You are given two positive integers `n` and `limit`.

n candies → max no of candies we can give to a child.

Return *the* ***total number*** *of ways to distribute* `n` *candies among* `3` *children such that no child gets more than* `limit` *candies.*

3 children

ch 1 ch 2 ch 3

$a + b + c = n$.

no child gets more than 'limit' candies.

Example 1:

→ no of candies

```
Input: n = 5, limit = 2
Output: 3
Explanation: There are 3 ways to distribute 5 candies such that no child gets more than 2 candies: (1, 2, 2), (2, 1, 2) and (2, 2, 1).
```

→ no child gets more than 2 candies

Example 2:

```
Input: n = 3, limit = 3
Output: 10
Explanation: There are 10 ways to distribute 3 candies such that no child gets more than 3 candies: (0, 0, 3), (0, 1, 2), (0, 2, 1), (0, 3, 0), (1, 0, 2), (1, 1, 1), (1, 2, 0), (2, 0, 1), (2, 1, 0) and (3, 0, 0).
```

Constraints:

- $1 <= n <= 10^6$
- $1 <= limit <= 10^6$

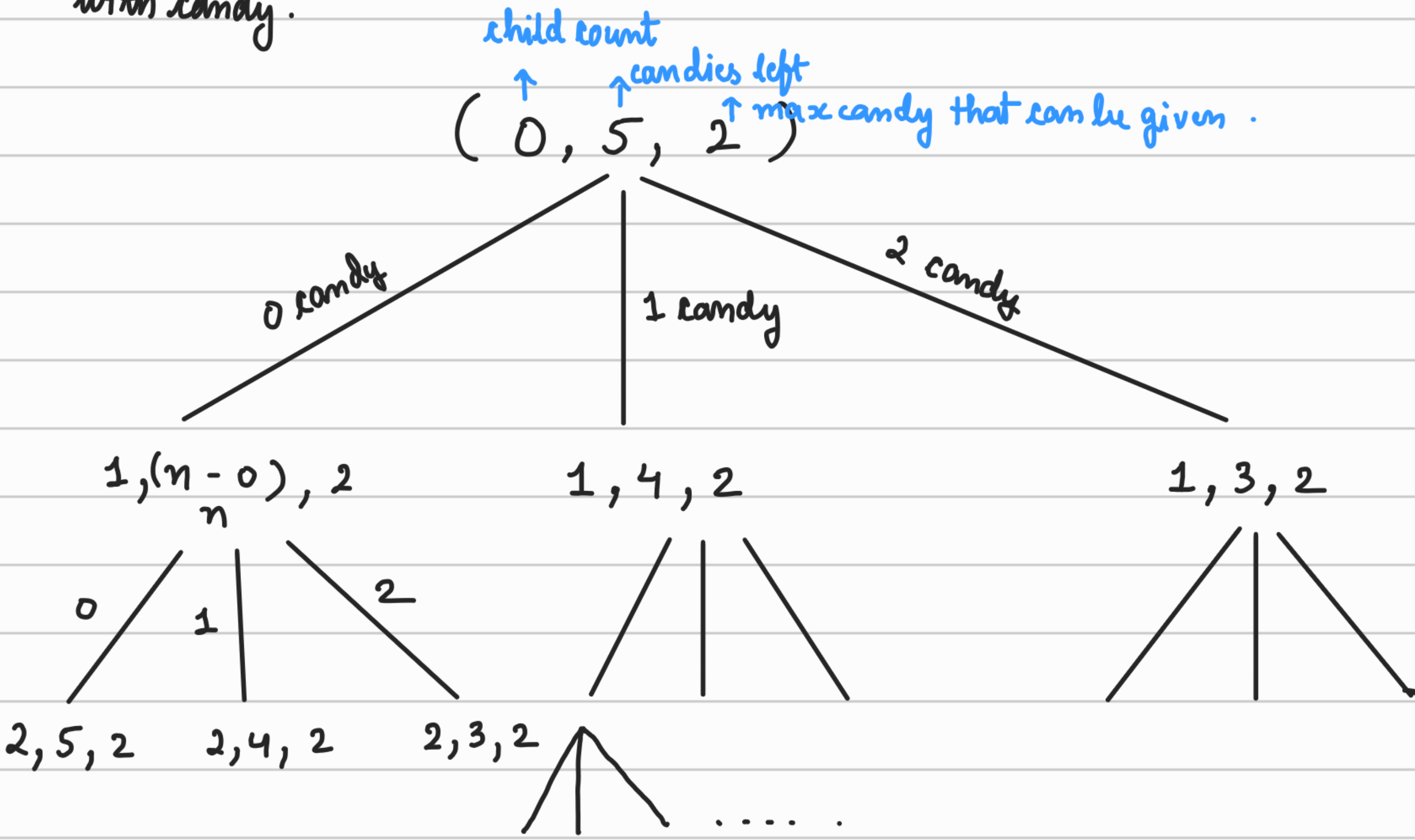
Thought Process

(How I will solve this in an interview)

$n = 5$, limit $= 2$

(0, n, 2)

- 0 → initially, 0 child assigned with candy.
- n → n candies left
- 2 → max candy that can be given to a children.



Solve (0, n, limit)

children count meaning how many children with candies.

int solve (int countchild, int n, int limit) {

total no of children (given)

```
    if ( countchild == 3 )
        if(n==0)
            return1; // got 1 such possible distribution.
```

all candies distributed

return 0; → else.

}

```
int ways = 0;
for (int assign = 0; assign <= limit; assign++){
    ways += Solve( countchild + 1, n - assign, limit )
}
return ways;
}
```

max no of candies is not more than limit.

countchild + 1 → one child has been assigned with candy ∴ +1

n - assign → total candies left

limit → max candy that can be assigned to single child.

Edge Case

n = 5 , limit = 100 → meaning max to max we can give a child 5 candies even though its limit is 100.

```
for (int assign = 0; assign <= limit; assign++){
                             <= 100
ways += Solve( countchild + 1, n - assign, limit )
                                5 - 100
```

5 - 100 → -ve → invalid.

∴ Solve(0, n, limit)

```
int solve(int countchild, int n, int limit) {
        if (countchild == 3) {
                if(n==0)
                        return 1;

                return 0;
        }
int ways = 0;
                                                min(n, limit)
for (int assign = 0; assign <= limit; assign++) {
    ways += Solve(countchild+1, n-assign, limit)
    }
    return ways;
}
```

T.L.E

Time comp → $O(\text{limit}^3)$

	ch1	ch2	ch3	
max candy that can be given	limit	* limit	* limit	$\Rightarrow O(\text{limit}^3)$

Approach - 2

(iterative)

$n = 5$
limit $= 2$

ch 1   ch 2   ch 3

```
for ( ch1 = 0 ; ch1 <= min ( n, limit ) ; ch++) {
                          if child one has consumed ch1 from total n
                          the the max ↑ we can give to ch2 is n-ch1 .
    for( ch2 = 0 ; ch2 <= min(n-ch1, limit ; ch2++) {

        for( ch3 = 0 ; ch3 <= min( n-ch1-ch2 ; ch3++) {

                     if( ch1 + ch2 + ch3 == n ) {
                            ways++ ;
                     }
        }
    }
}

return ways;
```

Time Complexity : $O(n^3)$

Approach - 3

Improving the above code further.

```
for ( ch1 = 0; ch1 <= min(n, limit); ch++) {      ⇒ child 1

    for ( ch2 = 0; ch2 <= min(n-ch1, limit; ch2++) {      ⇒ child 2
        in ch3 = n - ch1 - ch2 ;      ⇒ child 3
        ~~for( ch3 = 0; ch3 <= min( n-ch1-ch2; ch3++) {~~
                if ( ch3 <= limit ) {
                ~~if( ch1 + ch2 + ch3 == n ) {~~
                    ways++ ;
                }
    }
}

return ways ;
```

if we already know how much candy was given to child 1 & 2 then we don't need a for loop for child 3.

Ex : 10 → 2 → candy assigned to child 1
→ 3 → candy assigned to child 2
→ 10 - 3 - 2 → child 3.

$m = \min(n, limit)$;

Time complexity : $O(m)^2$

Approach - 4

[OPTIMAL SOLUTION]

O(N)

Input → n, limit, ch = 3.

Why is this Question difficult ? ⇒ one is constraint is hight
second is,
Let's say instead of children = 3, it's children = 1.

n = 5, limit = 3,

candies that can be given

children = 1   0 | 1 | 2 | 3

min = 0 → min candy can be given
max = limit → max " " " "

4 ways to give candies.
0 candy or 1 or 2 or 3.

$3 - 0 + 1 = 4 \rightarrow$ 4 ways.

$[a, b] \rightarrow$ range
$b - a + 1 \rightarrow$ no of elements
Ex: $[2, 10]$
$10 - 2 + 1 = 9$
2, 3, 4, 5, 6, 7, 8, 9, 10

$[0, 3] \rightarrow$ 0, 1, 2, 3.
4 ways.

(max range − min range + 1)

for, $n = 5$, limit $= 3$
children = 2.

		ch 1	ch 2	
1 way	←	0	$n - 0$	$== n$
2 "	←	1	$n - 1$	$== n$
3 "	←	2	$n - 2$	$== n$
4 "	←	3	$n - 3$	$== n$
5 "	←	4	$n - 4$	$== n$
6 "	←	5	$n - 5$	$== n$

no of possible ways = 6.

ch1 = 0 → min candy that can be given
ch1 = 5 → max " " " "

max − min + 1
5 − 0 + 1 = 6 ways.
of distribution.

limit = 3

ch1	ch2	
0	5 − 0 ×	invalid as max that can be given is 'limit'
1	5 − 1 = 4 ×	invalid.
2	5 − 2 = 3 ✓	valid.
3	5 − 3 = 2 ✓	valid.
4	5 − 4 = 1 ×	invalid limit = 3 candy given to 1 is 4.

∴ acc to limit.

ch1 min = 2.

ch1 max = 3

3 − 2 + 1 = 2 ways
of distribution

Another Ex :

children = 2

$n = 5$, $\underline{limit = 4}$.

ch 1	ch 2	
0	$5 - 0 = 5$	× invalid. limit = 4 ch2 = 5
1	$5 - 1 = 4$	✓
2	$5 - 2 = 3$	✓
3	$5 - 3 = 2$	✓
4	$5 - 4 = 1$	✓
5	$5 - 5 = 0$	× invalid limit = 4 ch1 = 5

4 possible ways.

min = 1
max = 4

$[1, 4] \rightarrow 4 - 1 + 1 = 4$ possible ways.

what is the min candy that can be given ?

Ex : max value (candy) that can be assigned to children 2 is $n - 1$ Ex : $5 - 1 = 4$.

if max value of ch2 is 4

no of candies

then min value of ch1 is $n - maxch2$.

Ex : $5 - 4 = 1$.

ch 1's minimum candies ?

→ child 2 given limit no of candies.

(n - limit)

if totalcandy is 5 then max assigned will be 5 only, even though limit = 100

$\boxed{ch1\ min = n - limit}$.

total no of candies

max can be assigned

edge case : $N = 5$ $limit = 100$

$\boxed{ch1\ max = limit}$

$\therefore$ ch1 max = min (N, limit) ;

One case to consider, n = 10, limit = 4

ch2 → max → 4

ch1 min → 10 − 4 = 6 → not possible limit = 4

ch 1 min = 6 .

In case where limit * 2 < n → we will never be able to allocate anything to any of the child.

$\therefore$ limit * 2 < n → never possible.

children = 3 ⟶ This is what makes the q difficult.

<u>**children = 3**</u>

n, limit

for O(n^3) soln → 3 loops → i, j, k
↓
O(n^2) soln → we found k (ch3) with help of i & j (ch1 & 2)

let's see if we can impliment something like this here or not, our new n

$N = (n - x)$ candies left, limit .

ch 1

x candies assigned

ch 2 ch 3

this becomes two children case which we solved before .

Children = 3

n , limit

$N = (n - x)$, limit

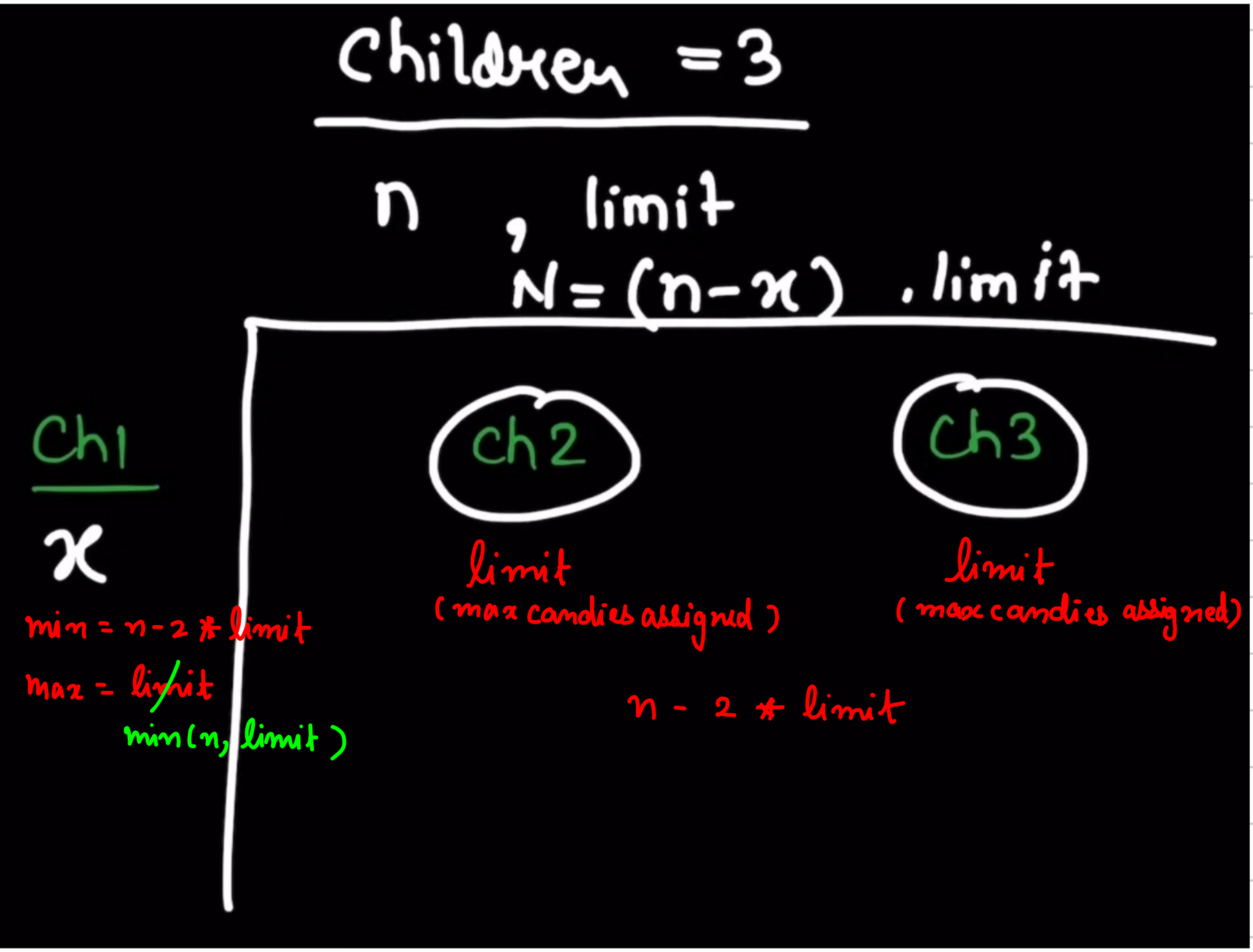
$n - 1$ ... & so on

Ch1

$x = 0, 1, \ldots$

Ch2 Ch3

we need to find valid min candy & max candy of ch 1.



max candies assigned to ch 2 & ch 3.

candies left = n - 2 * limit.

ch 1 min candies value = n - 2 * limit;

all three children's sum will be equal to n by this way (which we want)

$$\underset{CH1}{(n - 2\,limit)} + \underset{CH2}{limit} + \underset{CH3}{limit} = n$$

$$n = n$$

$\therefore$ rule is followed.

max value of ch1 = limit.

edge case

$n = 2$, $limit = 100$

↳ min (n, limit)

$\therefore$ max value of ch1 = ~~limit~~ min(n, limit).

```
for ( i = min ; i <= max ; i++ ) { // child 1 .

    // (n - i) → candies left after assigning to ch1

    // for ch2 & ch3
    // this becomes the case of children = 2
       (we solved before)                 limit
    ∴ if we find ch2's min & max
         we can find no of ways .

         (max ch2 - min ch2 + 1 )
       [ refer before explanation ]

    int N = (n - i) ;   // remaining candies for ch2 & ch3

    int min ch2 = ~~N - limit;~~  max( 0, N - limit )
    int max ch2 = min ( N, limit );

    // also keep in mind N - limit should not be -ve .
       ∴ if -ve assign 0 .  ⇒ max ( 0, N - limit )

       ways += ( max ch2 - min ch2 + 1 ) ;
         ↳ for every possible x / i
                                  ↳ whatever var is
                                    assigned .
}

return ways ;
```

Time Complexity : O (linear)

O (n) → only 1 for loop .

Space Complexity : O (1) .

O(cubic) → O(cubic) → O(square)

(iterative) (recursive) App 3

App 1 App 2

↓

O(linear)

App 4 .

```cpp
class Solution {
public:
    long long distributeCandies(int n, int limit) {
        long long ways = 0;

        //child1
        int minCh1 = max(0, n - 2*limit);
        int maxCh1 = min(n, limit);

        for(int i = minCh1; i <= maxCh1; i++) { //fixing child1 assignment of candies

            int N = n-i; //for ch2 and ch3

            int minCh2 = max(0, N - limit);
            int maxCh2 = min(N, limit);

            ways += maxCh2 - minCh2 + 1;
        }

        return ways;
    }
};
```