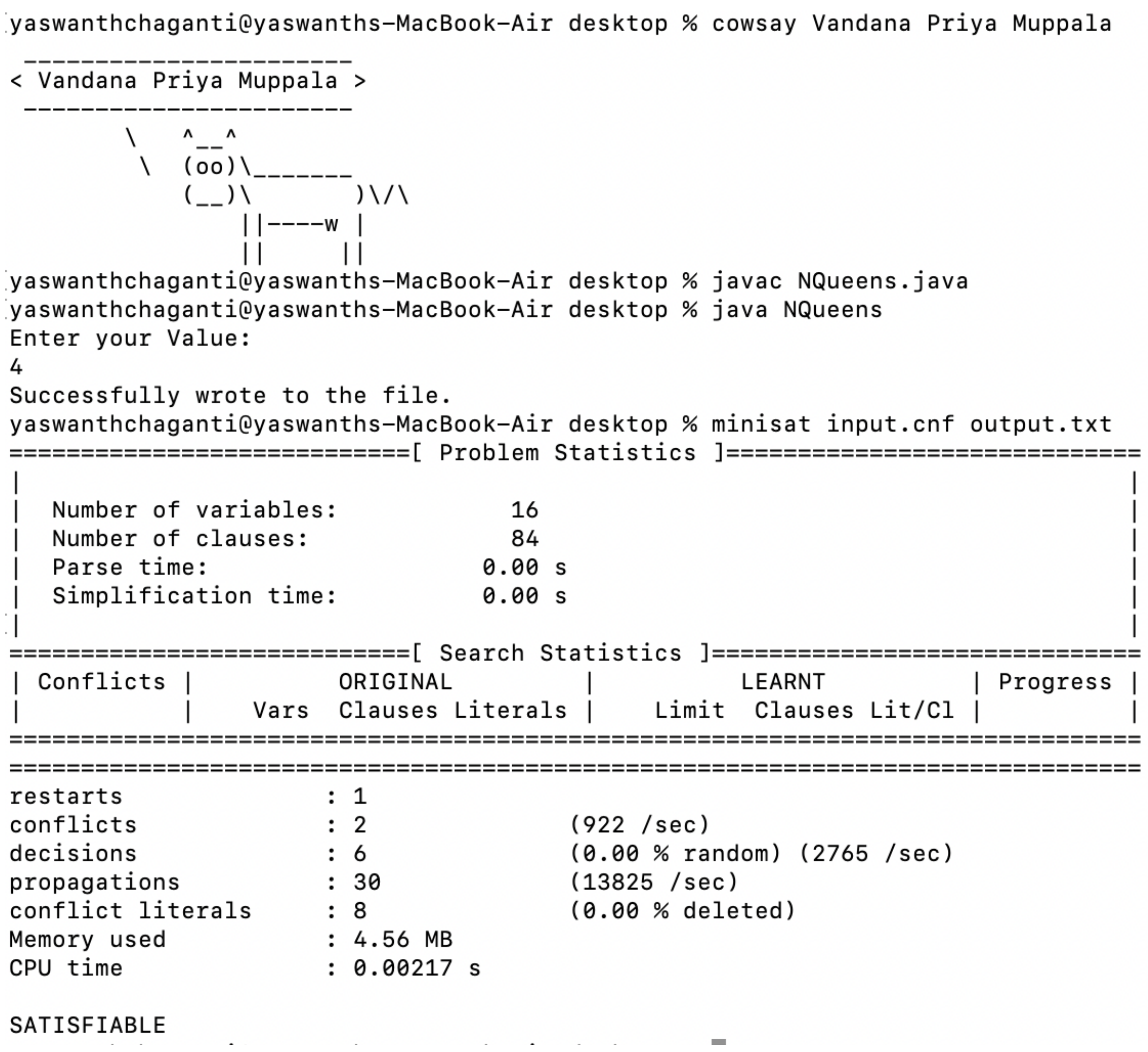
```
input.cnf
c cnf expression for N= 4
c Board has 16 positions
p cnf 16 84

 1 2 3 4 0
 -1 -2 0
 -1 -3 0
 -1 -4 0
 -2 -3 0
 -2 -4 0
 -3 -4 0
 5 6 7 8 0
 -5 -6 0
 -5 -7 0
 -5 -8 0
 -6 -7 0
 -6 -8 0
 -7 -8 0
 9 10 11 12 0
 -9 -10 0
 -9 -11 0
 -9 -12 0
 -10 -11 0
 -10 -12 0
 -11 -12 0
 13 14 15 16 0
 -13 -14 0
 -13 -15 0
 -13 -16 0
 -14 -15 0
```

(ii) We then used the input.cnf file as the input for the Minisat ,which tests the satisfiability of the CNFs .

Command: minisat input.cnf (input file) output.txt (output file).

(iii) Satisfiability: We know that an expression is satisfactory if there is an assignment to the variables that makes it true. It's hard to manually check each generated clause, so we're using a minisat that gives the result as SATISFIABLE(SAT) if true or UNSATISFIABLE(UNSAT) if not ,in either case.

(iv) The output of the minisat shows the given CNF for 4 queens is SATISFIABLE.

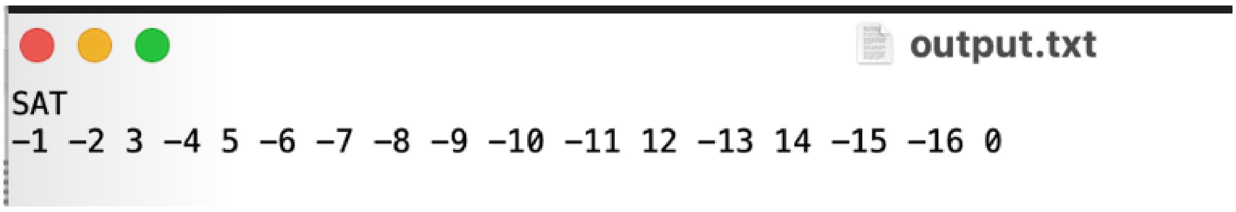


```
yaswanthchaganti@yaswanths-MacBook-Air desktop % cowsay Vandana Priya Muppala
 _______________________
< Vandana Priya Muppala >
 -----------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
yaswanthchaganti@yaswanths-MacBook-Air desktop % javac NQueens.java
yaswanthchaganti@yaswanths-MacBook-Air desktop % java NQueens
Enter your Value:
4
Successfully wrote to the file.
yaswanthchaganti@yaswanths-MacBook-Air desktop % minisat input.cnf output.txt
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:            16                                         |
|  Number of clauses:              84                                         |
|  Parse time:                   0.00 s                                       |
|  Simplification time:          0.00 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
===============================================================================
restarts              : 1
conflicts             : 2              (922 /sec)
decisions             : 6              (0.00 % random) (2765 /sec)
propagations          : 30             (13825 /sec)
conflict literals     : 8              (0.00 % deleted)
Memory used           : 4.56 MB
CPU time              : 0.00217 s

SATISFIABLE
```

(v) The result in the output.txt file is:

SAT

-1 -2 3 -4 5 -6 -7 -8 -9 -10 -11 12 -13 14 -15 -16 0



```
SAT
-1 -2 3 -4 5 -6 -7 -8 -9 -10 -11 12 -13 14 -15 -16 0
```

(vi) Chess Board representation for SAT Solver Solution :

-1	-2	3	-4
5	-6	-7	-8
-9	-10	-11	12
-13	14	-15	-16

Note: The highlighted variables in the above chess board are the positions of 4 queens such that they are not attacked by each other.

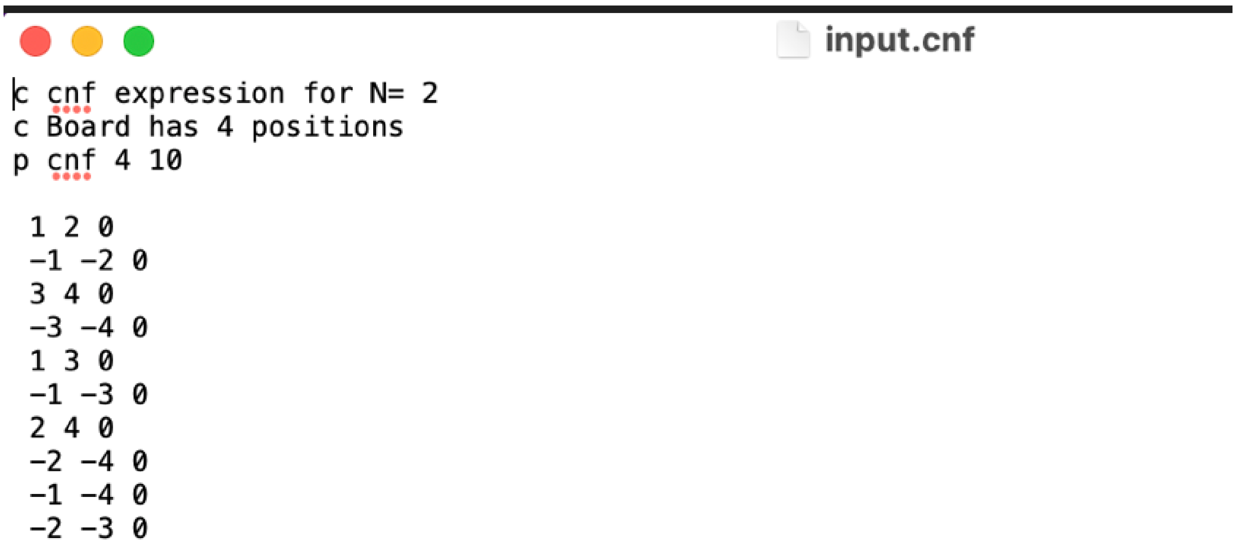
(a) For Unsatisfiable Output:

(i) Input, n=2 // the number of queens in the Java script which generates an output for 2 Queens in CNF format.

From the input file we could interfere that the board has 4 positions

p cnf 4 10.

Then the Java script generates an input.cnf file with the CNF of the given N-Queens(Given n=2)

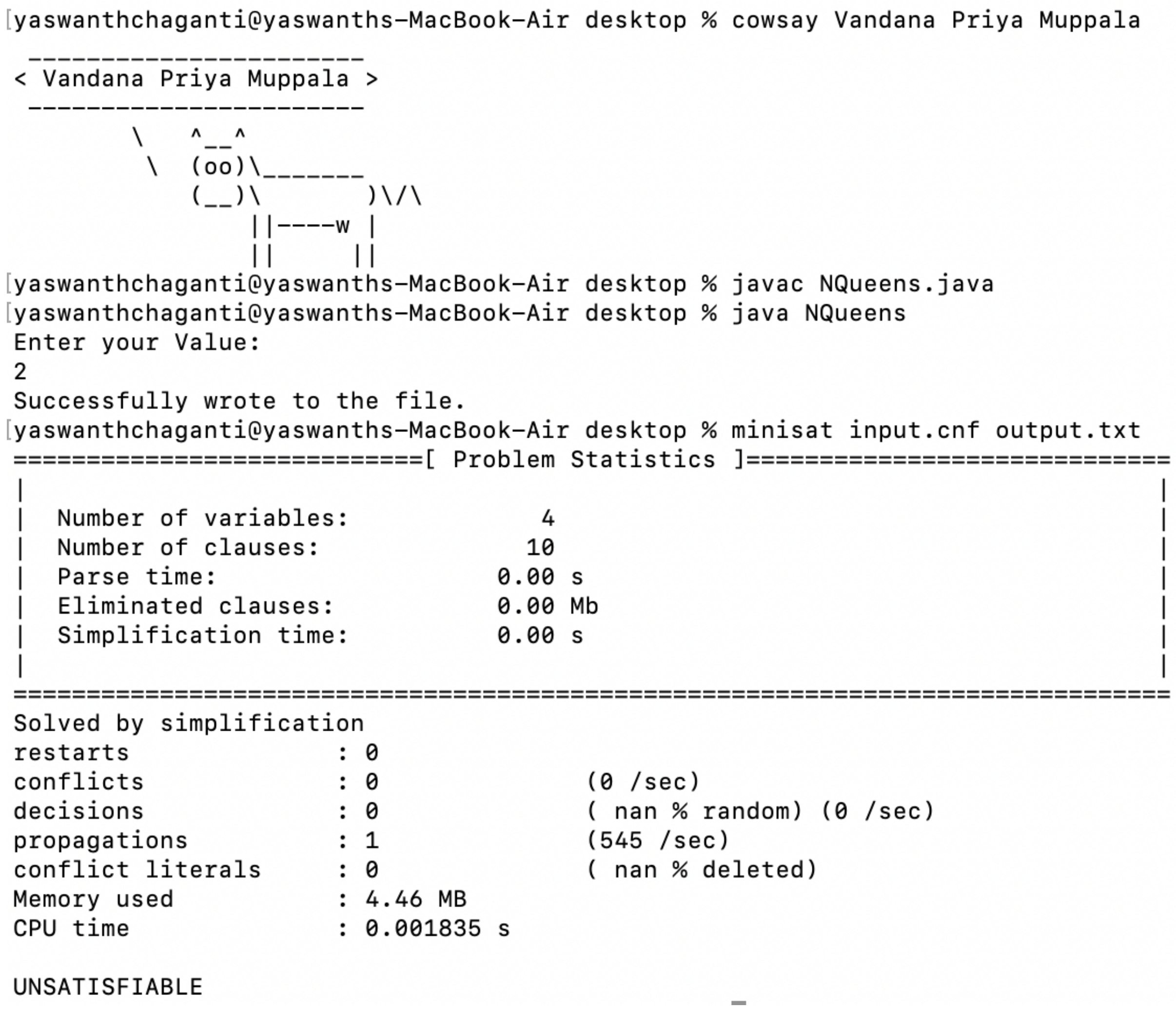


```
c cnf expression for N= 2
c Board has 4 positions
p cnf 4 10

 1 2 0
 -1 -2 0
 3 4 0
 -3 -4 0
 1 3 0
 -1 -3 0
 2 4 0
 -2 -4 0
 -1 -4 0
 -2 -3 0
```

(ii) We then used the input.cnf file as the input for the Minisat ,which tests the satisfiability of the CNFs .

Command: minisat input.cnf (input file) output.txt (output file).

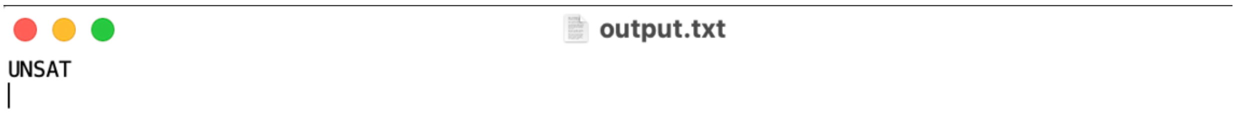
(iii) The minisat shows the CNF to be unsatisfiable as there is no possible solution to fit 2 queens when the board is of the dimensions 2x2 while satisfying the constraints.



```
[yaswanthchaganti@yaswanths-MacBook-Air desktop % cowsay Vandana Priya Muppala
 _______________________
< Vandana Priya Muppala >
 -----------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
[yaswanthchaganti@yaswanths-MacBook-Air desktop % javac NQueens.java
[yaswanthchaganti@yaswanths-MacBook-Air desktop % java NQueens
Enter your Value:
2
Successfully wrote to the file.
[yaswanthchaganti@yaswanths-MacBook-Air desktop % minisat input.cnf output.txt
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:             4                                         |
|  Number of clauses:              10                                         |
|  Parse time:                   0.00 s                                       |
|  Eliminated clauses:           0.00 Mb                                      |
|  Simplification time:          0.00 s                                       |
|                                                                             |
===============================================================================
Solved by simplification
restarts              : 0
conflicts             : 0              (0 /sec)
decisions             : 0              ( nan % random) (0 /sec)
propagations          : 1              (545 /sec)
conflict literals     : 0              ( nan % deleted)
Memory used           : 4.46 MB
CPU time              : 0.001835 s

UNSATISFIABLE
```

(iv) The result in the output.txt file is:

UNSAT



```
UNSAT
```