PSEUDOCODE :-

1. Making the first element of every row equal to 1
2. Because it is a 0 indexed array, if whichever row we intend to output is number 0, then simply we can output the array containing one element that is, 1.
3. Otherwise, we call for the previous row elements and assign , currRowElement[i] = prevRowElement[i-1] + prevRowElement[i];
4. And keep on storing the elements for the present row.

```
   vector<int> currRow;
   // Inserting the first element into the array that is, 1.
   currRow.push_back(1);

  // If we need to produce the row number 0, then simply return the output

   if (rowIndex == 0)
   {
      return currRow;
   }
   vector<int>  prevRowElement = recur(rowIndex - 1);

   for(int i = 1; i <  prevRowElement.size(); i++)
   {
      int currRowElement = prevRowElement[i-1] + prevRowElement[i];
      currRow.push_back(curRowElement);
   }

 // Inserting the last element into the array
   currRow.push_back(1);

   // Returning  the final output row
   return currRow;
}
```

Time Complexity :- O(n^2)

Space Complexity :- O(n)

OPTIMAL APPROACH :-

Generating the binomial coefficients for each term in each row using the previous term by the formula :-

${}^{N}c_{R} = {}^{N}c_{(R-1)} * (N - R + 1) / R;$

PSEUDOCODE :-

```
vector<int> pascalTriangle(int rowNo)
{
   vector<int> curRow;
   curRow.push_back(1);

   rowNo= rowNo - 1;

  // $^{N}c_{R} = {}^{N}c_{(R-1)} * (N - R + 1) / R;$

 // curRow[i-1]
   for (int i = 1; i <= rowNo; i++)
  {
       int curRowElement  = curRow.back() * (rowNo - i + 1) / i;
       curRow.push_back(curRowElement);
   }
   return  curRow;

}
```

TIME COMPLEXITY :- O(N)

SPACE COMPLEXITY :- O(N)