32. Pascal Triangle | Finding nCr in minimal time

118. Pascal's Triangle

Easy 9639 314 Add to List Share

Given an integer `numRows`, return the first numRows of **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:

Example 1:

```
Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
```

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

3 type of Question

1) Given Row and Column, tell the element in that place

R = 5, C = 3, o/p => 6

2) Print any n^{th} row of pascal traingle

N = 5 o/p ⇒ 1 4 6 4 1

3) Given N, print the entire traingle. N = 6

Solution for 1st

$$^{R-1}C_{C-1}$$

$$^{n}C_{R} = \frac{n!}{r! \times (n-r)!}$$

R = 5, C = 3

$$^{4}C_{2} = \frac{4!}{2!\,(2!)} = \frac{\overset{2}{\cancel{4}} \times 3 \times \cancel{2} \times 1}{\cancel{2} \times 1 \times \cancel{2} \times 1}$$

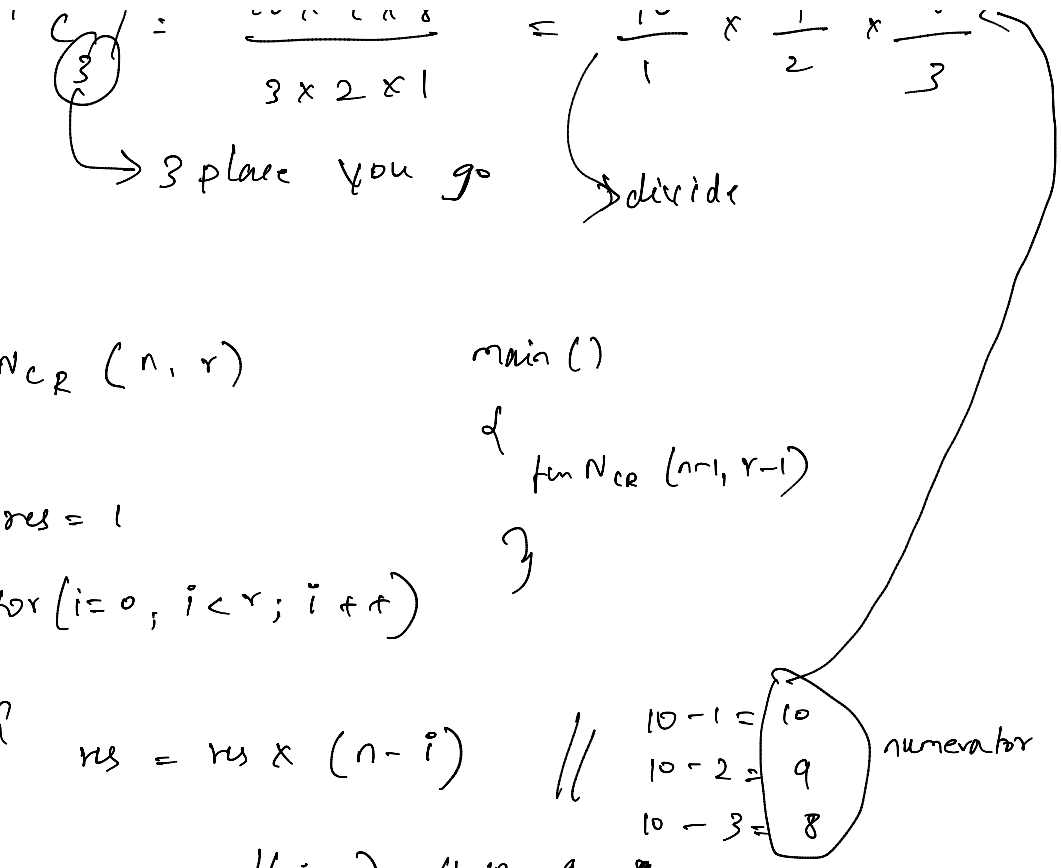
$^{4}C_{2} = 6$ o/p

$$^{7}C_{2} = \frac{7!}{2! \times (5!)} = \frac{7 \times 6 \times \cancel{(5 \times 4 \times 3 \times 2 \times 1)}}{2 \times 1 \times \cancel{5 \times 4 \times 3 \times 2 \times 1}}$$

2 plus you go before

$$= \frac{7 \times 6}{2 \times 1}$$

$$^{10}C_{3} = \frac{10 \times 9 \times 8}{3 \times 2 \times 1} = \frac{10}{1} \times \frac{9}{2} \times \frac{8}{3}$$




```
fun NCR (n, r)
{
long long res = 1
    for (i=0; i<r; i++)
    {
        res = res * (n-i)
        res = res/(i+1)
    }
    return res;
}
```

```
main()
{
    fun NCR (n-1, r-1)
}
```

T.C ⇒ O(r) S.C ⇒ O(1)

Solution for 2nd

1

2

3

1 1

1 1

1 2 1

4 1 3 3 1

5 1 4 6 4 1

6 1 5 10 10 5 1

[n^{th} row → n^{th} element]

Brute Force

$r-1$

C_{c-1}

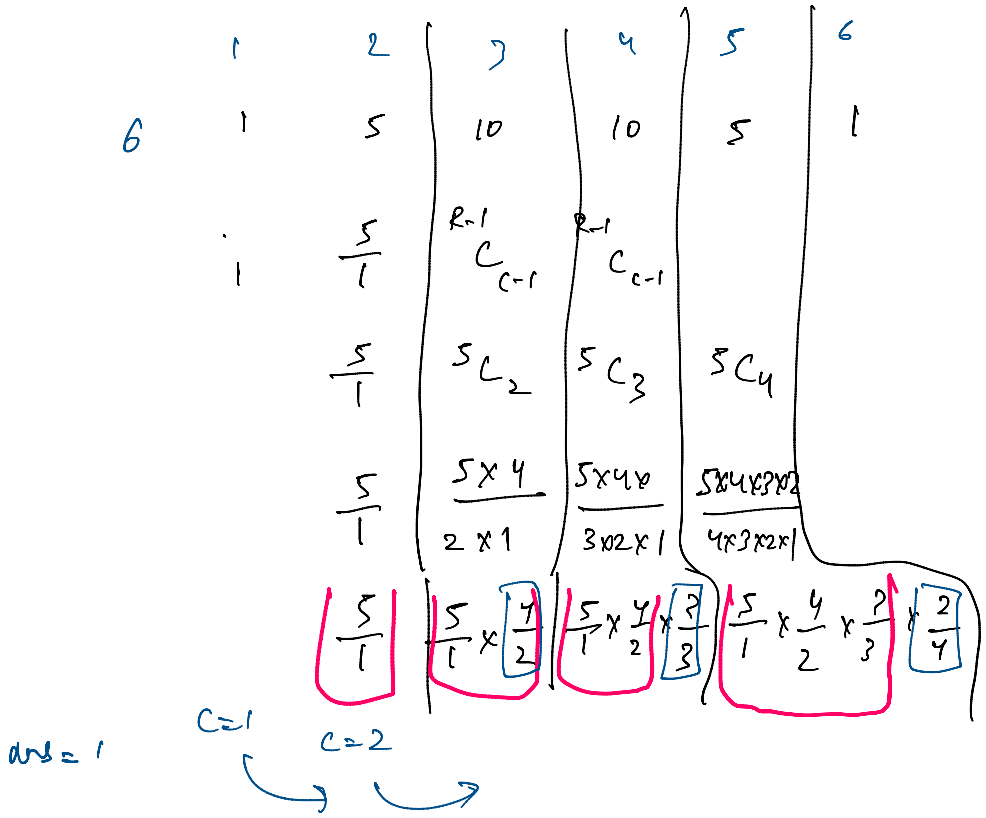
```
for ( c = 1 ; c <= n ; c ++)
{
    print ( fn NCR (n-1, c-1))
}
```

T.C $O(n \times r)$ S.C $\Rightarrow O(1)$

Optimal Solution

$R-1$

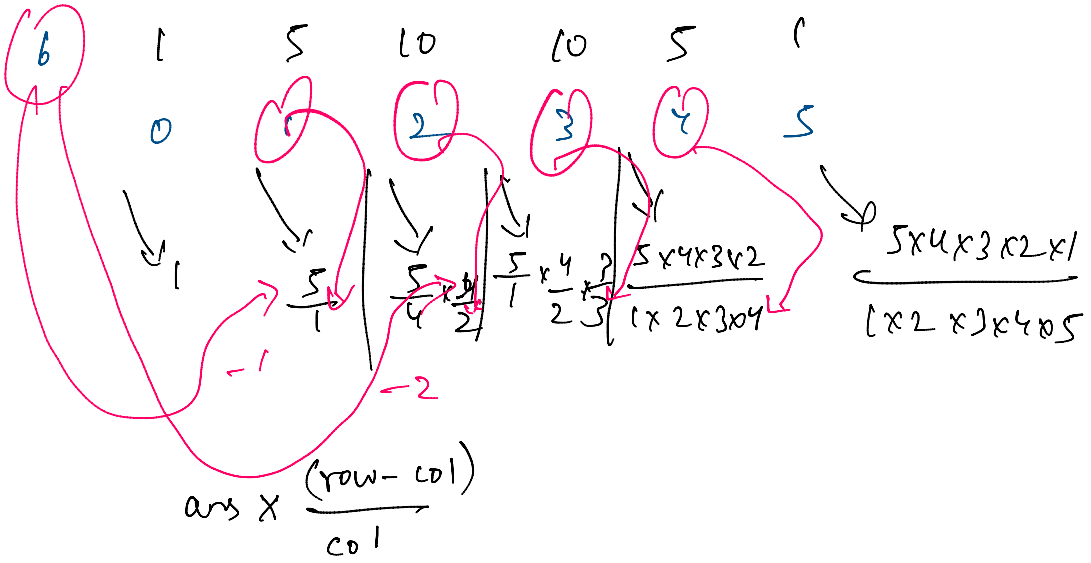
C_{c-1}



first & last element = 1 (always)

ans = 1

now if column is zero based indexing,



whatever is "ans" we are dividing it column

now for multiplying ⇒ (row − col)

Pseudo Code

```
ans = 1
print(ans)  //0th col
for(i=1; i<n; i++)
{
    ans = ans x (n-i)      // row, col
    ans = ans / i
    print(ans);
}
```

TC ⇒ O(n) SC ⇒ O(1)

Solution for 3rd type :

i) Brute Force

$^{r-1}C_{c-1}$

```
ans = []
for (row = 1 ---> n)
{
    templist = []
```

```
for (col = 1 --> row)
{
    templist.add( nCr(row-1, col-1));
}
ans.add(tempList);
}
print(ans)
```

→ function from 1st type

TC = ≈ O(n x n x r) ≈ O(n³)

Optimize Approach

— Using 2^{nd} type problem.

CodeStudio

```
vector<int> generateRow(int row){
    long long ans = 1;
    vector<int> ansRow;
    ansRow.push_back(1);
    for(int col = 1;col<row;col++){
        ans = ans * (row - col); //numerator
        ans = ans / col; // denomentor
        ansRow.push_back(ans);
    }
    return ansRow;
}
vector<vector<int>> pascalTriangle(int N) {
    vector<vector<int>>ans;
    for(int i = 1;i<=N;i++){
        ans.push_back(generateRow(i));
    }
    return ans;
}
```

LeetCode

```
class Solution {
public:
        vector<int> generateRow(int row){
        long long ans = 1;
        vector<int> ansRow;
        ansRow.push_back(1);
        for(int col = 1;col<row;col++){
            ans = ans * (row - col); //numerator
            ans = ans / col; // denomentor
            ansRow.push_back(ans);
        }
        return ansRow;
    }
public:
        vector<vector<int>> generate(int N) {
        vector<vector<int>>ans;
        for(int i = 1;i<=N;i++){
            ans.push_back(generateRow(i));
        }
        return ans;
    }
};
```