*“So if an algorithm is an idealized recipe, a program is the detailed set of instructions for a cooking robot preparing a month of meals for an army while under enemy attack,”*

*― Kernighan Brian W.*

Dear reader, welcome to a new problem based on *Arrays &* *Bit Manipulation*. The problem’s name is *‘Subsets of Array*’.

***Problem Statement***

You are given an array of size n, you have to print all subsets/subsequences of the array in separate lines. This problem is also known as: printing ***power set*** of an array. You can consider that the array elements will be distinct.

A subsequence/ subset of an array can be formed by choosing some (may be 0, 1, 2, ... or equal to size of array) elements out of all the possible array elements, in the same order in which they appear in the original array.

Important Links: [Problem Link](https://www.pepcoding.com/resources/online-java-foundation/function-and-arrays/subsets-of-array-official/ojquestion), [Question Video](https://www.youtube.com/watch?v=vk8sfizNtsY), [Solution Video](https://www.youtube.com/watch?v=iKSI_9NHR1M)

***Example***

For Array = {10, 20, 30}, there will be 2n (= 23) Subsets/Subsequences:

- - -

- - 30

- 20 -

- 20 30

10 - -

10 - 30

10 20 -

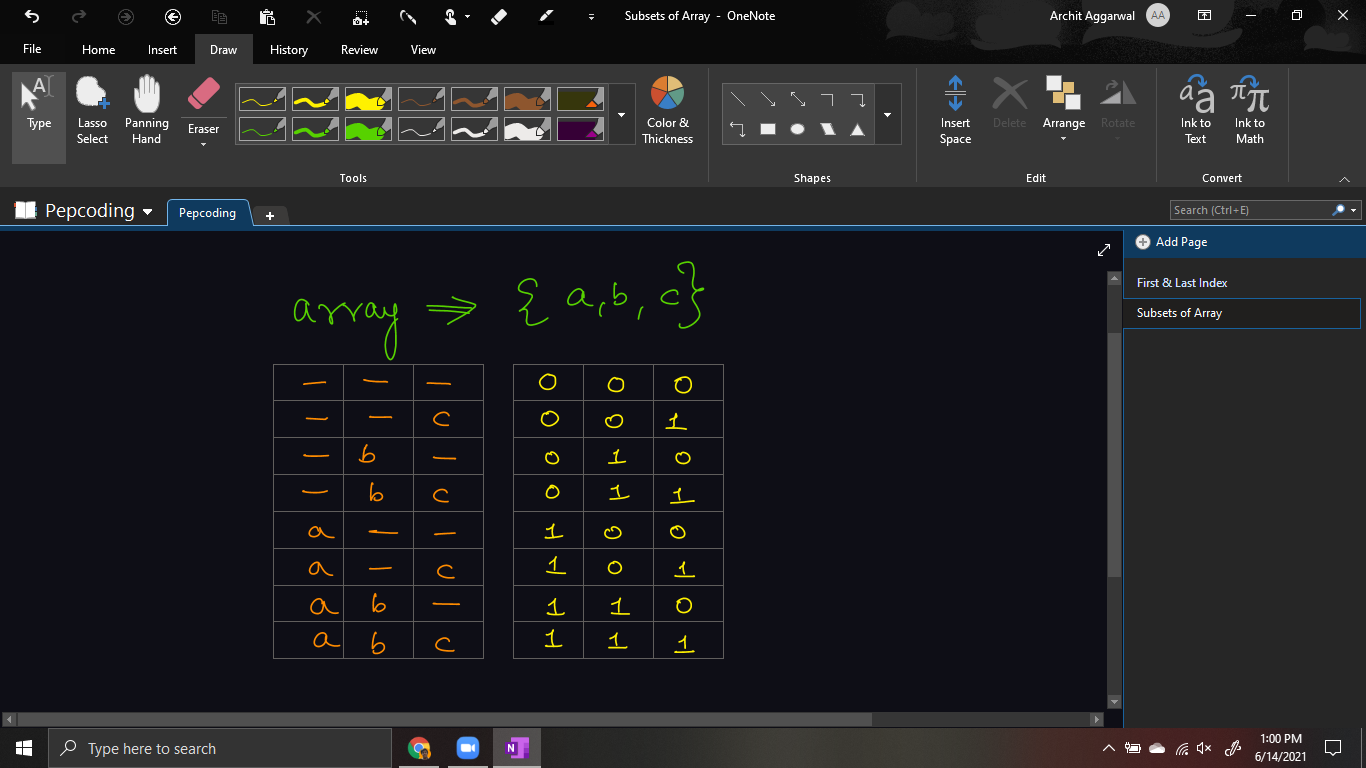
10 20 30

***Deducing Algorithm***

Let us consider the array = {a, b, c}. I want you to write down all the subsequences /subsets together, and find out a pattern on how we can generate them.

*HINT*: Try to assign number 0 to element if it is not present in the subset, and 1 if it is present. Now, try to visualize each subset using these combinations of 0s and 1s.

Let us look where we are heading towards:



If you remember ***number theory*** well, then, on reading the table on the right side row-by-row, you will find out that it represents the binary equivalent of numbers from 0 to 2n - 1, where n = number of elements in the array.

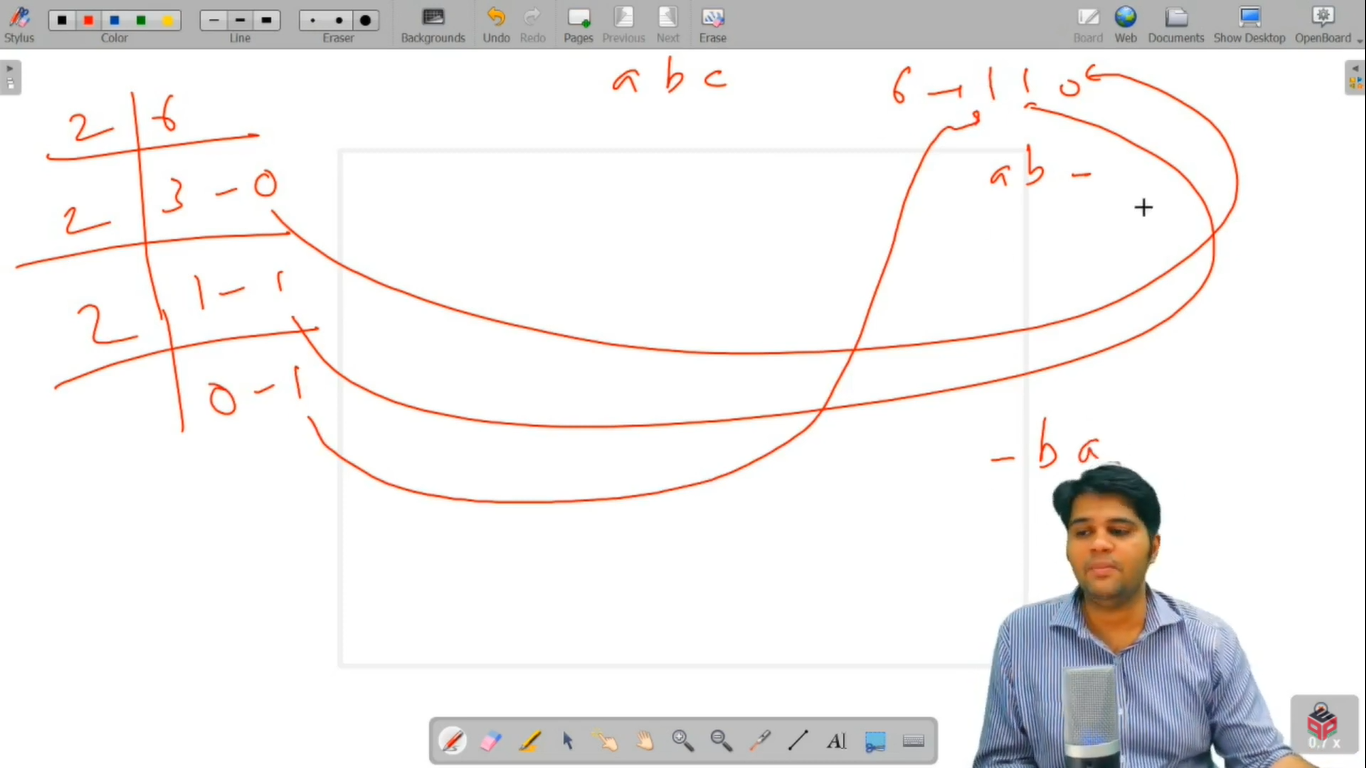
Hence, we can try to generate all numbers from 0, less than limit ( = 2n), and for each number, check all of it’s bits one by one, if kth bit is set (kth bit of the binary number is 1), then we will take kth element from the array to our subset, otherwise not.

But, how will we generate check if kth bit is set or not? For a given number, we will extract bits from the right-most place one by one. If the right-most bit of a number is set, then the number is odd, else the number is even.

So, we figured out how to check the last bit, but how will we move to the next bit (second-last bit and so on)? We can simply divide the number by 2. It will remove the last bit (irrespective of its value, whether it was 0 or 1).

Thus, extracting bits from a decimal is the same as repeated division of the decimal number by 2 and finding out the remainders. (It is the same logic as converting a number from base 10 to base 2).

To revise the concept, have a look at [this](https://www.youtube.com/watch?v=iKSI_9NHR1M)  [3:57, 4:50] section of the video.



Now, there is just a small problem left. As you are extracting bits from the right side, you will get the subset in reverse order. For eg for number 6 (110), you want the result as {a, b, -} but you will get {-, b, a}.

Thus, either reverse the subset in the last ***or*** while you are building the subset add elements (or empty -) in front of the string.

***Pseudo Code/ Algorithm***

* Calculate limit for outer loop, limit = 2n
* Run Outer Loop from 0 to limit
  + Initialize temp as i, and string str to store the current subset.
  + Run Inner Loop from n-1 to 0.
    - Check if the last bit is set or not (by checking if temp % 2 == 1 or not).
      * If bit is set, then add jth element from array to subset
      * Else add ‘-’ (empty slot)
    - Divide temp by 2 to remove the rightmost bit.
  + Print the current subset in string *str*.

Please try to code this without taking help of the video solution. It will help you develop a great insight on arrays. Once you have tried, do give a read to the solution code provided below and explained in the video.

***Implementation (Java)***

import java.io.\*;

import java.util.\*;

*public* class Main

{

*public* *static* void main (String[]args) throws Exception

{

Scanner scn = *new* Scanner (System.in);

int n = scn.nextInt ();

int[] arr = *new* int[n];

*for* (int i = 0; i < arr.length; i++)

{

arr[i] = scn.nextInt ();

} int limit = (int) Math.pow (2, arr.length); *//calculating the number of subsets*

*for* (int i = 0; i < limit; i++)

{

*//convert i to binary and use 0's and 1's*

*//to check if an array's element is to be printed or not*

String set = "";

*// we make use of set to print in required order*

int temp = i;

*// we store i because we need to use value*

*// of i without manipulating the actual i*

*// as i is the outer loop iterator*

*for* (int j = arr.length - 1; j >= 0; --j)

{

*// calculating the binary, extracting*

*// the remainder one and by one*

*// and putting required element*

*// in the String to be printed.*

int rem = temp % 2;

temp = temp / 2;

*if* (rem == 0) *//nothing to be printed*

{

set = "- " + set;

}

*else*

{

*// we print the element, so we add it to our answer string*

set = arr[j] + " " + set;

}

}

System.out.println (set);

*// printing the required pattern line-by-line*

}

}

}

This code is written and explained by our team in [this video](https://www.youtube.com/watch?v=iKSI_9NHR1M) from *[2:05, 10:00]*. Please refer to it if you are stuck somewhere.

***Time & Space Complexity Analysis***

I admit, this is a hard one to analyze. Consider two nested loops, and the logic inside the nested loop. Still, give it a try.

**Time Complexity** -

* Outer loop is running from 0 to limit = 2n : O(2n)
* The inner loop runs for n times: O(n)
* Inside the inner loop, we are doing a constant work (reducing temp by 2 and updating resultant string)

Hence the time complexity will be O(2n \* n).

**Space Complexity** - If we will only print (not store) all the subsets, then only one string is enough, which can be of maximum size as the size of array, hence auxiliary space required will be = O(n).

*Note*: If we will have to store all the subsets, then, since there can be O(2n) subsets, hence 2^n strings of n size each, auxiliary space required will be O(n \* 2n).

**Extra Gyaan(Knowledge)**:

* Above problem can also be solved by applying the concepts of *Recursion & Backtracking*. We will encounter this problem again in the Recursion Section.
* You know how to manipulate your parents to get what you want, right? But since you do not know how to manipulate bits, you are missing out on how to handle the logic inside the inner loop using just one line).

But, don’t worry! We will learn these approaches soon.

***Note***: If you are answering the space complexity of a solution, then there can be few aspects to the answer: whether we have to consider the space taken by the input or not, whether we have to consider the space taken by the output or not. Hence, you should know how much space your solution is taking with and without considering the space of input and output data also.

***Note***: This algorithm will result in printing duplicate subsets if there are duplicates present in the array. To avoid that, we can create a unique set of all subsets using *hashing,* (again don’t worry, it is yet to come), and insert all subsets in this unique set.

**Asked in Companies**: *Amazon, Microsoft*

I hope you enjoyed solving the problem with me. We will see you in the next section: *2D Arrays*. Good Bye!

Contributor : Archit Aggarwal