Dear reader, welcome to the next problem in the Recursion & Backtracking section named ‘[***Permutations - 2***](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/permutations-ii-official/ojquestion)’.

Please, I request you to solve the previous problems ‘[Permutations - 1](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/permutation-i-official/ojquestion)’ and ‘[Combinations - 1](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/combinations-i-official/ojquestion)’ before jumping on to this problem. Infact, I recommend you to watch these sets of problems in given order and try to complete the entire set together.

***Problem Statement:***

* You are given a number of boxes (nboxes) and a number of non-identical items (ritems).
* You are required to place the items in those boxes and print all such configurations possible.
* Items are numbered from 1 to ritems.
* Note 1: Number of boxes is greater than number of items, hence some of the boxes may remain empty.
* Note 2: Check out the question video and write the recursive code as it is intended without changing signature. The judge can't force you but intends you to teach a concept.

***Example:***

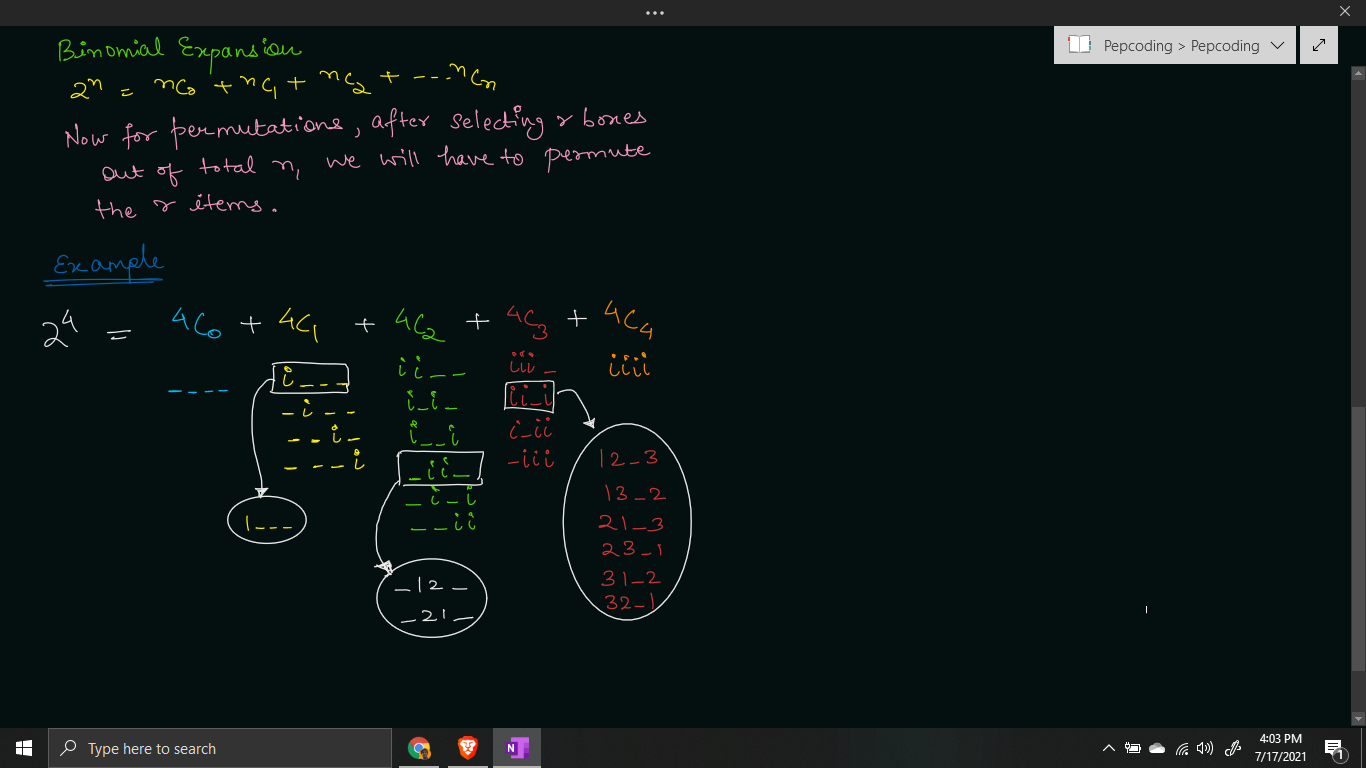
*Input*: Number of boxes (nboxes) = 4, number of non-identical items (ritems) = 3

*Output*: [1230, 1203, 1320, 1023, 1302, 1032, 2130, 2103, 3120, 0123, 3102, 0132, 2310, 2013, 3210, 0213, 3012, 0312, 2301, 2031, 3201, 0231, 3021, 0321]

***Solution***

In the previous solution for the same permutations problem, we used the formula of permutation coefficient and printed all permutations by taking **LEVELS as ITEMS and CHOICES/EDGES as BOXES** in the recursion tree.

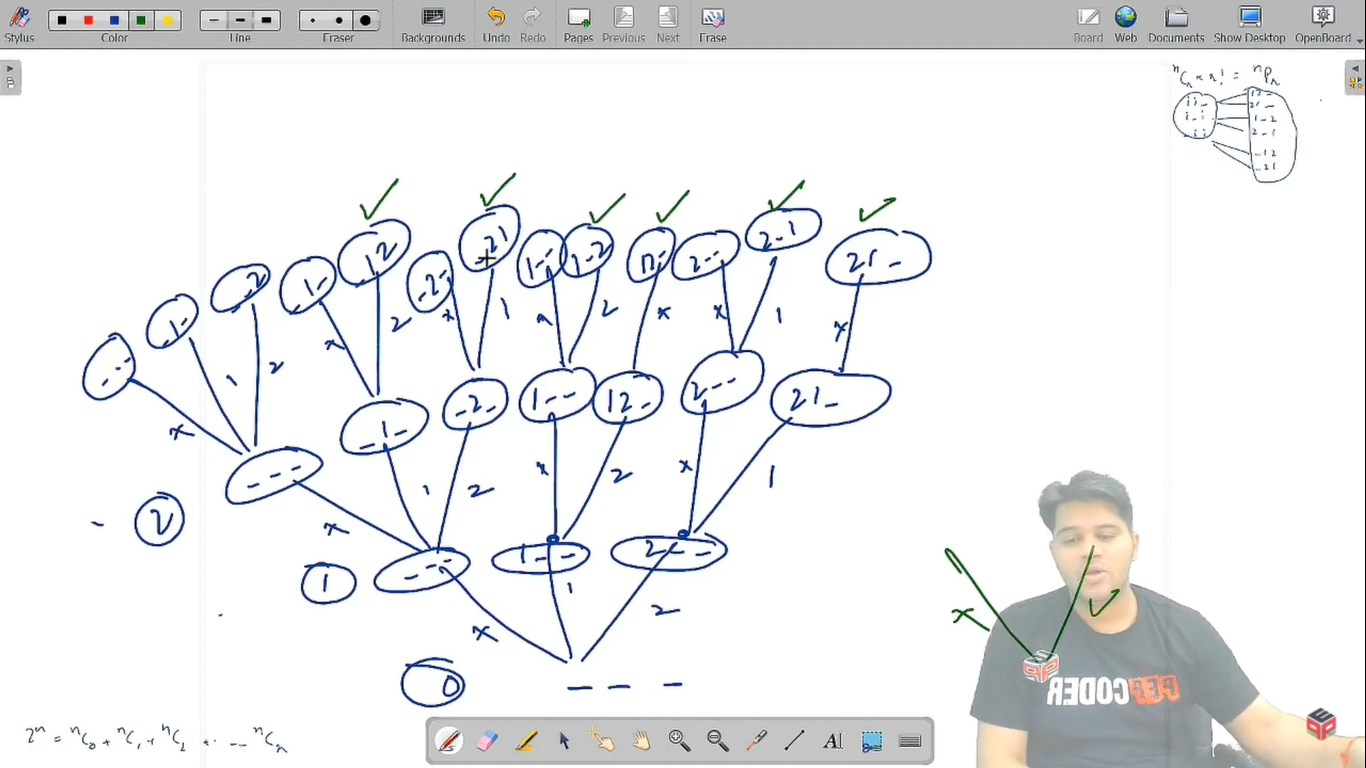
In this solution, we will find a relation between binomial expansion and the permutation coefficient. Hence, we will try to find permutations by taking **LEVELS as BOXES** and find out what should be the choices.



**Q)** Can you see a relation, and find out how all the permutations of r non-identical items (1, 2, 3, .. r) can be generated using the binomial expansion?

* In combinations, for each box, we had only two choices, whether to place an identical item (i) or not.
* But for permutations, since the items are non-identical, instead of having 2 choices, we will have **r + 1 choices**.
* These choices will be: whether to place 1st item in the current box, or 2nd item in the current box, … rth item in the current box or do not place any item.

Hence in a recursion tree, **LEVELS are equivalent to BOXES**, but the **CHOICES/EDGES are ITEMS** = 1, 2, 3, … r or no item (0).



Let us quickly define the expectation, faith and finally meet our expectations with the faith.

**Expectation**: We expect that the recursive function will give us all the permutations of r non-identical items. At each level, we are deciding for the current box cb.

permutations(int cb, int tb, int[] items, int ssf, int ts, String asf)

Here, cb is the current box whose decision is to be taken, tb is the total number of boxes, ssf is the number of items placed so far, ts is the total number of identical items, and asf is the answer so far.

Also, there is an items array which will store the box number, in which the item is stored. If items[i] = j, it means ith item is placed in jth box. If the item is not placed yet, then items[i] should be 0.

**Faith**: We will keep faith on the recursive function that will help us make all the choices for the remaining n-1 boxes.

permutations( nextBox, tb, items, ssf, ts, String asf)

**Meeting Expectation with Faith:**

For the current box, we have r + 1 options, whether we should place an item in it or let it remain empty.

But, an item can be placed in the current box only if it has not been placed in any of the previous boxes.

Hence, we will run a loop over all the items [0, r-1] and check whether the current item is placed or not. If it is not placed yet (item[i] == 0), then we will place it in the current box, by updating item[i] = cb, in the ***node-pre area***.

After placing an item, we will recursively call for the remaining boxes by increasing cb by 1, adding the item to the answer so far, and incrementing items placed so far by 1.

Also, after returning from the recursive function of remaining boxes, i.e in the ***edge-post area***, we will update the current box back to empty (item[i] = 0) .

for(int i = 0; i < items.length; i++){

if(items[i] == 0){

items[i] = cb;

permutations(cb + 1, tb, items, ssf + 1, ts, asf + (i + 1));

items[i] = 0;

}

}

We have explored r choices out of the available r + 1 choices. The only choice remaining to be executed is to not place any item in the current box. Hence, we will simply call for the next box (cb + 1) and add “-” (empty box) to the answer so far .

permutations(cb + 1, tb, items, ssf, ts, asf + 0);

**Base Case**

Since we are picking all the boxes level by level, we should stop when decisions for all the boxes have been made. Hence, we should stop when the current box becomes greater than the total boxes (cb > tb).

Now, there is a catch. As discussed in the mathematical formula, we will generate all the permutations of 0 items, 1 items, 2 items, and so on. But we only require those permutations which have ts (total number of non- identical) items in the answer string. Hence, we will print the permutation only if ssf = ts.

if(cb > tb){

if(ssf == ts){

System.out.println(asf);

}

return;

}

**Java Code**

import java.io.\*;

import java.util.\*;

public class Main {

public static void permutations(int cb, int tb, int[] items, int ssf, int ts, String asf){

if(cb > tb){

if(ssf == ts){

System.out.println(asf);

}

return;

}

for(int i = 0; i < items.length; i++){

if(items[i] == 0){

items[i] = cb;

permutations(cb + 1, tb, items, ssf + 1, ts, asf + (i + 1));

items[i] = 0;

}

}

permutations(cb + 1, tb, items, ssf, ts, asf + 0);

}

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

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int nboxes = Integer.parseInt(br.readLine());

int ritems = Integer.parseInt(br.readLine());

permutations(1, nboxes, new int[ritems], 0, ritems, "");

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=NWFpebVkmTI&list=TLGG8cS7LJrrhd8xNzA3MjAyMQ). Please refer to it for a better understanding of the algorithm and the implementation.

* What is the ***time complexity*** of the above code?

At levels, we have boxes, and since there are n boxes, hence the maximum depth of the recursion tree will be equal to n.

Now, for each box, we have r + 1 choices, of placing any of the unplaced items, or not placing any item. But to find out the unplaced items, we need to iterate through all the items and check whether they are placed or not.

Hence, the total time complexity will be O(r + r + .... n times) = **O(rn)**.

* What is the ***space complexity*** of the above code?

We are using an integer items array, which will take **O(r)** space. Also, we are using recursion which will take **O(n)** function call stack space.

Hope that you liked the article on ***’Permutations - 2’***.

Subscribe to Pepcoding’s youtube channel for more such amazing video content on Data Structures & Algorithms. You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

Article Contributed by:

[Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)