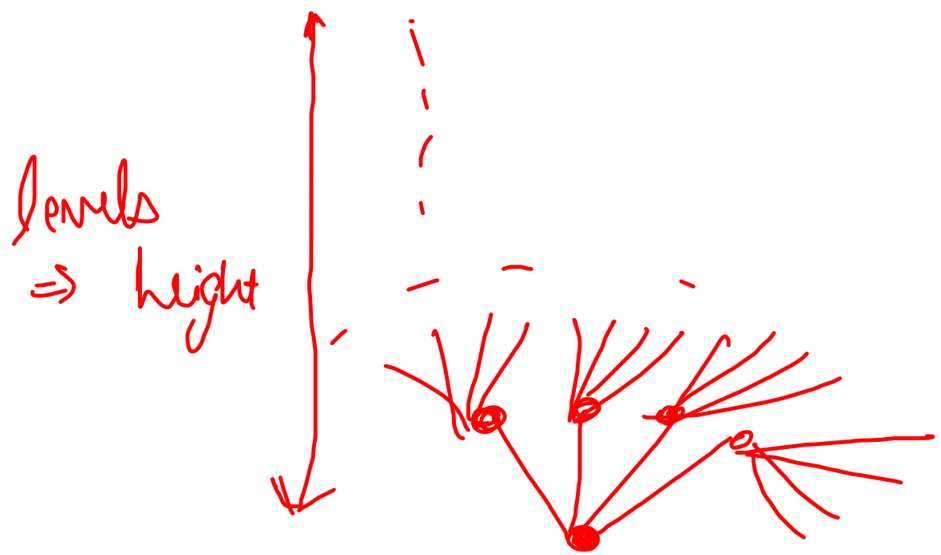
Recursion

1. Expectation
2. Faith
3. Meeting expectation with faith
4. Base Case

#Time Complexity

$(calls)^{height}$ + preorder * height + postorder * height



① Print Increasing

1.1 fun(n) : 1 --- n

1.2 fun(n-1) : 1 --- (n-1)

1.3 M.E ⇒ Postorder : Sys0(n);

n=0 : return;

② Print decreasing

1.1 fun(n) : n --- 1

1.2 fun(n-1) : (n-1) -- 1

1.3 M.E ⇒ Preorder : Sys0(n);

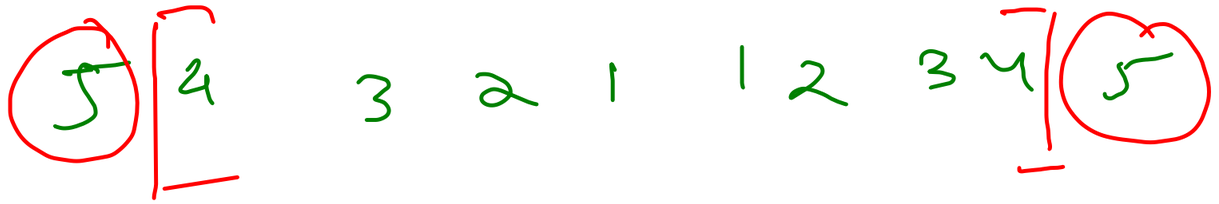
n=0 : return;

{ TC ⇒ $(1)^n$ + k*n = O(n) }

C++ Main

③ Print Increasing Decreasing

fun(5) :



fun(4):

4 3 2 1 1 2 3 4

M.E :→

Pre :
Syso(n)

fun(n-1)

Post
Syso(n)

TC ⇒ $\left(\frac{1}{}\right)^{n}$ + k * n + k * n

= 2 * n + 1 ⇒ O(n)

Power - linear

power(x, n)

$x^{n-1} = x * x * \dots$ (n-1) times

return $x * x^{n-1}$;

power(x, n)

$x * x * x * \dots$ n times

Power - logarithmic

$x^{n/2}$ = power(x, n/2)

if (n%2 == 0)
return $x^{n/2} * x^{n/2}$;

if (n%2 == 1)
return $x^{n/2} * x^{n/2} * x$;

Base Case : $n = 0$: return 1;

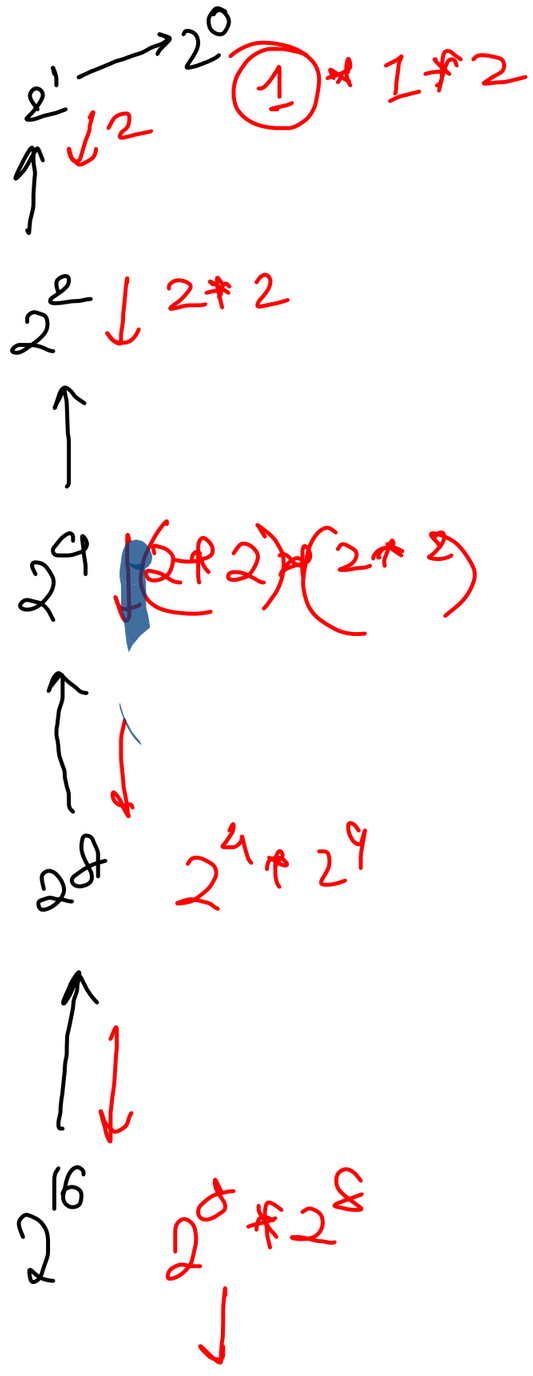
TC ⇒ $(1)^n + k * n \Rightarrow O(n)$

TC ⇒ $(1)^{\log_2 n} + k * \log_2 n$

⇒ $O(\log n) \simeq O(1)$

Power - log

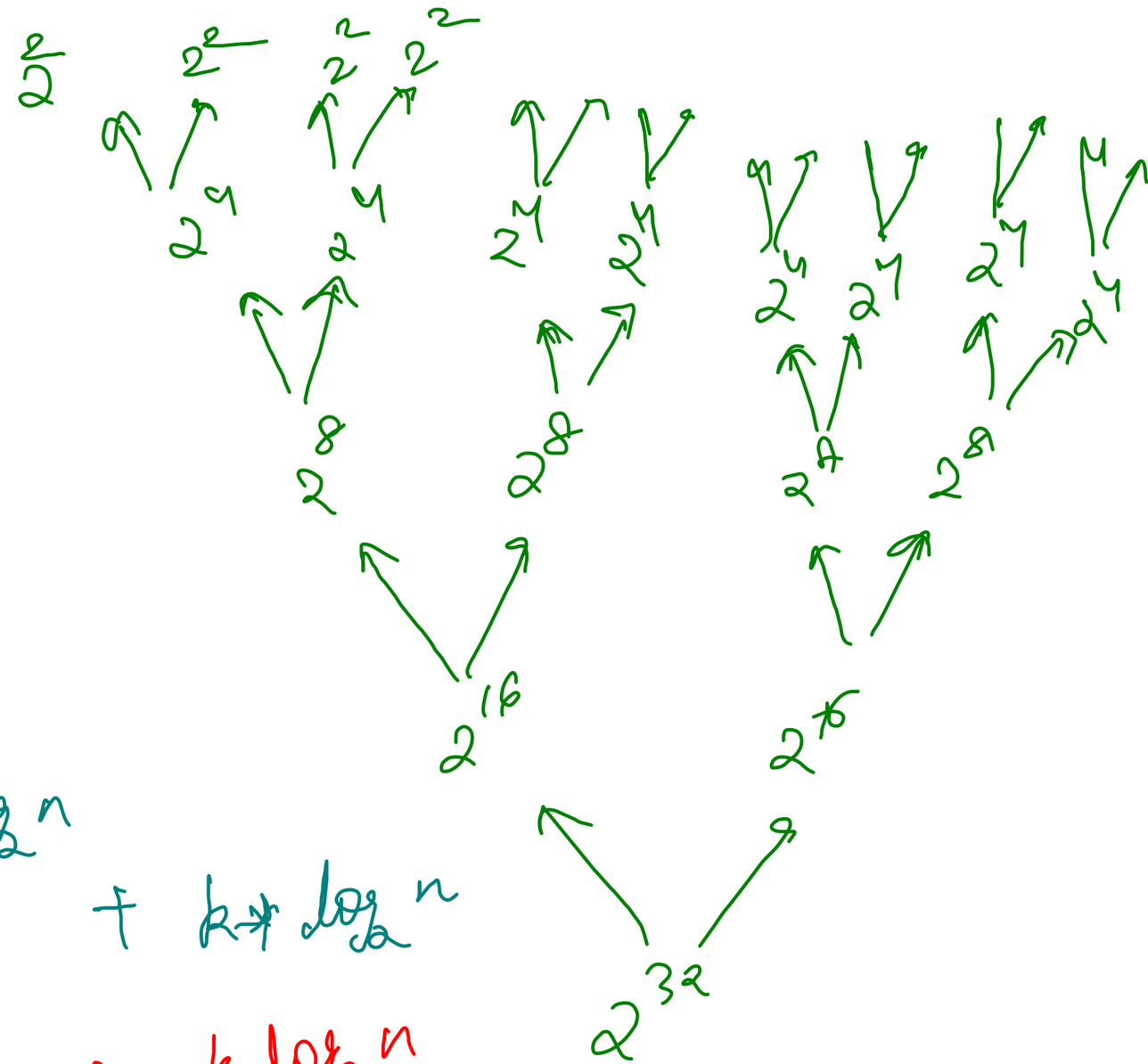
log n height




```java
public static int power(int x, int n){
    // Base Case x^0 = 1
    if(n == 0) return 1;

    // 1. Faith : x ^ n/2
    int xpnb2 = power(x, n/2);

    // 2. x^n = x^n/2 * x^n/2
    int xpn = xpnb2 * xpnb2;

    // 3. If n is odd
    if(n % 2 == 1) xpn = xpn * x;

    // 4. Return
    return xpn;
}
```

Power - 2

```
if(n%2==0)
  return power(x, n/2) * power(x, n/2)
else
  retn pow(x, n/2) * pow(x, n/2) * x;
```



height $\Rightarrow \log_2 n$

calls $\Rightarrow 2$

$(2)^{\log_2 n} + k * \log_2 n$

$= n + k \log_2 n$

$\Rightarrow O(n)$

long

① 10^{18} % mod

② → (a+b) % m

Because AB

int max value

$10^9 + 7$

long ⇒ $\approx 10^{18}$

int ⇒ $2^{31} - 1 \approx 10^9$

(a+b) % m

(a*b) % m

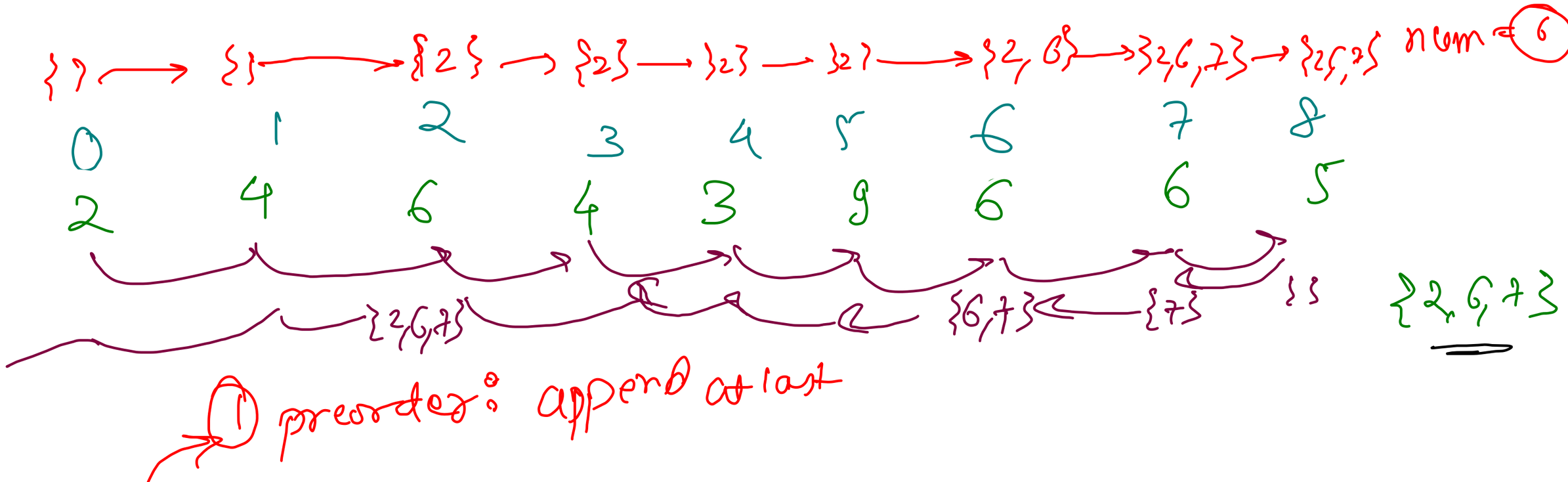
Modulus formulas

1. $(a+b) \% m = (a \% m + b \% m) \% m$

2. $(a-b) \% m = ((a \% m - b \% m) \% m + m) \% m$

3. $(a * b) \% m = (a \% m) * (b \% m) \% m$

4. Division $\Rightarrow$ **Modulo Inverse**

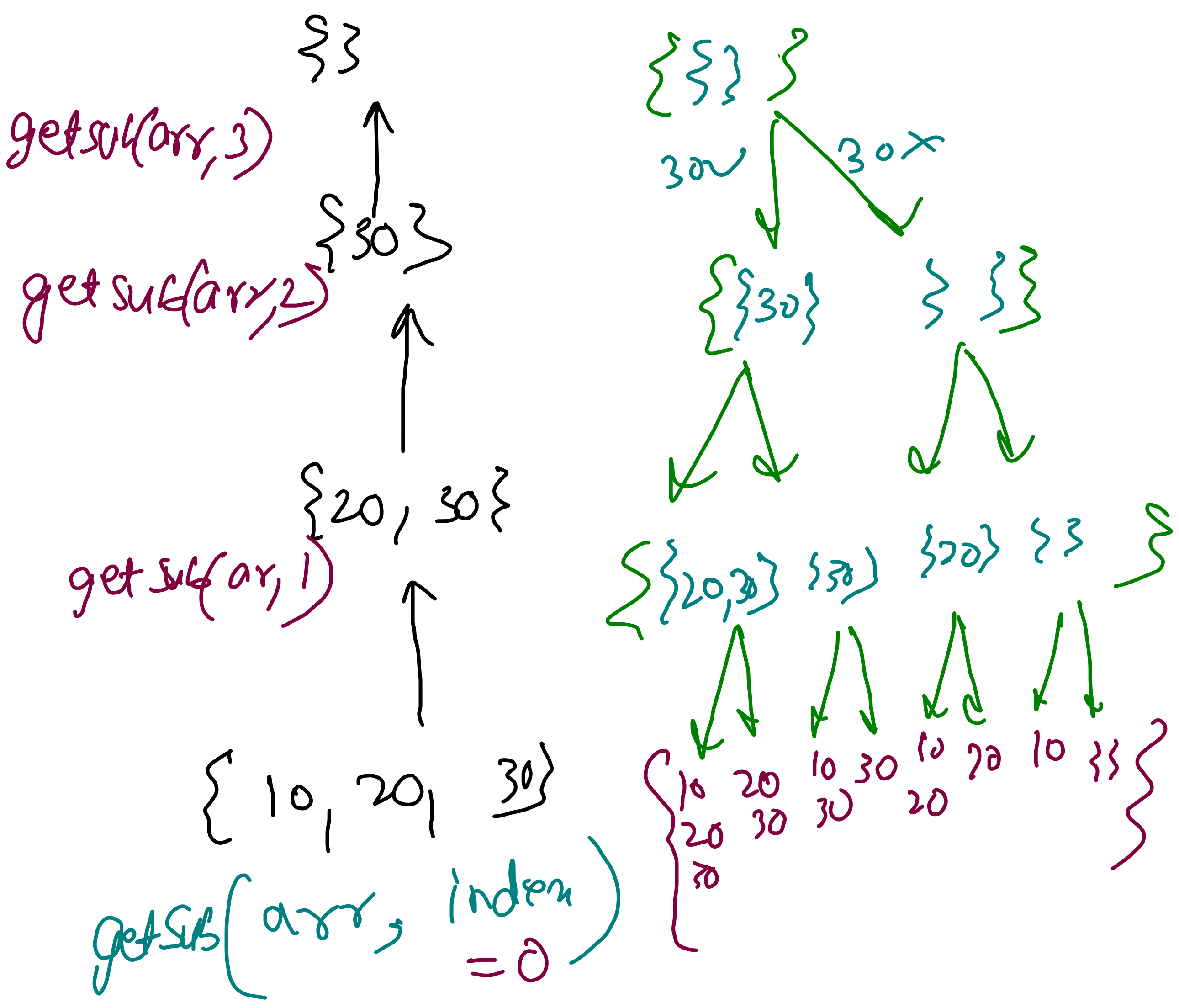


① preorder: append at last

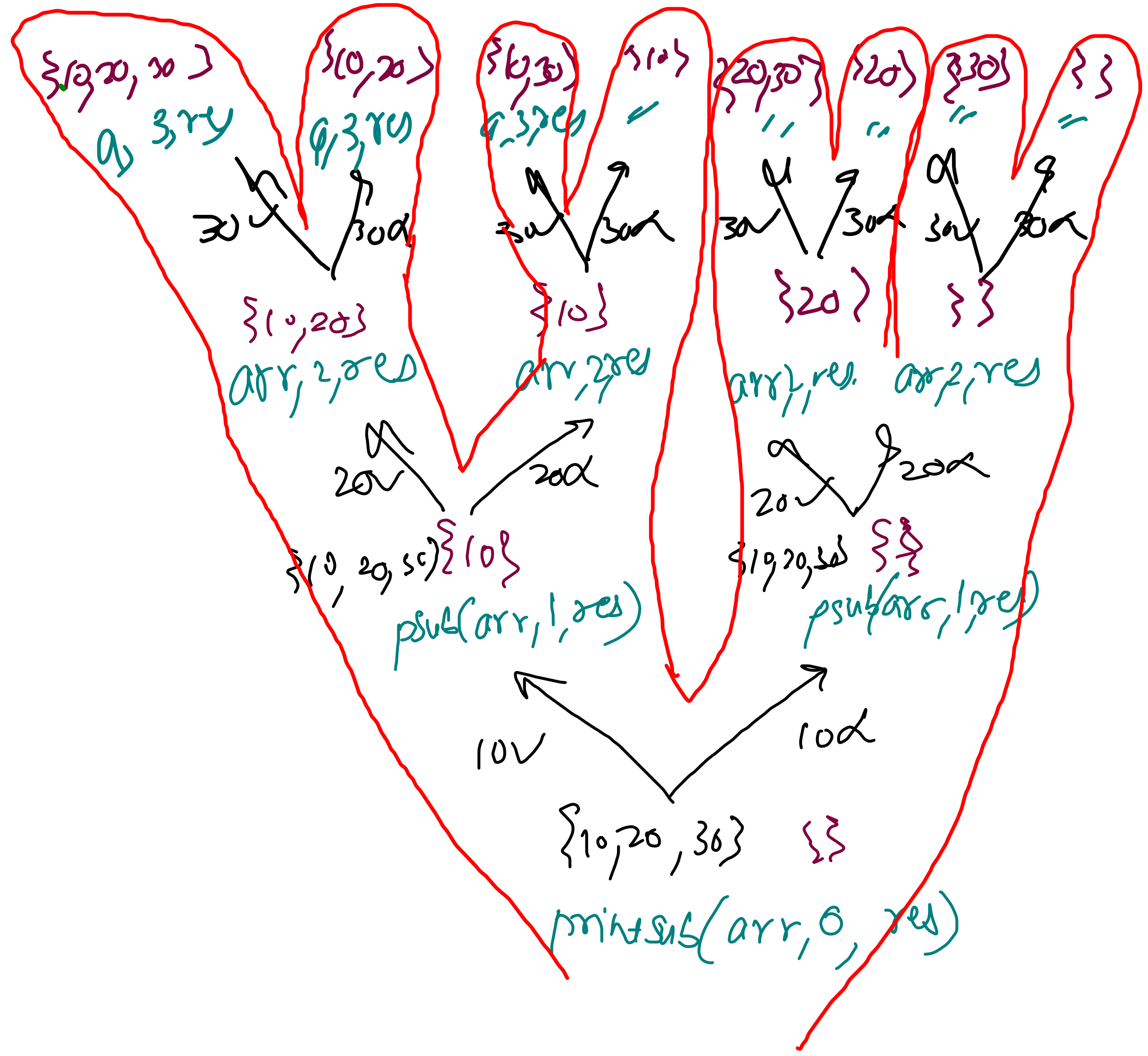
fun(arr, 0, res, num)

② postorder: append at first

Get Subsequence



Print Subsequence



$$TC \Rightarrow \left(2^n + k*n + k*n \right) \Rightarrow O(2^n)$$

Today's Target

1. Permutations
   - → Box on level
   - → Item on level
   - → Swap method
2. Combinations
   - → Box on level
   - → Item on level

15-20 → follow up } → 30 Q

3. String Permutations {Unique}
   - → Box on level
   - → Item on level
4. Palindromic Permutations

Homework :-

- Factor combinations
- Dictionary Order
  - → small
  - → large

Combinatn

^nC_r

$$\frac{n!}{(n-r)!\,r!}$$

i i — —

4C2

identical items

Permutation

Selectn + Arrangement

^nP_r

$$\frac{n!}{(n-r)!}$$

non-identical items

12 — — 21 — —

1 — 2 — 2 — 1 —

1 — — 2 2 — — 1

— 12 — — 21 —

— 1 — 2 — 2 —

— — 12 — — 21

$^4P_2 = {}^4C_2 * 2!$

Combinations → Box on level

box = 3

box = 2

box = 1

box = 0

{1, 2}

n = 4

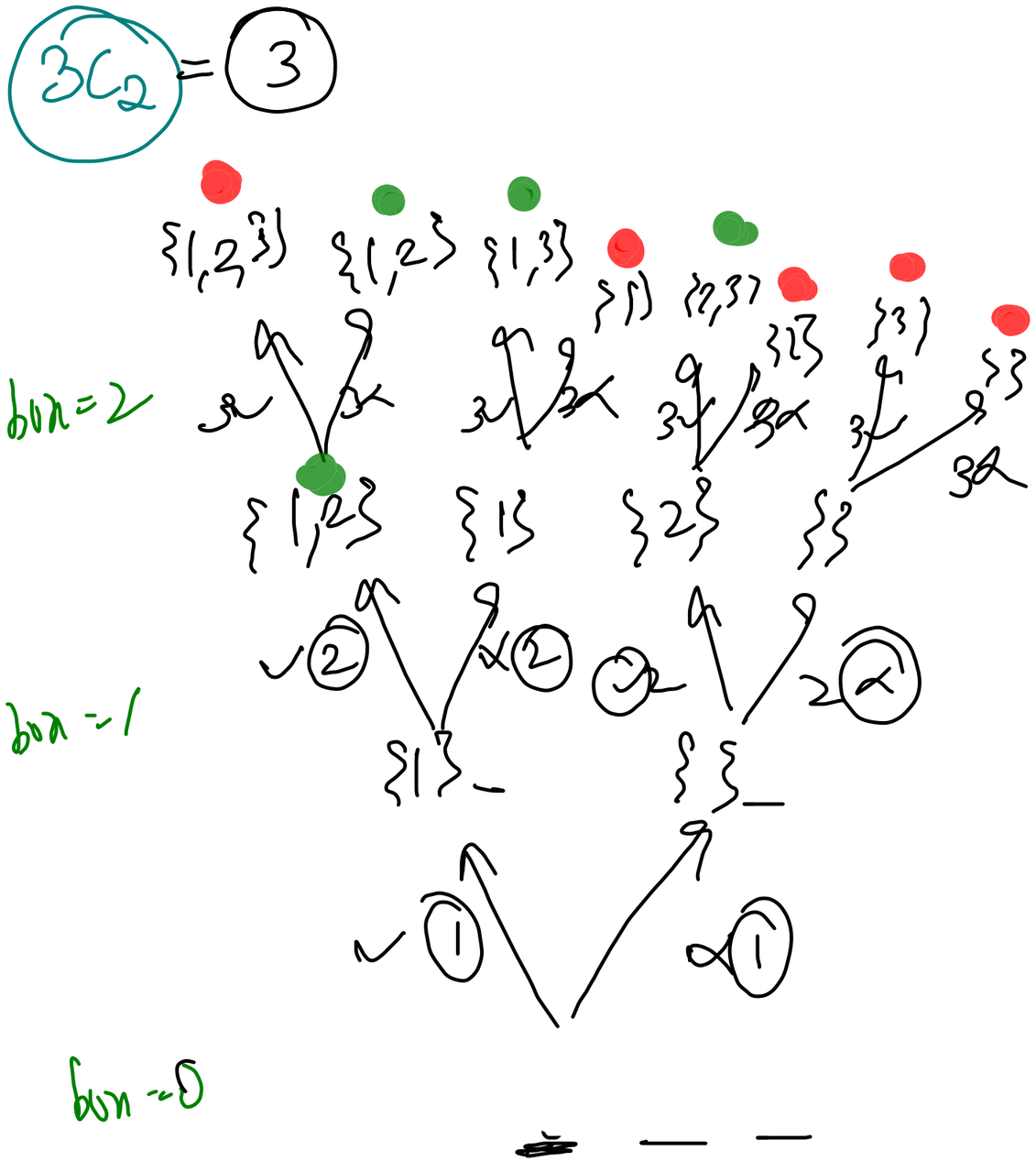
r = 2

$$2^n = {}^nC_0 + {}^nC_1 + {}^nC_2 + \cdots {}^nC_n$$

$$2^4 = {}^4C_0 + {}^4C_1 + {}^4C_2 + {}^4C_3 + {}^4C_4$$

$$2^4 = 1 + 4 + 6 + 4 + 1$$

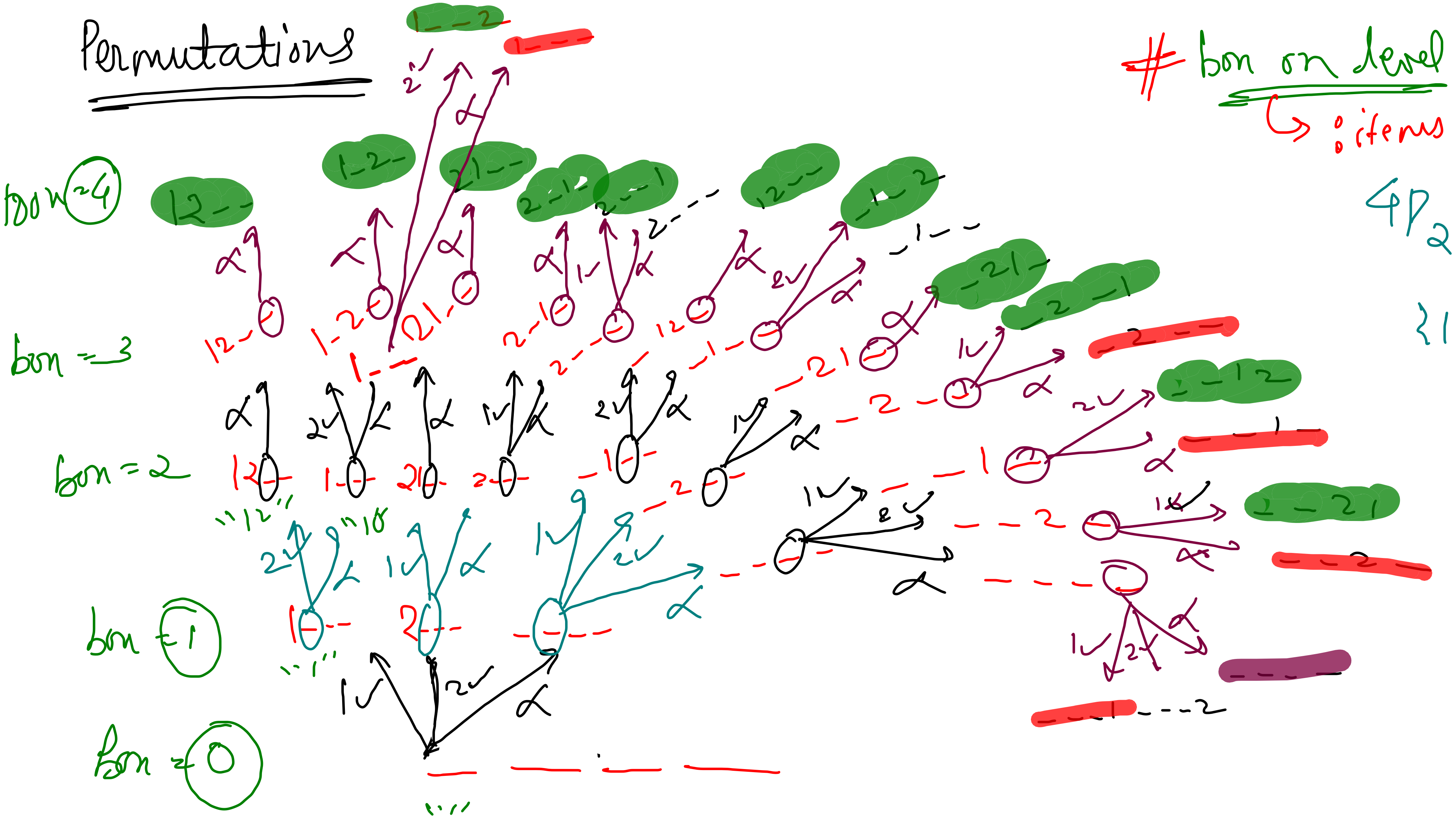
TC ⇒ $(2)^n + \cdots + \cdots$




```java
public void combine(List<List<Integer>> combinations,
    List<Integer> combination, int currentBox, int n, int k){
    if(currentBox == n){
        if(combination.size() == k){
            // deep copy
            List<Integer> temp = new ArrayList<>(combination);
            combinations.add(temp);
        }
        return;
    }

    // options -> current Box -> item should be placed or not

    // yes
    combination.add(currentBox + 1);
    combine(combinations, combination, currentBox + 1, n, k);
    combination.remove(combination.size() - 1);

    // no
    combine(combinations, combination, currentBox + 1, n, k);
}
```

{1,2} , {1,3} , {2,3}

Permutations



box on level

↳ items were on option + α

$^4P_2$

{1, 2}

$^4P_0 + {}^4P_1 + {}^4P_2$

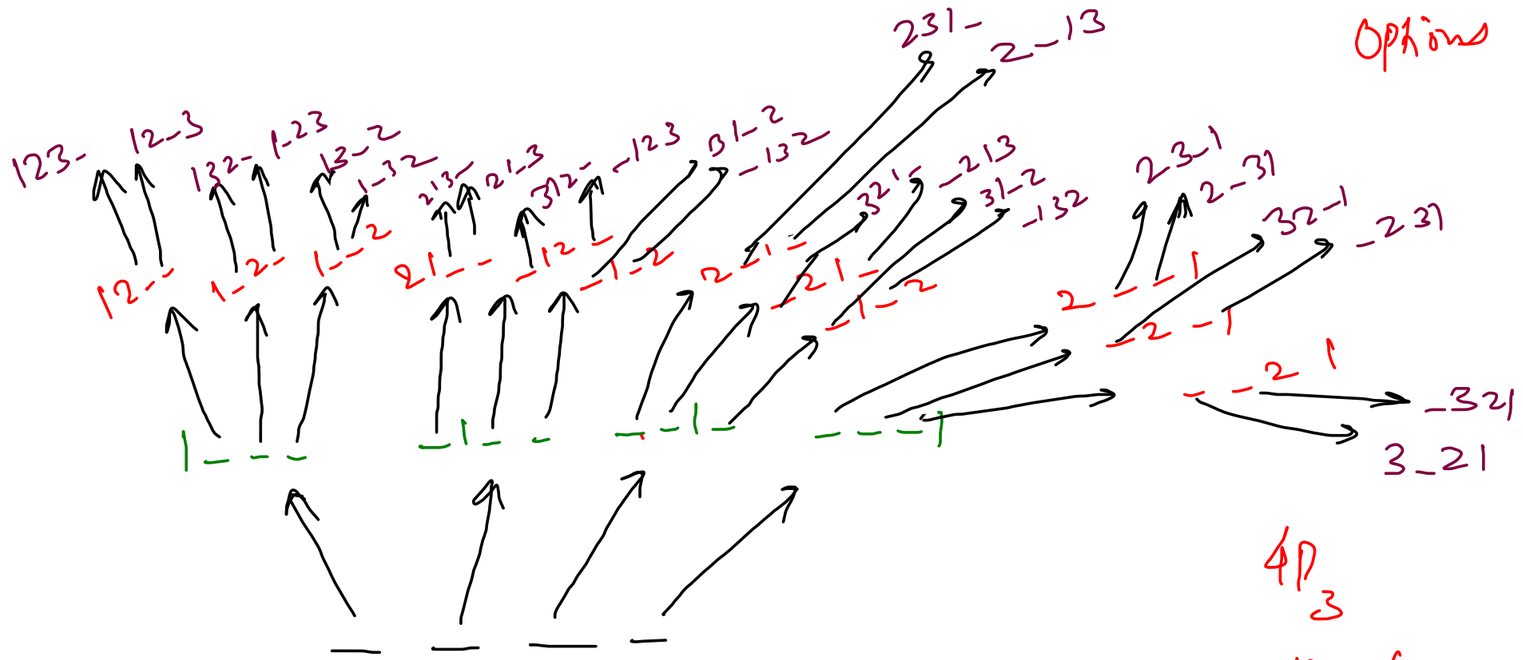
Permutation - 2

items on level : boxes are Options

item = 3

item = 2

item = 1



$^4P_3$

n = 4

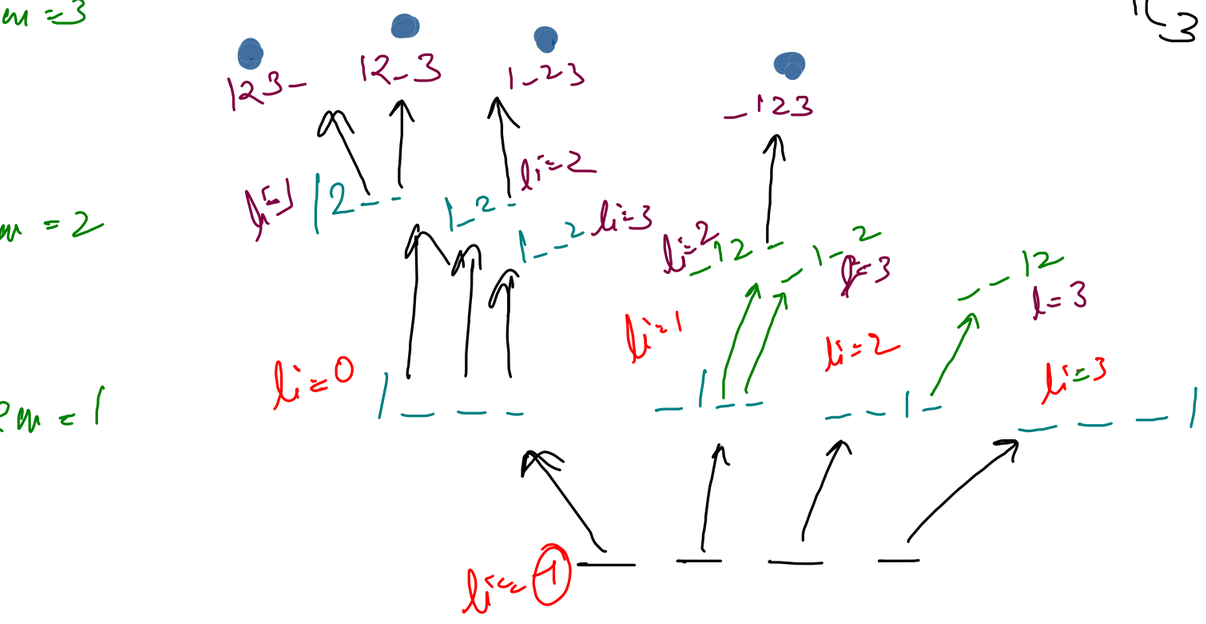
b = 3

```
public static void permutations(int[] boxes, int ci, int ti){
  // write your code here
  if(ci == ti){
      for(int val: boxes) System.out.print(val);
      System.out.println();
  }

  // Item -> Chose box
  for(int i=0; i<boxes.length; i++){
      if(boxes[i] == 0){
          boxes[i] = ci + 1;
          permutations(boxes, ci + 1, ti);
          boxes[i] = 0;
      }
  }
}
```

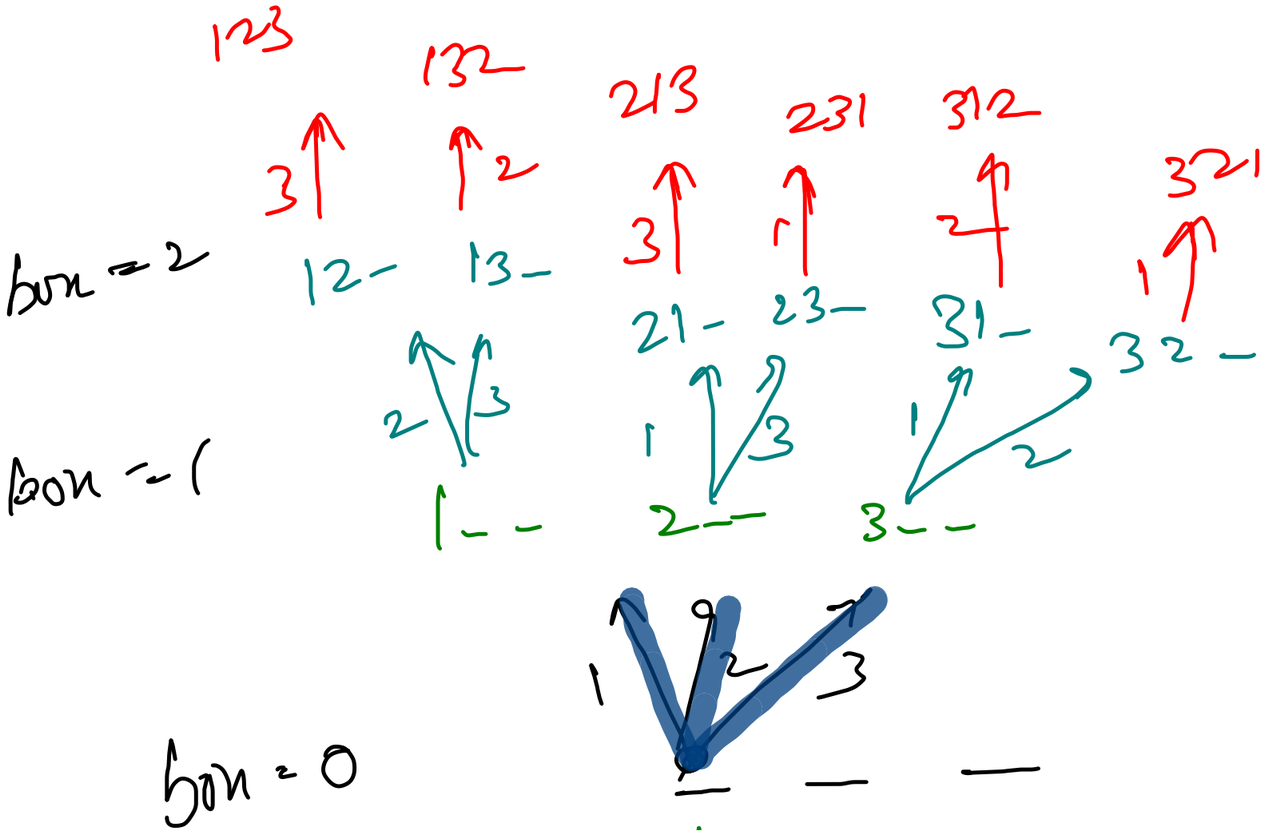
Combinations -2

{Items ⇒ level ; Option ⇒ box}

$^4C_3 = 4$




```java
public static void combunations(int[] boxes, int ci, int ti, int lastItemIdx){
  // write your code here
  if(ci == ti){
      for(int val: boxes){
          if(val == 0) System.out.print("-");
          else System.out.print("i");
      }

      System.out.println();
  }

  // Item -> Chose box
  for(int i=lastItemIdx + 1; i<boxes.length; i++){
      if(boxes[i] == 0){
          boxes[i] = ci + 1;
          combunations(boxes, ci + 1, ti, i);
          boxes[i] = 0;
      }
  }
}
```

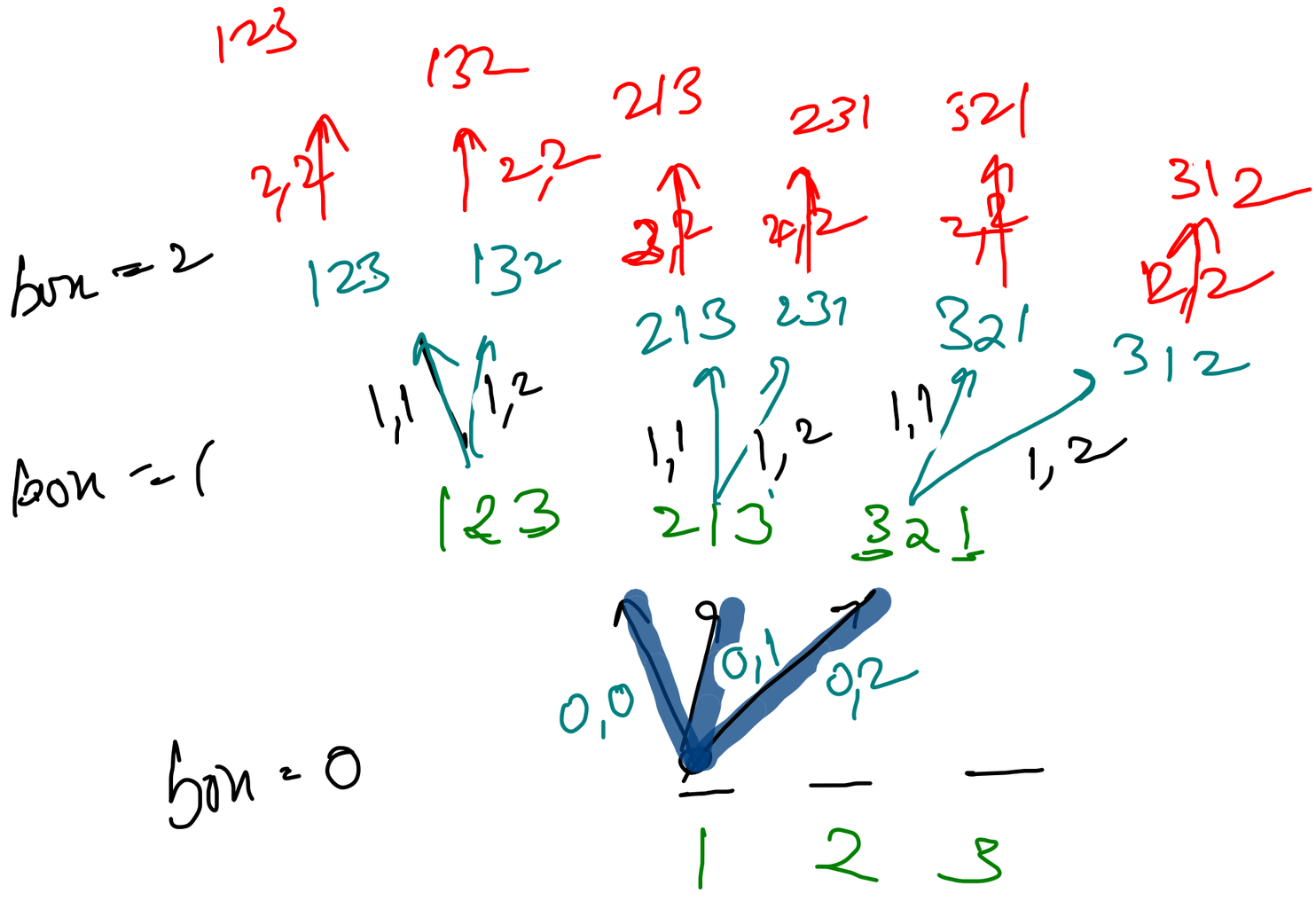


box = items

items

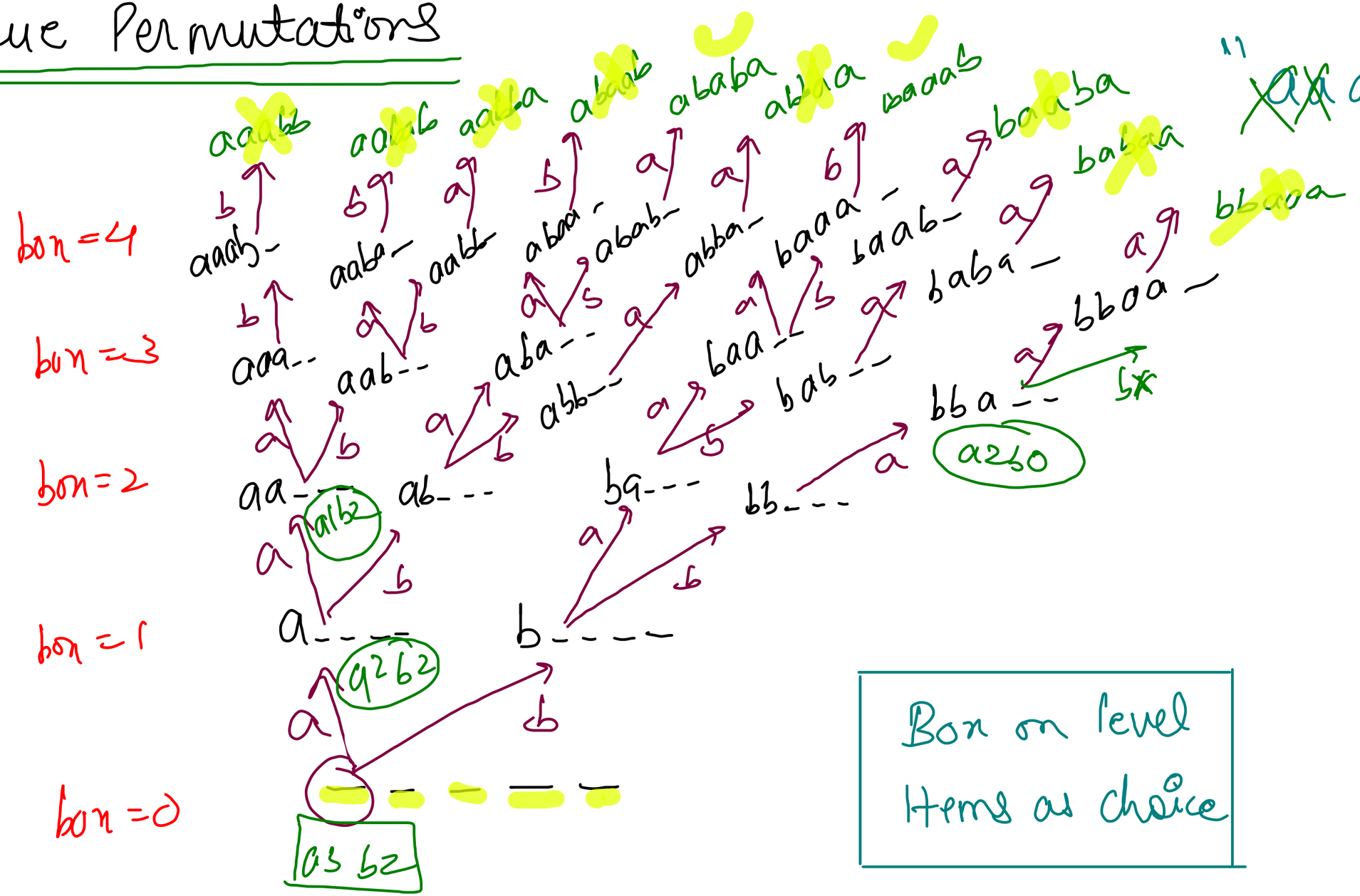
boxes = items

{1, 2, 3}




```java
public void permutations(int currentBox, int[] nums){
  if(currentBox == nums.length){
      List<Integer> copy = new ArrayList<>();
      for(int i=0; i<nums.length; i++){
          copy.add(nums[i]);
      }
      res.add(copy);
      return;
  }

  for(int i=currentBox; i<nums.length; i++){
      swap(nums, currentBox, i);
      permutations(currentBox + 1, nums);
      swap(nums, currentBox, i);
  }
```

Unique Permutations



"aaab"

= 5!

3! x 2! x 1!

= 5 x 4 x 3 x 2

= 10

Box on level
Items as choice

helper

```java
public void permutations(HashMap<Integer, Integer> freq, int[] nums,
        int currentBox, List<Integer> ans){
    if(currentBox == nums.length){
        List<Integer> copy = new ArrayList<>(ans);
        res.add(copy);
    }

    for(Integer key: freq.keySet()){
        int val = freq.get(key);
        if(val > 0){
            freq.put(key, val - 1);
            ans.add(key);

            permutations(freq, nums, currentBox + 1, ans);

            ans.remove(ans.size() - 1);
            freq.put(key, val);
        }
    }
}
```

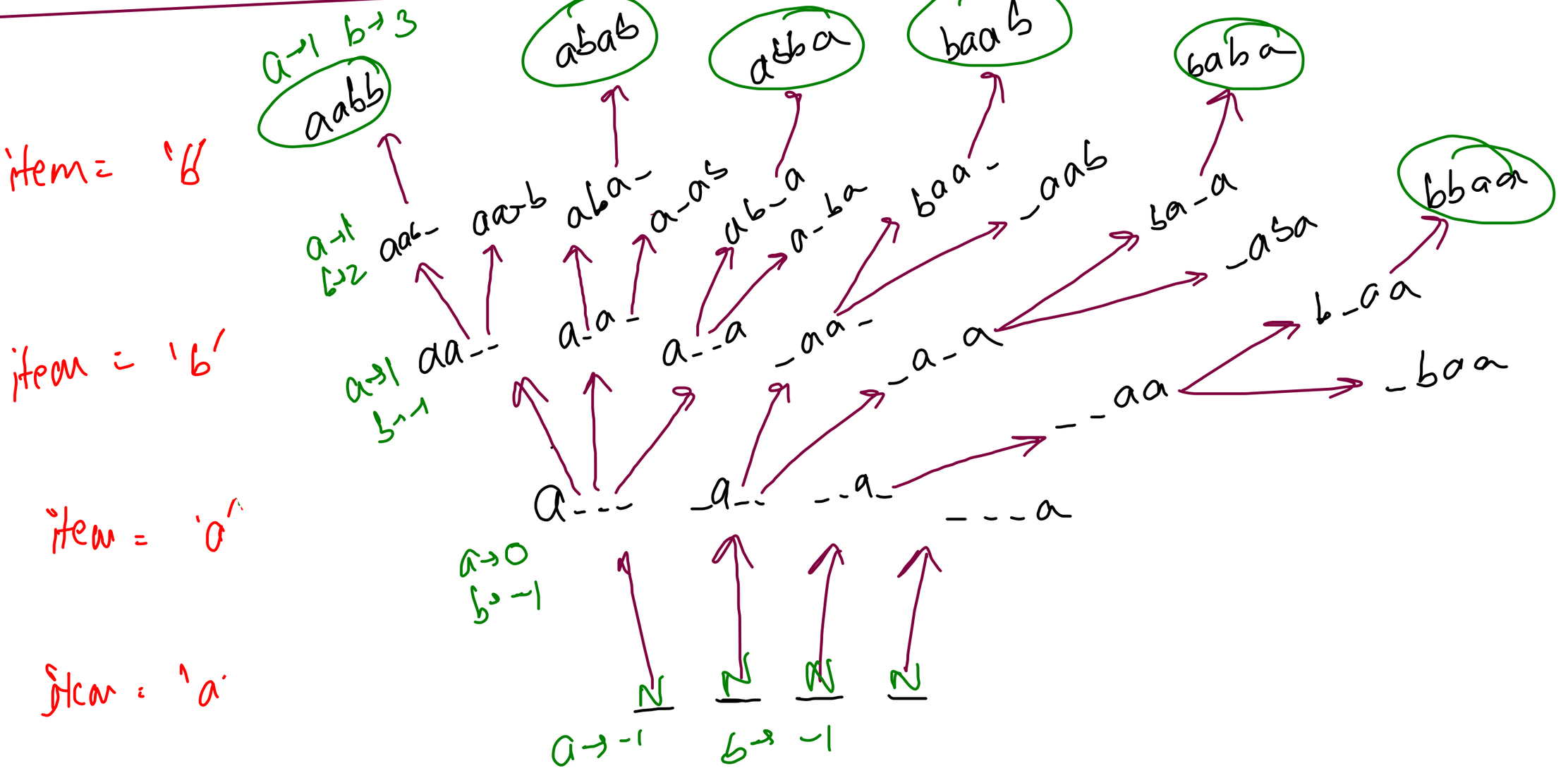
main

```java
HashMap<Integer, Integer> freq = new HashMap<>();
for(int i=0; i<nums.length; i++){
    if(freq.containsKey(nums[i])){
        int val = freq.get(nums[i]);
        freq.put(nums[i], val + 1);
    } else {
        freq.put(nums[i], 1);
    }
}
res = new ArrayList<>();
permutations1(freq, nums, 0, new ArrayList<>());
```

Item on level
Box as choice

'aabb'

$$\frac{4C}{2!2!} = \frac{4\times3\times2}{2\times2} = 6$$



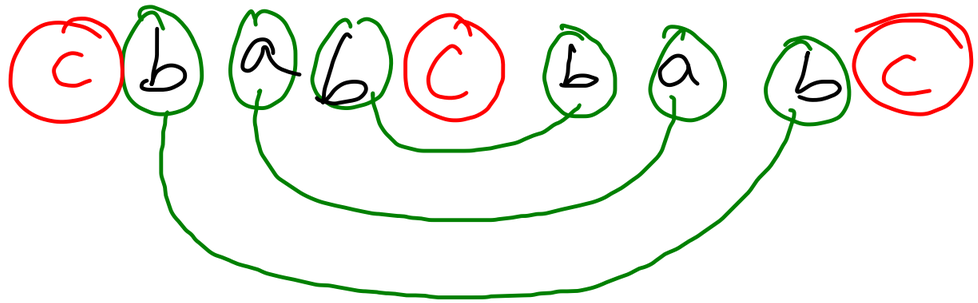
helper

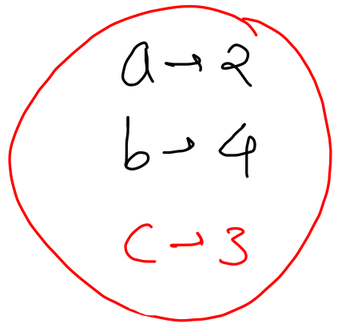
```
public void permutations2(HashMap<Integer, Integer> lastIdx, int[] nums,
        int currentItem, List<Integer> ans){
    if(currentItem == nums.length){
        List<Integer> copy = new ArrayList<>(ans);
        res.add(copy);
        return;
    }

    int st = lastIdx.get(nums[currentItem]);

    for(int i=st+1; i<ans.size(); i++){
        if(ans.get(i) == null){
            ans.set(i, nums[currentItem]);
            lastIdx.put(nums[currentItem], i);

            permutations2(lastIdx, nums, currentItem + 1, ans);

            lastIdx.put(nums[currentItem], st);
            ans.set(i, null);
        }
    }
}
```

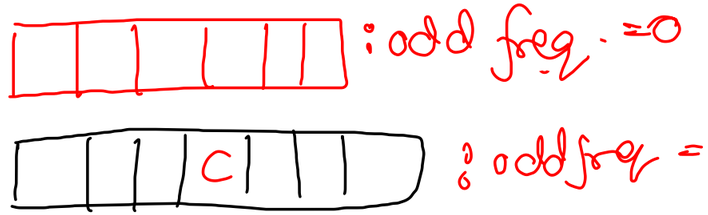
main

```
HashMap<Integer, Integer> lastIdx = new HashMap<>();
for(int i=0; i<nums.length; i++){
    lastIdx.put(nums[i], -1);
}

List<Integer> ans = new ArrayList<>();
for(int i=0; i<nums.length; i++){
    ans.add(null);
}

res = new ArrayList<>();
permutations2(lastIdx, nums, 0, ans);
return res;
```

#Palindrome Permutation - 1







all char ✓ even freq

atmost 1 char with odd freq

```java
public boolean canPermutePalindrome(String s) {
    HashMap<Character, Integer> freq = new HashMap<>();
    for(int i=0; i<s.length() ; i++){
        if(freq.containsKey(s.charAt(i)) == false){
            freq.put(s.charAt(i), 1);
        } else {
            freq.put(s.charAt(i), freq.get(s.charAt(i)) + 1);
        }
    }

    int oddCount = 0;
    for(Character c: freq.keySet()){
        if(freq.get(c) % 2 == 1){
            oddCount++;
        }
    }

    if(oddCount > 1) return false;
    return true;
}
```

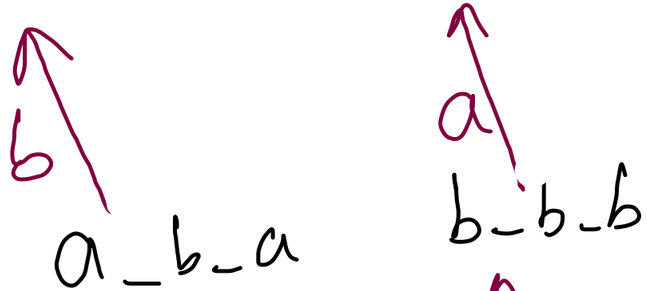
All Palindromic Permutations {Pg-11}

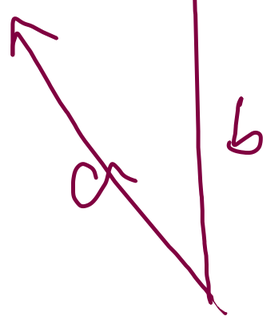
aabbb

a2b2

abbba

baba b





box on level

b

aaaabbc

a4b2c

aabcbaa

5

aa_c_aa

a

b

a _ c _ a

a2b2c0

abababa

a

ab_c_ba

baacaaca

a

ba_c_ab

a

b _ c _ b

a

b

C

— — — — — —

CB

1a4b2c0

n-1-CB

box on level item as choice

K words { Repitition not allowed } {Combinations}

a2 b3 c2 d2 e

aabbbccdde

3

unique characters : 5 bcos

1. Bcos on level
2. Hence on level :→ 5 items
   → k characters

a, b, c, d, e

↓

3 items

{ a b c d e }

_ _ _ _ _

$^5C_3 = \frac{5*4*3*2}{3*2*1*2} = 10$

{ abc abd abe acd
ace ade bcd bce
bde cde }

abcd abc- ab-d a-cd a-c- a--d a--- -bcd -bc- -b-d --cd

(d) dα dα

abc- ab-- a-c- a--- -bc- -b-- --c- ---- --c-

(c) cα cα cα cα

ab-- a--- -b-- ---- ---d

(b) bα bα

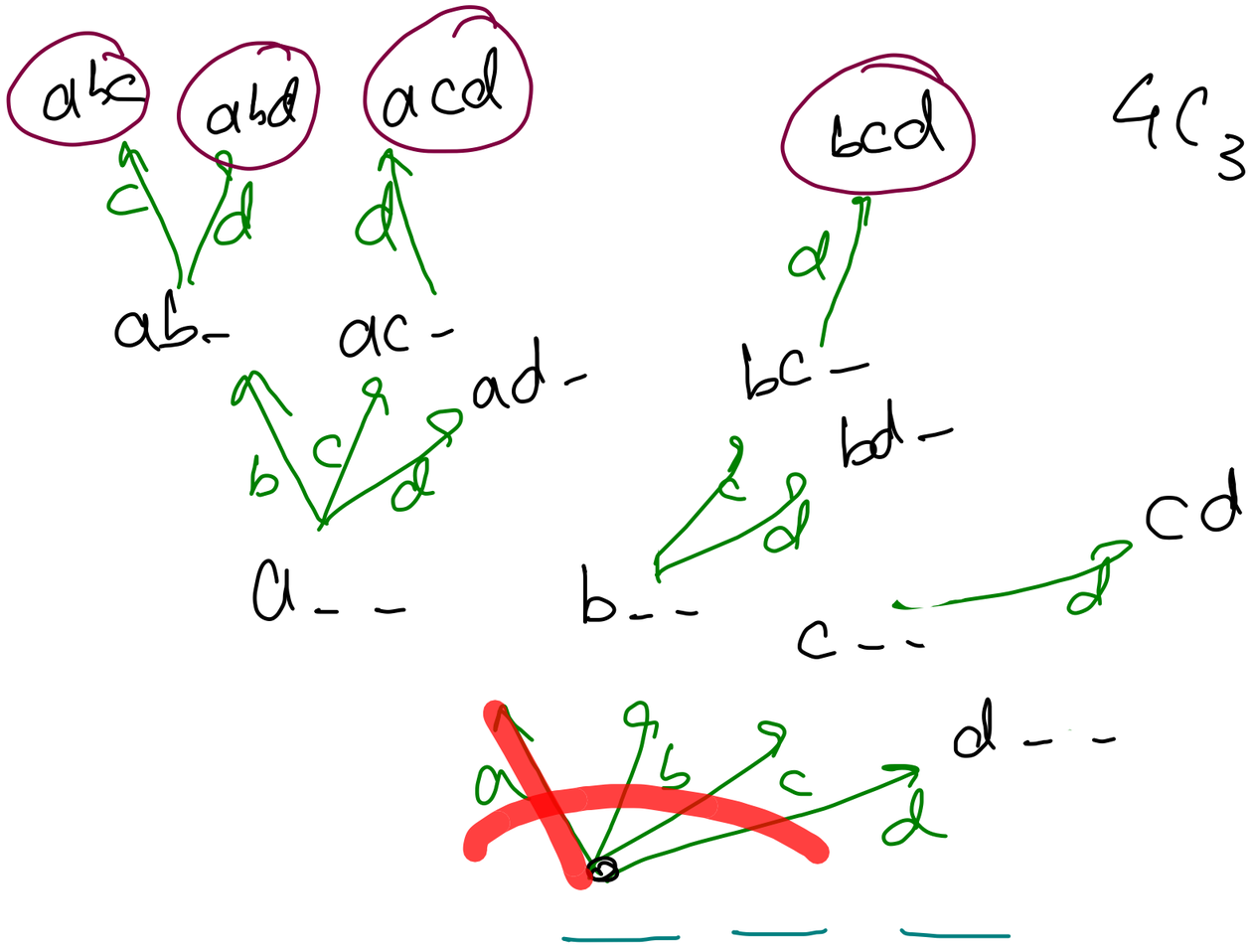
a---

(a) aα

a2b3c2d2

{a, b, c, d}

$2^4 = {}^4C_0 + {}^4C_1 + {}^4C_2 + {}^4C_3 + {}^4C_4$



$4C_3 = 4$

$O((k)^{\text{unique char}})$

```
public static void combination(ArrayList<Character> chs, int lastIdx, String res, int k){
  if(res.length() == k){
      System.out.println(res);
      return;
  }

  for(int i=lastIdx + 1; i<chs.size(); i++){
      combination(chs, i, res + chs.get(i), k);
  }
}
```

a2b3c2d

{a,b,c,d}