Pascal’s Triangle

Pascal’s triangle is a triangular array of the binomial coefficients. Write a function that takes an integer value n as input and prints first n lines of the Pascal’s triangle. Following are the first 6 rows of Pascal’s Triangle.

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

Method 1 (O(n^3) time complexity)
Number of entries in every line is equal to line number. For example, the first line has “1”, the second line has “1 1″, the third line has “1 2 1″,.. and so on. Every entry in a line is value of aBinomial Coefficient. The value of ***i***th entry in line number *line* is *C(line, i)*. The value can be calculated using following formula.

```
C(line, i)   = line! / ( (line-i)! * i! )
```

A simple method is to run two loops and calculate the value of Binomial Coefficient in inner loop.

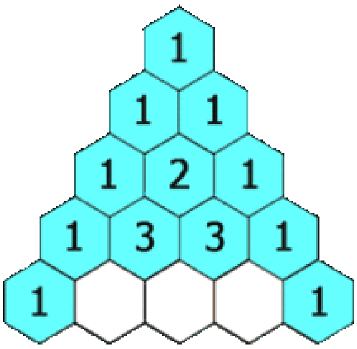
```
// A simple O(n^3) program for Pascal's Triangle
#include <stdio.h>

// See http://www.geeksforgeeks.org/archives/25621 for d
int binomialCoeff(int n, int k);

// Function to print first n lines of Pascal's Triangle
void printPascal(int n)
{
  // Iterate through every line and print entries in it
  for (int line = 0; line < n; line++)
  {
    // Every line has number of integers equal to line n
    for (int i = 0; i <= line; i++)
```

```
      printf("%d ", binomialCoeff(line, i));
    printf("\n");
  }
}

// See http://www.geeksforgeeks.org/archives/25621 for d
int binomialCoeff(int n, int k)
{
    int res = 1;
    if (k > n - k)
       k = n - k;
    for (int i = 0; i < k; ++i)
    {
        res *= (n - i);
        res /= (i + 1);
    }
    return res;
}

// Driver program to test above function
int main()
{
  int n = 7;
  printPascal(n);
  return 0;
}
```

Time complexity of this method is O(n^3). Following are optimized methods.

Method 2(O(n^2) time and O(n^2) extra space)

If we take a closer at the triangle, we observe that every entry is sum of the two values above it. So we can create a 2D array that stores previously generated values. To generate a value in a line, we can use the previously stored values from array.



```
// A O(n^2) time and O(n^2) extra space method for Pasca
void printPascal(int n)
{
  int arr[n][n]; // An auxiliary array to store generate

  // Iterate through every line and print integer(s) in
  for (int line = 0; line < n; line++)
  {
    // Every line has number of integers equal to line nu
    for (int i = 0; i <= line; i++)
    {
      // First and last values in every row are 1
      if (line == i || i == 0)
           arr[line][i] = 1;
      else // Other values are sum of values just above
           arr[line][i] = arr[line-1][i-1] + arr[line-1]
      printf("%d ", arr[line][i]);
    }
    printf("\n");
  }
}
```

This method can be optimized to use O(n) extra space as we need values only from previous row. So we can create an auxiliary array of size n and overwrite values. Following is another method uses only O(1) extra space.

Method 3 (O(n^2) time and O(1) extra space)

This method is based on method 1. We know that ***i***th entry in a line number *line* is Binomial Coefficient *C(line, i)* and all lines start with value 1. The idea is to calculate *C(line, i)* using *C(line, i-1)*. It can be calculated in O(1) time using the following.

```
C(line, i)   = line! / ( (line-i)! * i! )
C(line, i-1) = line! / ( (line - i + 1)! * (i-1)! )
We can derive following expression from above two expressions.
C(line, i) = C(line, i-1) * (line - i + 1) / i

So C(line, i) can be calculated from C(line, i-1) in O(1) time
```

```c
// A O(n^2) time and O(1) extra space function for Pasca
void printPascal(int n)
{
  for (int line = 1; line <= n; line++)
  {
    int C = 1;  // used to represent C(line, i)
    for (int i = 1; i <= line; i++)
    {
      printf("%d ", C);  // The first value in a line is
      C = C * (line - i) / i;
    }
    printf("\n");
  }
}
```