5.2.2 Algorithm

```
Algorithm Queen(n)
//Problem description : This algorithm is for implementing n
//queen's problem
//Input : total number of queen's n.
        for column ←1 to n do
        {
            if(place(row,column))then
            {
             board[row]column//no conflict so place queen
            if(row=n)then//dead end
          print_board(n)
            //printing the board configuration
            else//try next queen with next position
                Queen(row+1,n)
            }
        }
```

This function checks if two queens are on the same diagonal or not.

Row by row each queen is placed by satisfying constraints.

```
Algorithm place(row,column)
//Problem Description : This algorithm is for placing the
//queen at appropriate position
//Input : row and column of the chessboard
//output : returns 0 for the conflicting row and column
//position and 1 for no conflict.
        for i ← 1 to row-1 do
        { //checking for column and diagonal conflicts
          if(board[i] = column)then
            return 0
        else if(abs(board[i]- column) = abs(i - row))then
              return 0
        }
        //no conflicts hence Queen can be placed
    return 1
```

Same column by 2 queen's

This formula gives that 2 queens are on same diagonal

```
Algorithm Sum_Subset(sum, index, Remaining_sum)
//sum is a variable that stores the sum of all the
//selected elements
//index denotes the index of chosen element from the given
//Remaining_sum is initially sum of all the elements.
//selection of some element from the set subtracts the
//chosen value
//from Remaining_sum each time.
//w[1...n] represents the set containing n elements
//a[j] represents the subset where 1 ≤ j ≤ index
```

```
//sum = Σ_{j=1}^{index-1} w[j]*a[j]
//Remaining_sum = Σ_{j=index}^{n} = w[j].
// w[j] is sorted in non-decreasing order
// For a feasible sequence assume that w[1] ≤ d and
        Σ_{i=1}^{n} w[i] ≥ d
  // Generate left child until sum + w[index] is ≤ d
  a[index]←1
if(sum + w[index] = d) then
  write(a[1...index]) //subset is found
 else if (sum + w[index] + w[index+1] d) then
      Sum_Subset((sum+w[index]), (index+1),(Remaining_sum – w[index]));
            // Generate right child
  if(sum+Remaining_sum – w[index] ≥ d) AND
            (sum+w[index+1]≤ d)) then
  {
      a[index] ← 0
      Sum_Subset(sum, (index+1), (Remaining_sum –
      w[index]));
  }
```

The subset is printed

Search the next element which can