AZA lab 6

1. **Implement the Backtracking algorithm for the n-Queens problem (Algorithm 5.1) on your system, and run it on problem instances in which n = 4, 8, 10, and 12.**

The Backtracking Algorithm for the *n*-Queens Problem

Problem: Position n queens on a chessboard so that no two are in the same row, column, or diagonal.

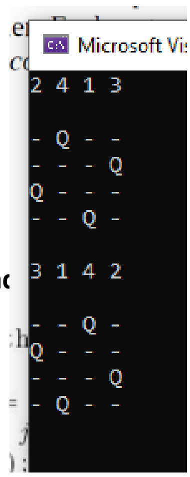
Inputs: positive integer n.

Outputs: all possible ways n queens can be placed on an $n \times n$ chessboard so that no two queens threaten each other. Each output consists of an array of integers *col* indexed from 1 to n, where *col*[*i*] is the column where the queen in the *i*th row is placed.

```
void queens (index i)
{
  index j;

  if (promising(i))
     if (i == n)
        cout << col[1] through col [n];
     else
        for (j = 1; j <= n; j++){        // See if queen in
           col[i + 1] = j;               // (i + 1)st row can be
           queens(i + 1);                // positioned in each of
                                         // the n columns.
        }
}
bool promising (index i)
{
   index k;
   bool switch;

   k = 1;
   switch = true;                  // Check if any queen threatens
   while (k < i &&  switch){       // queen in the ith row.
      if (col[i] == col[k] || abs(col[i] - col[k]) == i - k)
         switch = false;
      k++;
   }
   return switch;
}
```



2. **Visualize the results in 2D. See Examples for n = 4**
3. **Modify the Backtracking algorithm for then-Queens problem (Algorithm 5.1) so that, instead of generating all possible solutions, it finds only a single solution.**

2. **Given an n × n × n cube containing n^3 cells, we are to place n queens in the cube so that no two queens challenge each other (so that no two queens are in the same row, column, or diagonal). Can then-Queens algorithm (Algorithm 5.1, see AZA lab 5 sheet) be extended to solve this problem? If so, write the algorithm and implement it on your system to solve problem instances in which n = 4 and n = 8.**