5. you are given an array containing all distinct elements. Find the number of possible subsequences

Ex:

N = 3 , Arr = [1,2,3]

Ans. 7

Subsequences are : [1] , [2] , [3], [1,2] , [1,3] , [1,2,3]
[2,3]

Solution Approach:

what's a subsequence?

[_ _ _ _] ⇒ you can decide take it (an ele.) or not take it.
T/NT T/NT T/NT T/NT

★ 2 choices for each element, possible subsequences :

$2^N - 1$

→ no. of elements

↳ when we don't take any element (that's an invalid substring)

⇒ $2^3 - 1 = 7$

Note: [2,1] ✗ not a subsequence as the order of elements should remain same

★★ Instead of using pow(2,n) use a for loop to store the no. of subsequence as pow returns a floating pt. value.

6. You are given an array. Find the number of its subarrays

Ex:

N = 3, Arr = [1,2,3] Ans = 6

Subarrays are : [1],[1,2],[1,2,3],[2],[2,3] , [3]

Note: A subarray is continuous. Eg: [1,3] ✗ not a subarray.

It's a counting problem

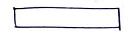
★ 2 Ways to see an array:

①.

No. of 1 sized array = N

No. of 2 sized array = N-1

No. of 3 sized array = N-2



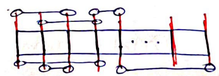
No. of N sized array = 1

total arrays = N + (N-1) + (N-2) + . . . + 1

$= \frac{N(N+1)}{2}$ {sum of 'n' natural nos.}

= possible subarrays.

②



for array of n elements there are n+1 lines that divides them.

Choose any 2 you'll getta subarray between them.

So, in how many ways we can choose 2 lines from n+1 lines

$\boxed{{}^{n+1}C_2 = \frac{(n+1)(n)}{2}}$ Combinatorics

7. You are given an array. Find sum of all subarrays of the array. [Time limit : 1 sec] Constraints:

i. N <= 100   ii. N <= 1000   iii. N <= N^5

Example:

N = 3, Arr = [1,2,3]   Ans = 20

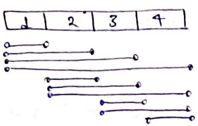
Subarray	[1]	[1,2]	[1,2,3]	[2]	[2,3]	[3]	
sum	1	3	6	2	5	3	=> Total: 20

Approach 1: (using 3 for loops)

Logic? → find start and end of all the possible subarrays & the inner loop iterates over each subarray & sum of each subarray is added to total

```
int total = 0;
for(int st=0; st<n; st++){
  for(int en=st; en<n; en++){
    sum = 0;
    for(int i = st; i<=end; i++){
      sum += arr[i];
    }
    total += sum;
  }
}
```

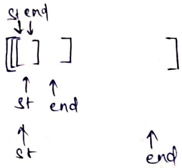
eg:



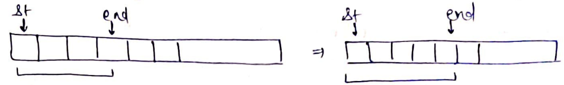
start & end of all the subarrays

+1
+3
+6
+10
+2
+5
+9
+3
+7
+4

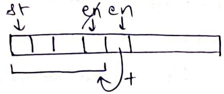
50



Appraoch 2: [using 2 for loops]



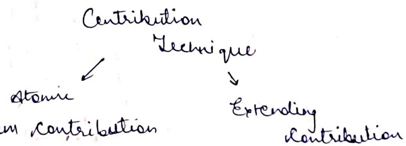
Logic: what we are currently doing is every time the end is shifting we are looping through the entire previous subarray. Now, we will store this sum of previous subarray and add the end element to it and use it in total




```
int total = 0
for(int st=0; st<n; st++){
  int sum=0;
  for(int en=st; en<n; en++){
    int sum += arr[i];
    total += sum;
  }
}
```

Using the sum of previous subarray, adding just the new end value to it.

Approach 3: (Using contribution technque) → Atomic item contribution



1	2	3	
1			1
	2		2
		3	3
1	2		3
	2	3	5
1	2	3	6
			20

$\frac{n(n+1)}{2}$

} these are basically row wise sum of the 2D matrix.

↓

If we add row wise we will get n^2 cuz we saw no. of sub arrays / rows are $\frac{n(n+1)}{2}$

However, if we try to add it column wise, there are only N columns as 'n' elements in the array.

↓

We can observe ~~the no of times~~ an element has appeared in ~~a subarray~~ how many subarrays.

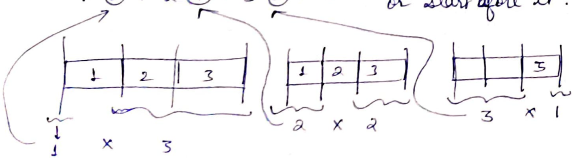
↓

~~If we could just find how many times each element of array has appeared in the subarrays & sum them we will get our result.~~

If we could just ~~find out~~ formulate a mathematics that ~~in~~ how many subarrays ~~each~~ contains an element, we get the no. of times that element has appeared & multiply it with the element

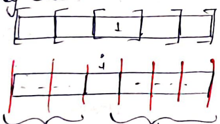
1	2	3
1		
	2	
		3
1	2	
	2	3
1	2	3

$1*③ + 2*④ + 3*③ = 20$



Calculate how many sub arrays contains an element:

Case by Case



→ A sub array can only contain ith element if it ends after it or start before it.

↓ i+1 lines

→ arrays possible

↓

$^{i+1}C_1 = i+1$

→ total lines in arr of 'n' elements − i+1

$= (n+1) - (i+1)$

$= (n-i)$

→ arrays possible

$^{n-i}C_1 = n-i$

⇒ No. of subarrays in which i comes =

$(i+1) * (n-i)$

Final code:

```
int total = 0;
for (int i = 0; i < n; i++) {
        total += arr[i] * (i+1) * (n - i);
}
cout << total;
```