(2) Pascal's triangle

#V1 - When youre given a row & column just find that particular element
- use $^{n}C_{r}$ formula $\frac{n!}{r! \times (n-r)!}$
- but here $n!$ runs on r times
- so just iterate n till r & use the formula

$$value = value \times \frac{n+1-i}{i+1}$$

which will iteratively give the value but then prints the required value

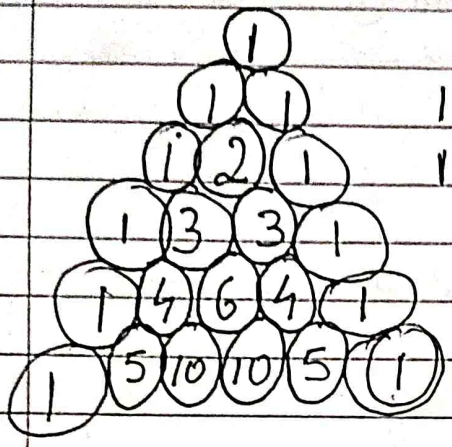
#V2 - just like above using the same formula just iterate throughout the entire array which generates the whole row.

#V3 - using the same formula run row times loop (outer) & for every row the value should start from 1
- the inner loop runs row times since there are row number of elements in each row.
- this generates the triangle till given row no

__/__/__

SDE Sheet

(2) Pascal triangle



1+1

1+2 & 2+1

Variant 1 - given row no & col no
- do Row-1 & col-1 $\quad {}^{r-1}C_{c-1}$

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

ex: $r=5 \quad c=3$

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

$$= 6$$

n! is run r! times so

value = 1, i = 0 → r

value = value $\times \frac{n-i}{i+1}$

in each iteration we multiply the current value with (n-i) & divide it by (i+1) where i = 0 → r

Variant 2 - given row no print the whole row

- just using the given formula print each element
- iterate through the columns - 0 ⇒ n for each place use the formula.

Formulas working

Ex: row = 6 1 5 10 10 5 1

$\frac{5}{5} = 1$

$\frac{5!}{1! \times 4!} = \frac{5}{1} = 5$.

$\frac{5!}{2! \times 3!} = \frac{5 \times 4 \times 3 \times 2 \times 1}{2 \times 1 \times 3 \times 2 \times 1} = \frac{10}{1} = 10$

$\frac{5!}{3! \times 2!} = \frac{5 \times 4 \times 3}{3 \times 2 \times 1} = 10$

there's a pattern in numerator it's decrementing by 1 & denominator there always one increasing so based on this we get the formula

value = 1

$$\text{value} = \text{value} \times \frac{n - i}{i + 1}$$

(2) majority element > n/3

#BF Just using two loops find out how many times an ele is repeated & if its > n/3 then store the 2 in vector & return it
(this problem at max can have 2 valid cdts cuz ex: N = 9 (0-8) N/3 = 3
X ele - can be 4 or more times
Y ele - can be 4
Z ele - cant be 4 since it would exceed array

#Bet - using hashmap just store whichever number is > n/3.

#OPt - Select possible candidates
- verify the selected candidates
- then print

__/__/____

③ 3 sum problem

BF - Basically you iterate through the array using 3 loops (i - 0 → n)(j - i+1 → n)(k - j+1 → n)

- now add up all three elements
- if sum = 0 then add the triplet to a vector(temp)
- now sort the triplet(temp) (you sort cuz later if you get the same triplet in a different combo that shouldnt be added)
- now just insert the triplet into a set (since were sorting triplets any triplets that sums to 0 with a different combination will be skipped basically set recognises the patterns) ex: 1st triplet(-1 0 1) & unsorted triplet(1 0 -1) → after sorted(-1 0 1) → set will recognize this that it exists already.

__/__/____

#OP - so basically we have i pointer fixed at i at start
- when (i != 0) && (a[i] == a[i-1]) skip further execution
- because with i as starting element we get the same triplets that sums to 0.
- now having i as our starting element of the triplet we have (j = i+1) & (k = (n-1)
- while (j<k) we add all pointer eles i + j + k
- now if sum < 0 we increase sum value by j++
- if sum > 0 we decrease sum value by k--
- if sum = 0 you store the triplet in a tempvector & store the tempvector(triplet) into 2D answerVector & then move j & k to find next triplet.
- after moving j & k if the new j & k is same element as its prev ones then move j or k again.
- Then when j > k then terminate & go i loop & repeat.

② merge overlapping intervals

#BF - Iterate through each interval comparing it with all following intervals
- if the current interval belongs to a previous interval then skip & move to next interval
- or else check the current interval with the next interval if the next interval comes under the current interval if yes then expand the current interval
- store the expanded interval after comparing current interval with all possible ~~interval~~ following intervals.

__/__/____

#OP- basically just iterate through the intervals
- if the current interval is new (i.e if the ans array is empty or if current interval is bigger than prev interval) then just push the new interval to ans array
- but if current interval belongs to prev array interval then just expand the prev interval.

— Optimized

- ans vector
- Sort
- iterate through intervals {(1,3) (2,6) (8,10) (15,18)} (0) (1) (2) (3)
- if (ans is empty or current interval is bigger than last interval in ans) [compare current ints first ele to prev ints last ele]
  - then push that interval directly to ans
- else (i.e if current interval belongs to last interval)
  - then update last interval in ans by comparing last intervals last ele with current interval second ele
- do this for all intervals n return ans.

	0	1
0	1	3
1	2	6
2	8	10
3	15	18

③ Merge two sorted arrays without extra space

#BF Basically just take two arrays keep two pointers pointing at first places of both array now iterate through both & compare the two elements whichever is smaller you take that & start filling up from the start of the array a temp array.

#Better:-
- keep two pointers
- 1 pointing at end of 1st array, 2nd pointing at start of 2nd array
- compare the 2 & whichever ele is smaller bring that to first array (basically swap)
- do this for all elements such that the first array will have all smaller elements than 2nd array
- once there are no more possible swaps then break (that means arr1[i] < arr2[0])
- break & just sort both the arrays.

__/__/__

optimal - basically when you're given 2 arrays 1st thing is resize the 1st array to hold both array elements
- now keep 2 pointers pointing at the end of both arrays
- whichever is largest start filling the expanded first array from last
(here first array + 0000 for 2nd array elements)
so 2nd array might have remaining elements which might need to be handled but first or any eles are already there.

⑤ Find the missing number & Duplicate

BF - just using a nested loop
- ~~Missing of N the sum of array~~
- if any of the element is repeated store that to Dupli
- now Sum of N - Sum of array + Dupli gives missing number

Better - Store all ele into a mp
- retrevie dupli
- & use same formula SofN - SofA + Dupli

# OPT - ~~Sum~~ SofA - SofN (val1) // a - b
- SofA² - SofN² (val2) // a² - b² → (a-b)(a+b)
- .
- val3 = val2/val1 // a+b = (a²-b²)/(a-b)
- Dupli = (val3 + val1)/2 // (a+b + a-b)/2a = ans → a = ans/2 = a
- Missing = Val3 - Dupli // a+b - Dupli → b = val3 - a
  (a known)

  or

  missing = SofN - SofA + Dupli.

__/__/__

(6) count inversions

basically count elements or pairs whose left elements are larger than its right.

ex: {5, 3, 2, 4, 1} - (5,3), (5,2) (5,4) (5,1)
- (3,2) (3,1)
- (2,1) (4,1) total 8

BF just use two loops for every element check the elements ahead of it if its lesser, then count it as a pair
then finally return the count.

OP - using merge sort we recursively break down the given array as left half & right half till the last element
- now while merging the arrays if an element from right half is less than element from left then that means all elements on right left can form inversion with the smaller right ele
- so we increase inversion count by taking all elements on left (ie mid - left + 1)
- then finally return count.