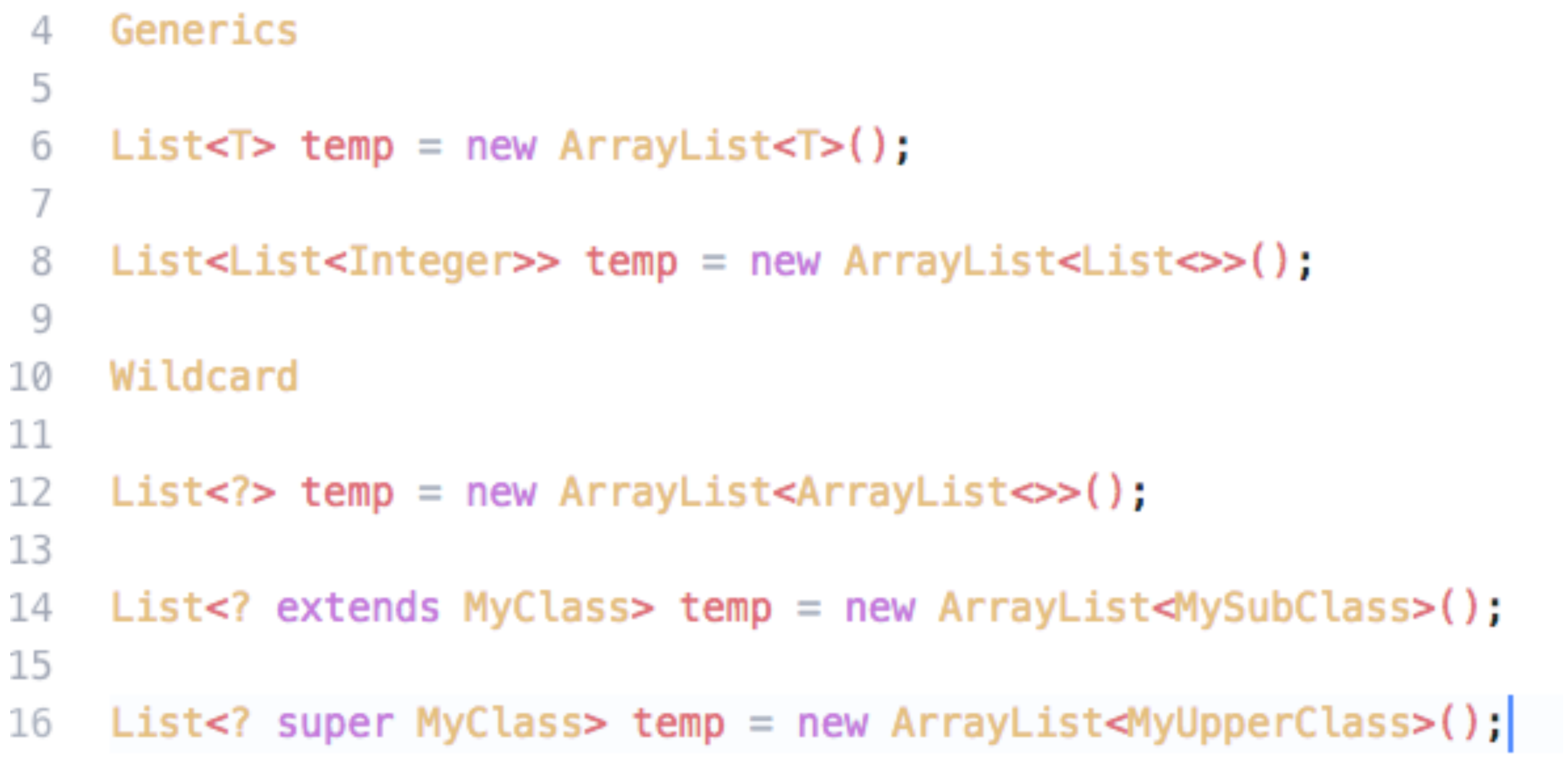
BITTIGER

CLASS_4
BINARY STRUCTURE

Content of Class_4

DFS（**排列组合问题模板**）	***DFS (*****染色*****)***
Permutations 1 2	*Number of Island*
Subsets 1 2	

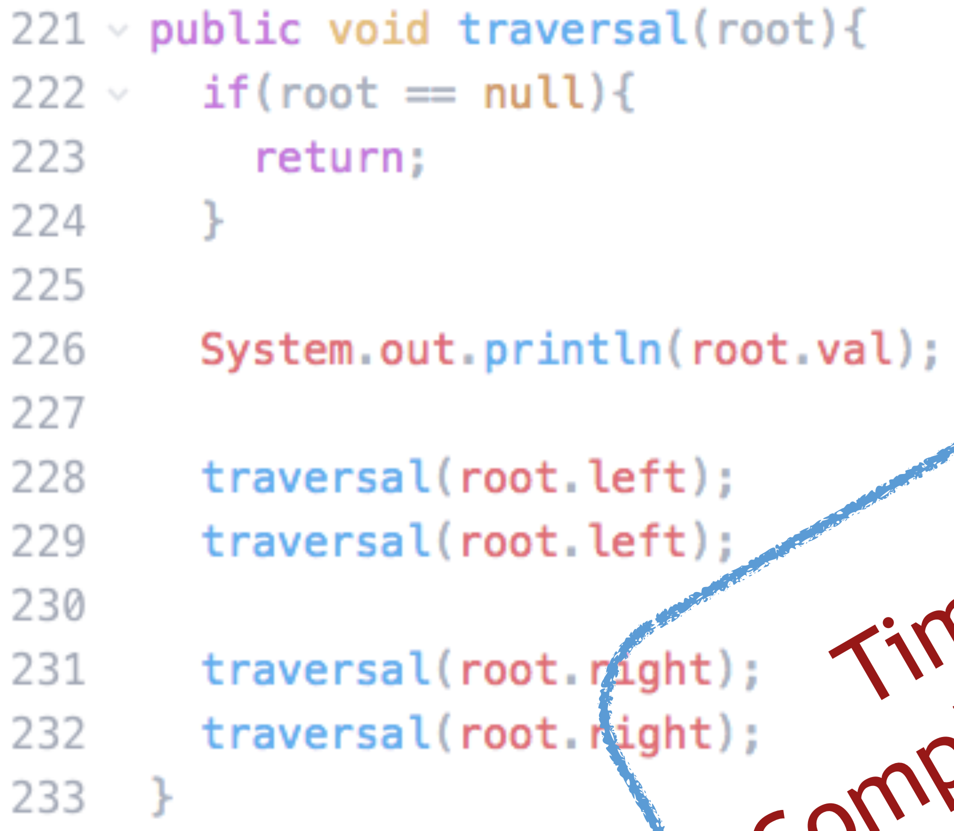


```
Generics

List<T> temp = new ArrayList<T>();

List<List<Integer>> temp = new ArrayList<List<>>();

Wildcard

List<?> temp = new ArrayList<ArrayList<>>();

List<? extends MyClass> temp = new ArrayList<MySubClass>();

List<? super MyClass> temp = new ArrayList<MyUpperClass>();
```

Tree Time Complexity

Number of Nodes in Tree: N

Tree Height: LogN



```
public void traversal(root){
  if(root == null){
    return;
  }

  System.out.println(root.val);

  traversal(root.left);
  traversal(root.left);

  traversal(root.right);
  traversal(root.right);
}
```

Time Complexity ?

画　recursion　tree

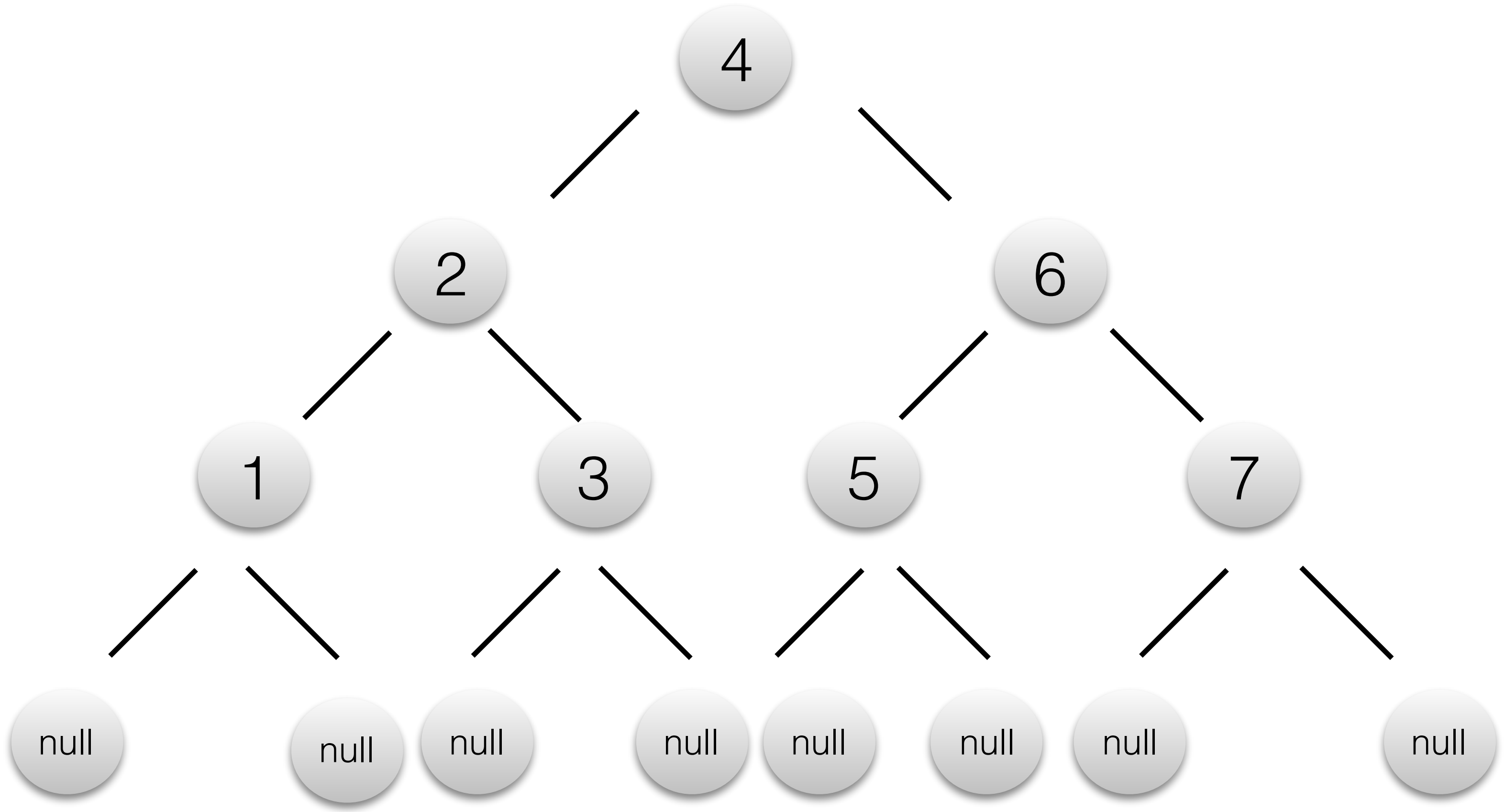
二叉树 ——> 四叉树（树高不变）

Recursion 三步走

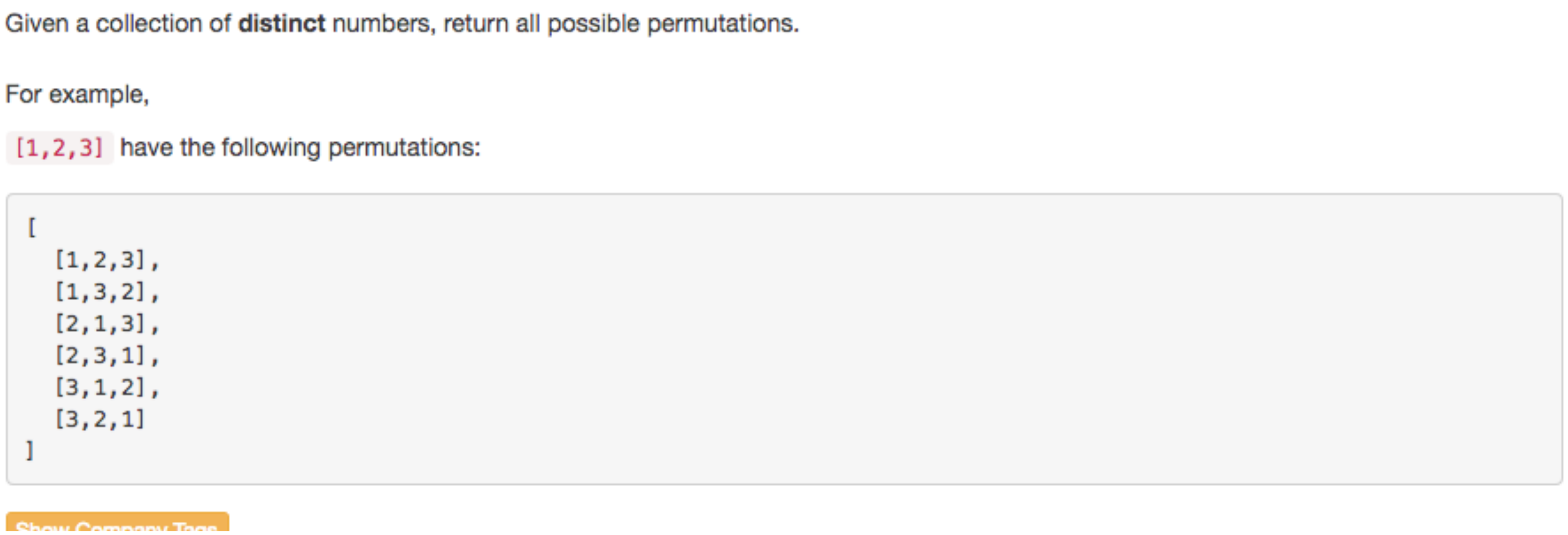
Base / Corner Case

Current Layer Logic

Next Lay Logic



46. Permutations



Given a collection of **distinct** numbers, return all possible permutations.

For example,

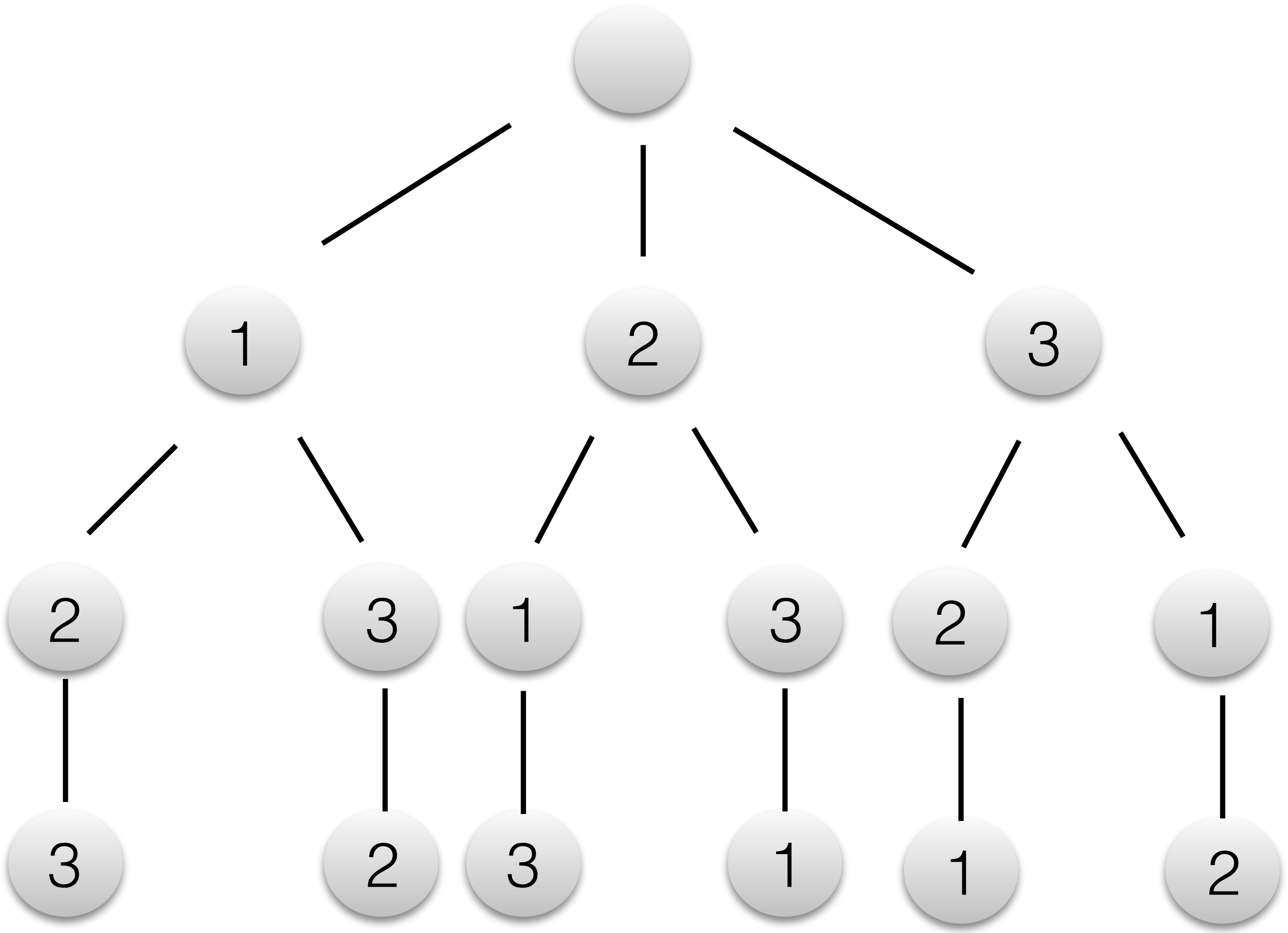
`[1,2,3]` have the following permutations:

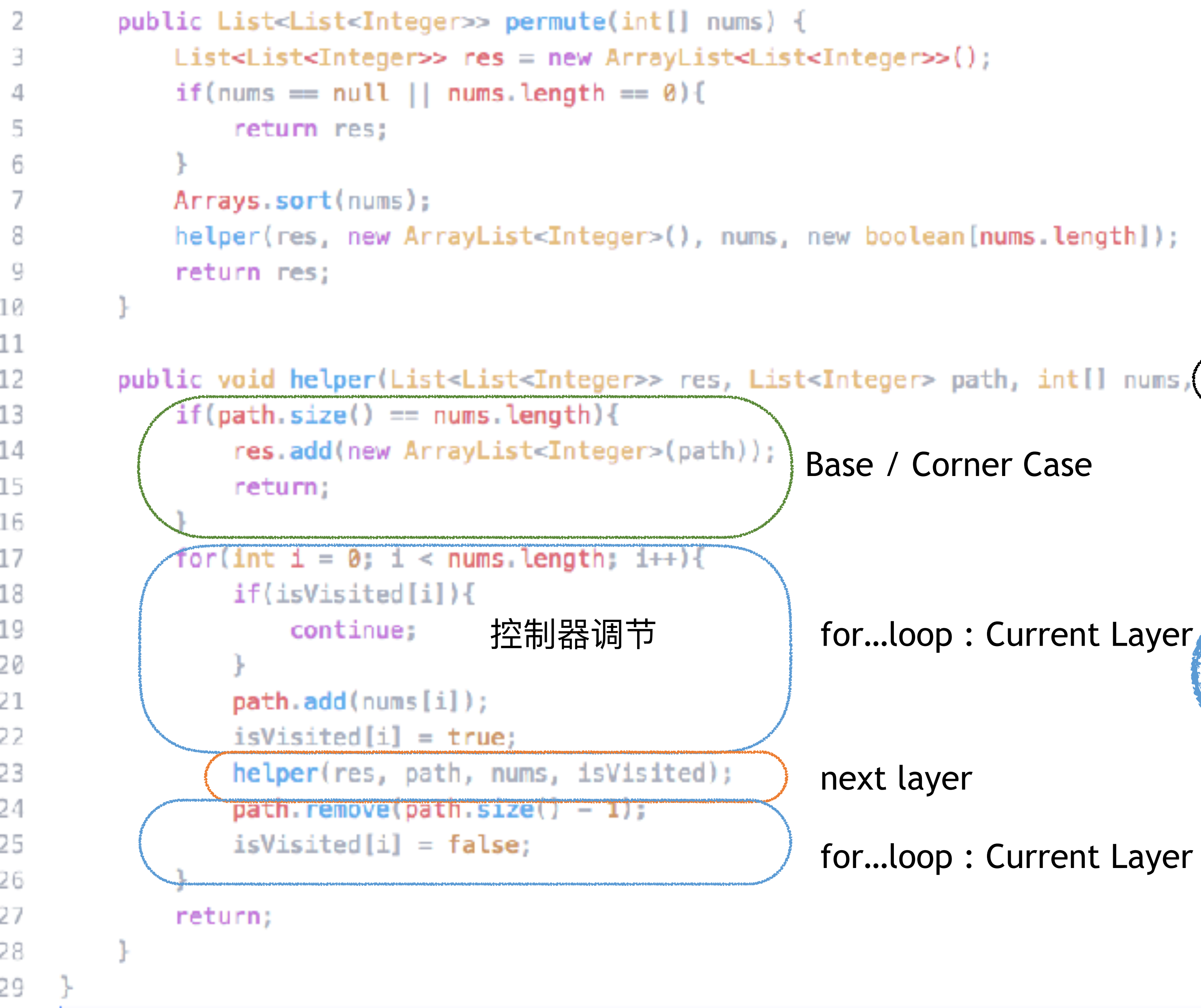
```
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

10 min

```
public List<List<Integer>> permute(int[] nums) {

}
```

Recursion Tree





```java
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        if(nums == null || nums.length == 0){
            return res;
        }
        Arrays.sort(nums);
        helper(res, new ArrayList<Integer>(), nums, new boolean[nums.length]);
        return res;
    }

    public void helper(List<List<Integer>> res, List<Integer> path, int[] nums, boolean[] isVisited){
        if(path.size() == nums.length){
            res.add(new ArrayList<Integer>(path));
            return;
        }
        for(int i = 0; i < nums.length; i++){
            if(isVisited[i]){
                continue;
            }
            path.add(nums[i]);
            isVisited[i] = true;
            helper(res, path, nums, isVisited);
            path.remove(path.size() - 1);
            isVisited[i] = false;
        }
        return;
    }
}
```

控制器（去重，起始位置）

Base / Corner Case

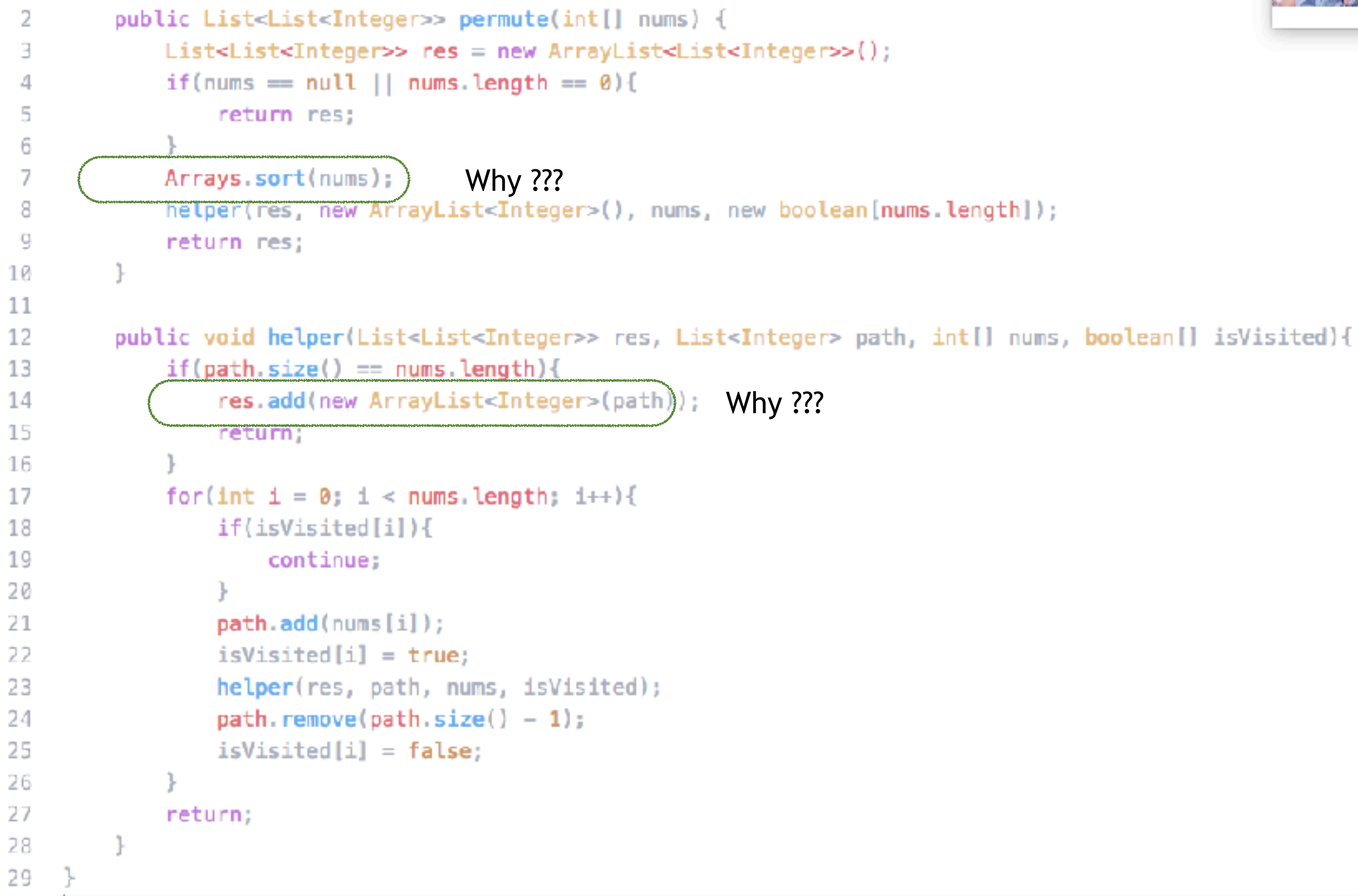
控制器调节

for…loop : Current Layer

next layer

for…loop : Current Layer

DFS模板

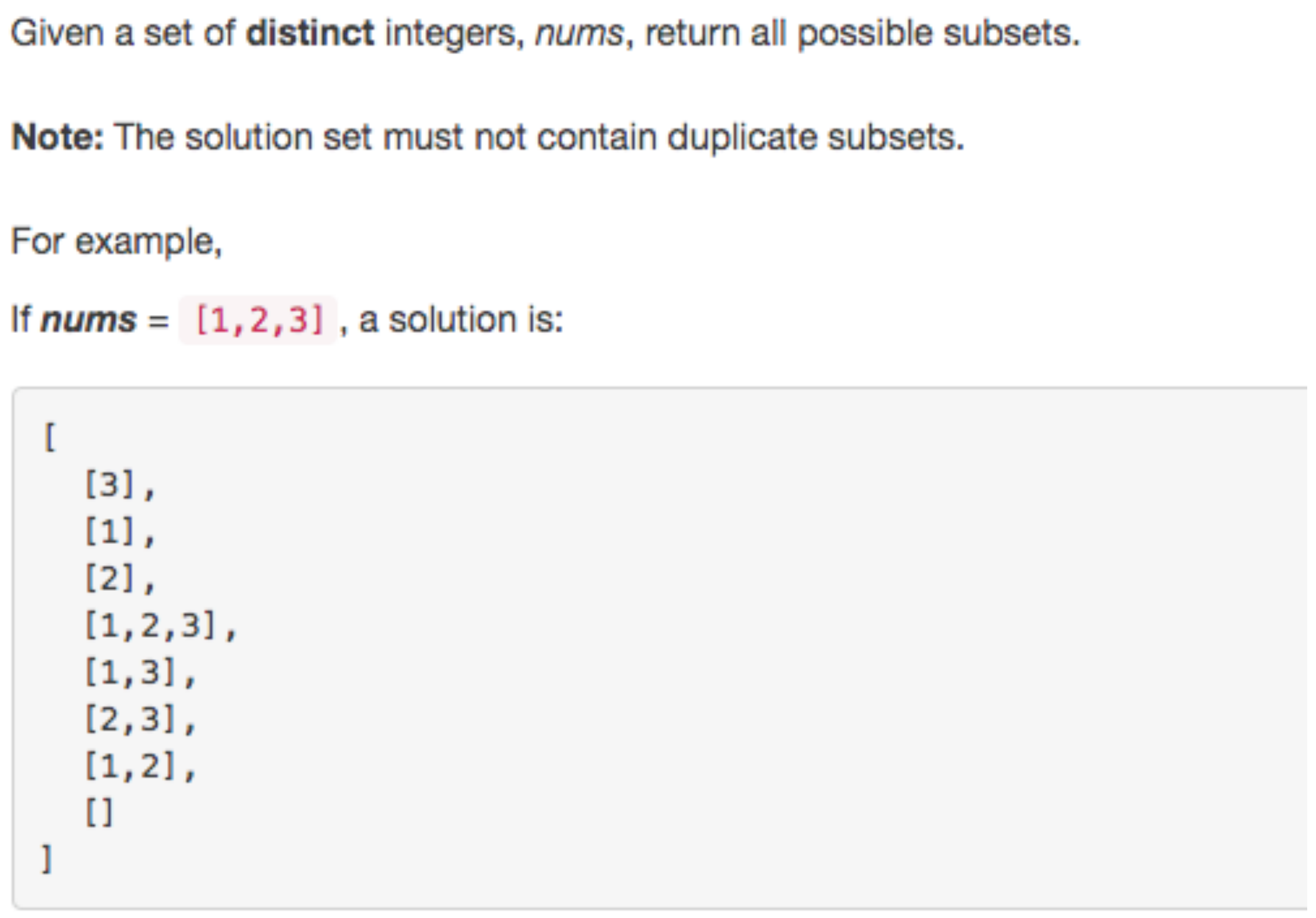


```java
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        if(nums == null || nums.length == 0){
            return res;
        }
        Arrays.sort(nums);
        helper(res, new ArrayList<Integer>(), nums, new boolean[nums.length]);
        return res;
    }

    public void helper(List<List<Integer>> res, List<Integer> path, int[] nums, boolean[] isVisited){
        if(path.size() == nums.length){
            res.add(new ArrayList<Integer>(path));
            return;
        }
        for(int i = 0; i < nums.length; i++){
            if(isVisited[i]){
                continue;
            }
            path.add(nums[i]);
            isVisited[i] = true;
            helper(res, path, nums, isVisited);
            path.remove(path.size() - 1);
            isVisited[i] = false;
        }
        return;
    }
}
```

Why ???

Why ???

78. Subsets



Given a set of **distinct** integers, *nums*, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

For example,

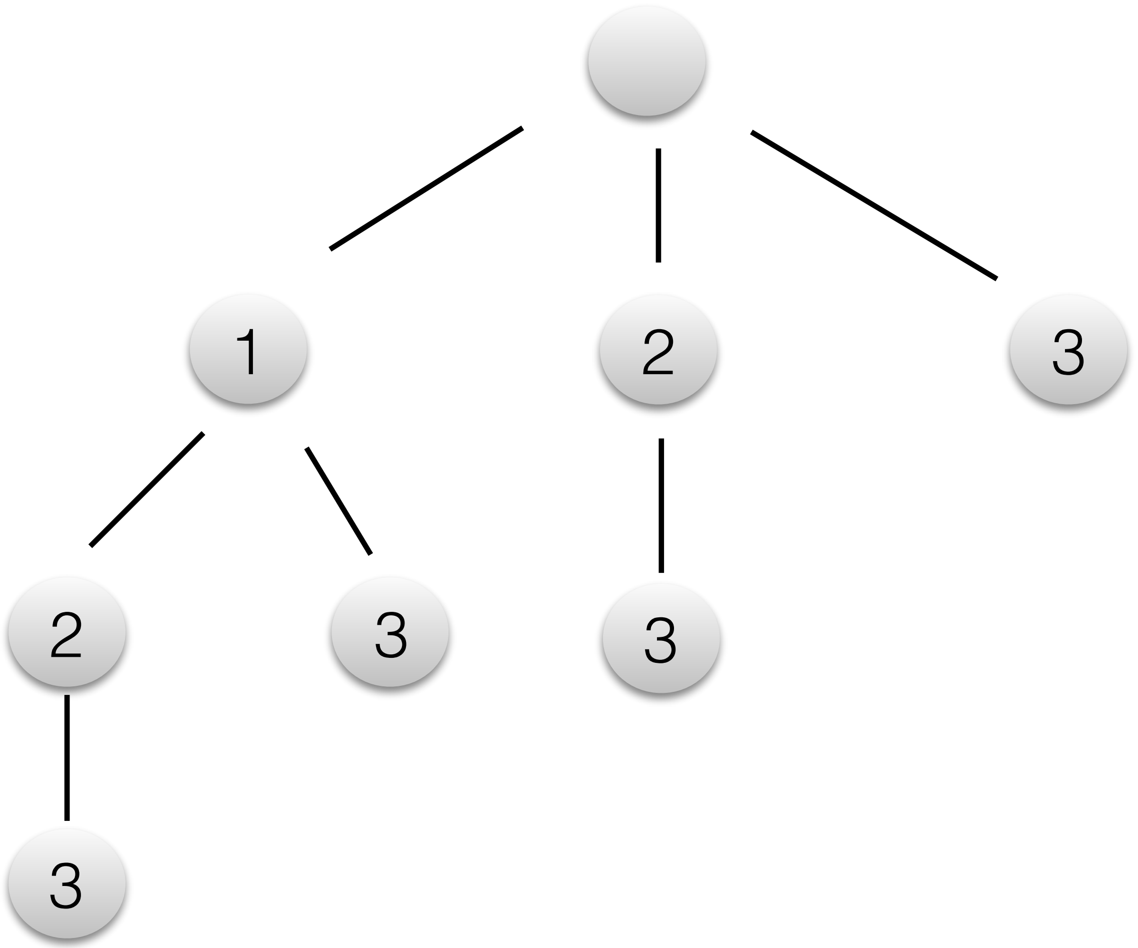
If ***nums*** = `[1,2,3]`, a solution is:

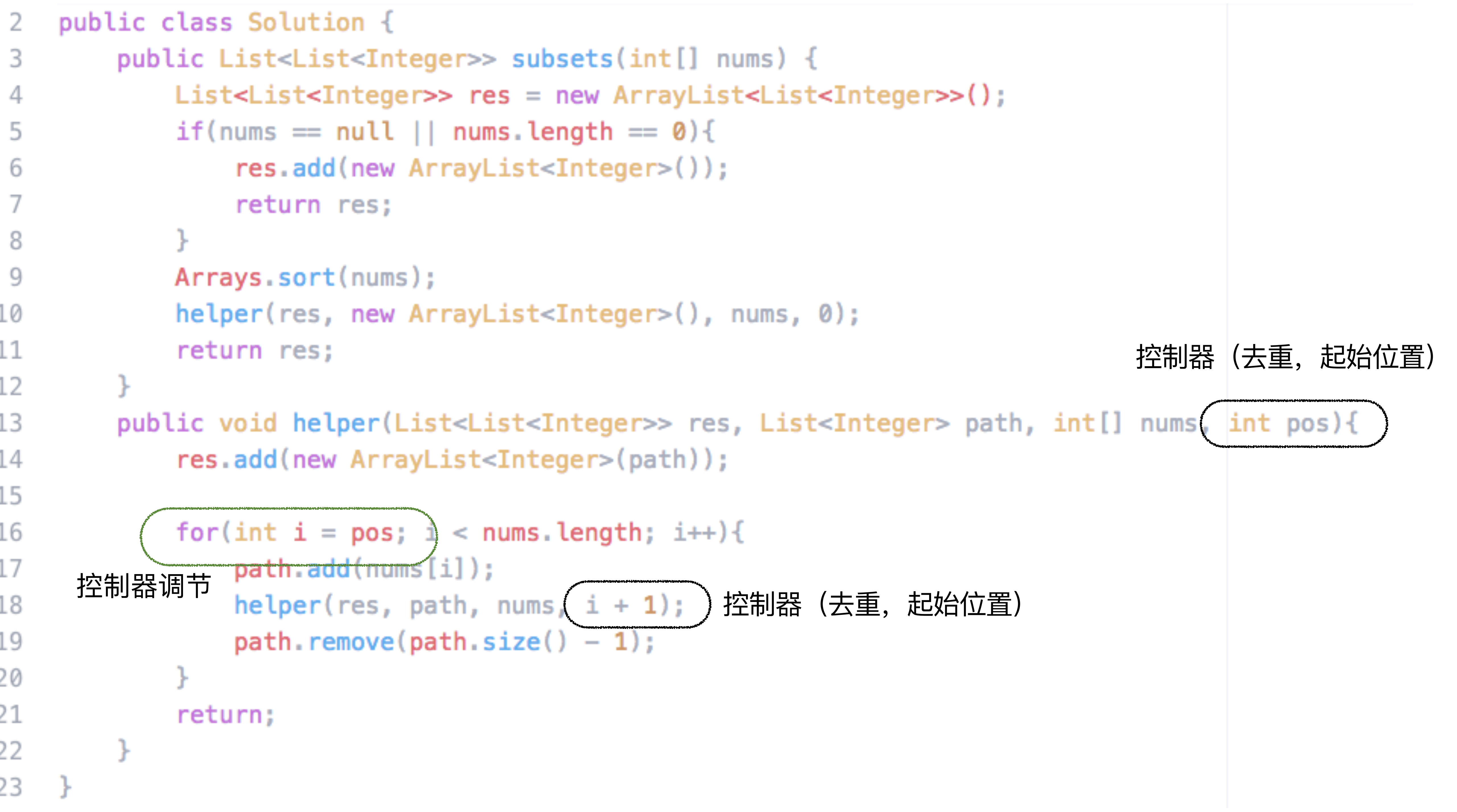
```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

10 min

```
public List<List<Integer>> subsets(int[] nums) {

}
```

Recursion Tree



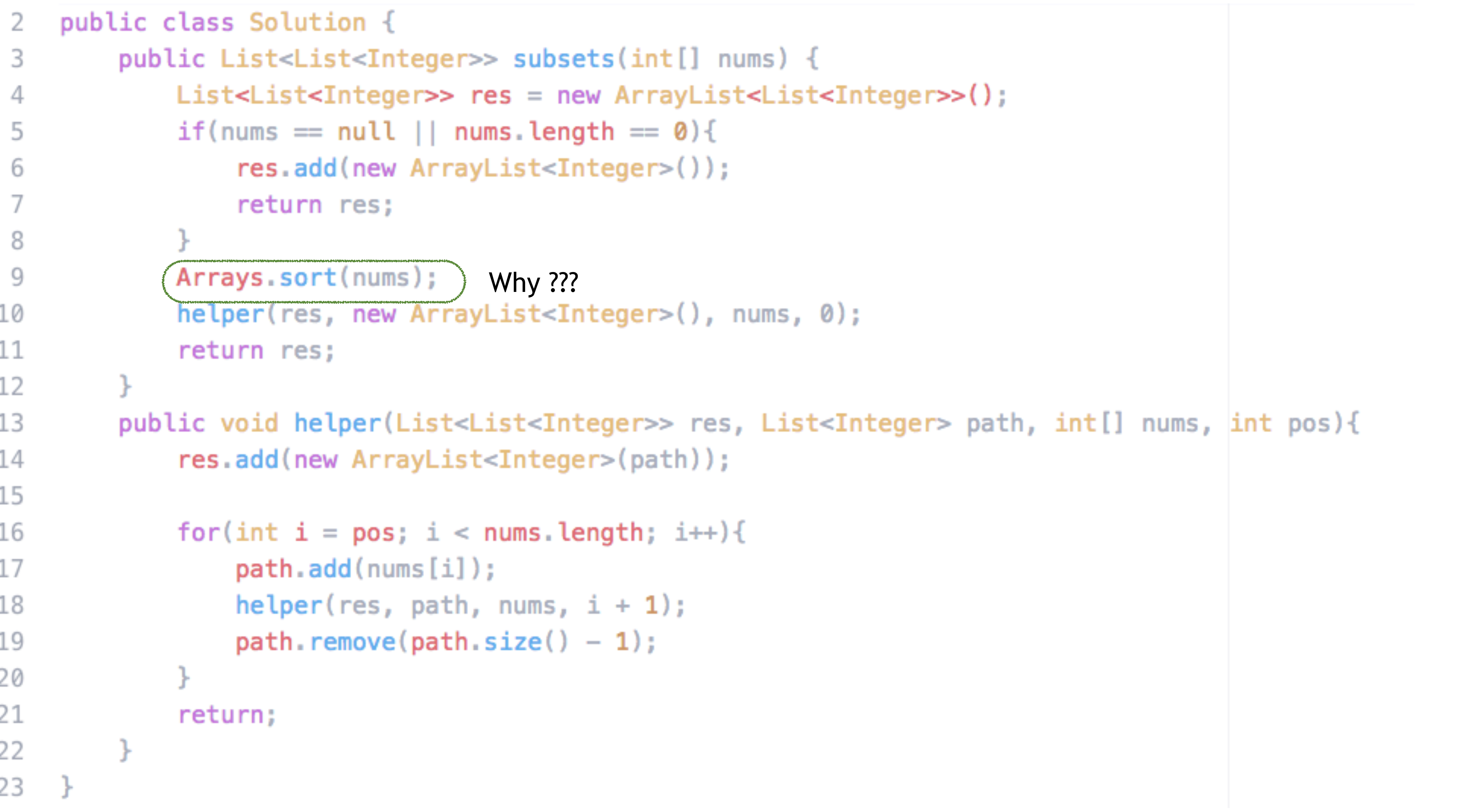


```java
public class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        if(nums == null || nums.length == 0){
            res.add(new ArrayList<Integer>());
            return res;
        }
        Arrays.sort(nums);
        helper(res, new ArrayList<Integer>(), nums, 0);
        return res;
    }
    public void helper(List<List<Integer>> res, List<Integer> path, int[] nums, int pos){
        res.add(new ArrayList<Integer>(path));

        for(int i = pos; i < nums.length; i++){
            path.add(nums[i]);
            helper(res, path, nums, i + 1);
            path.remove(path.size() - 1);
        }
        return;
    }
}
```

控制器（去重，起始位置）

控制器调节

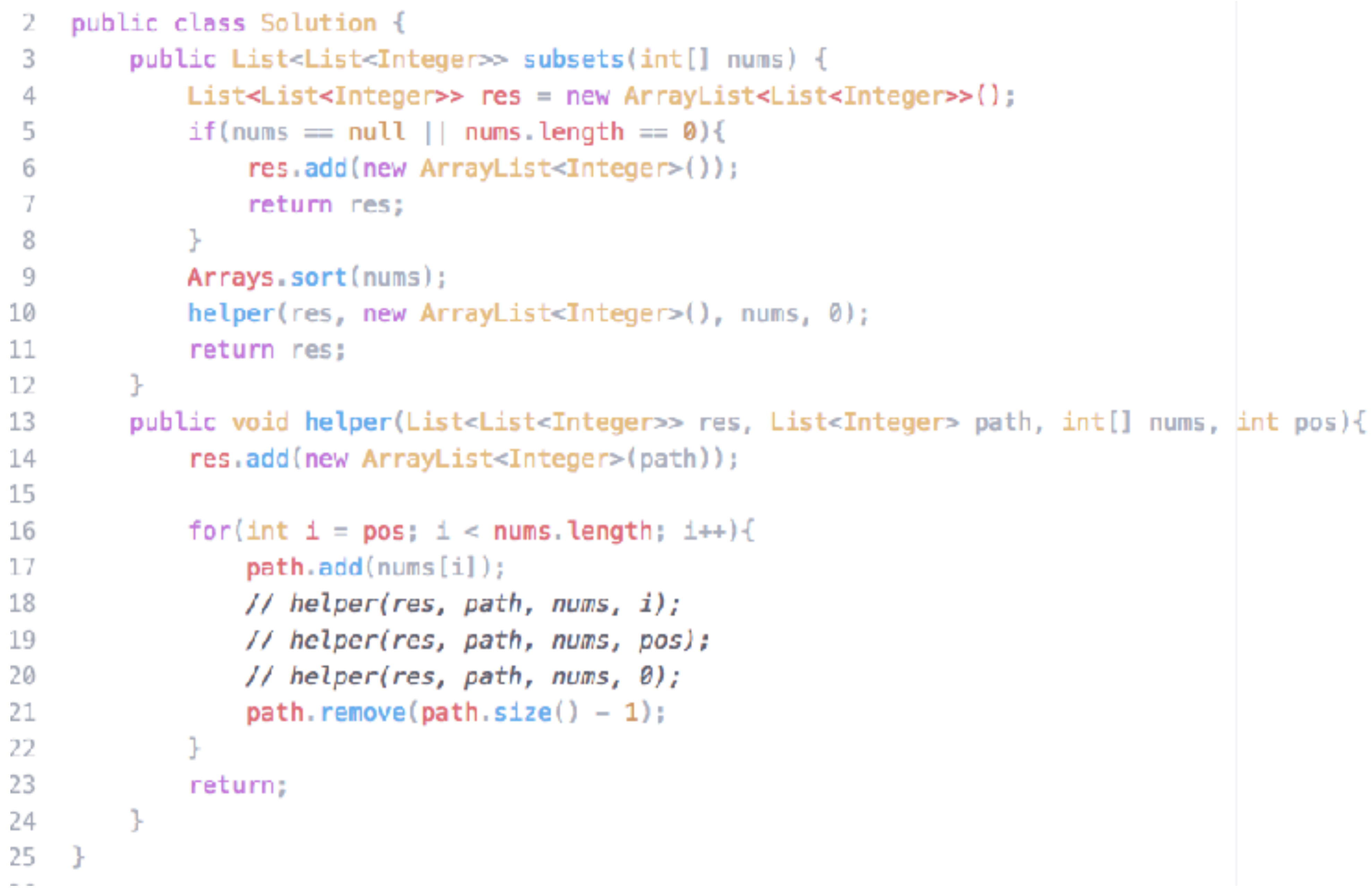
控制器（去重，起始位置）



```java
public class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        if(nums == null || nums.length == 0){
            res.add(new ArrayList<Integer>());
            return res;
        }
        Arrays.sort(nums);
        helper(res, new ArrayList<Integer>(), nums, 0);
        return res;
    }
    public void helper(List<List<Integer>> res, List<Integer> path, int[] nums, int pos){
        res.add(new ArrayList<Integer>(path));

        for(int i = pos; i < nums.length; i++){
            path.add(nums[i]);
            helper(res, path, nums, i + 1);
            path.remove(path.size() - 1);
        }
        return;
    }
}
```

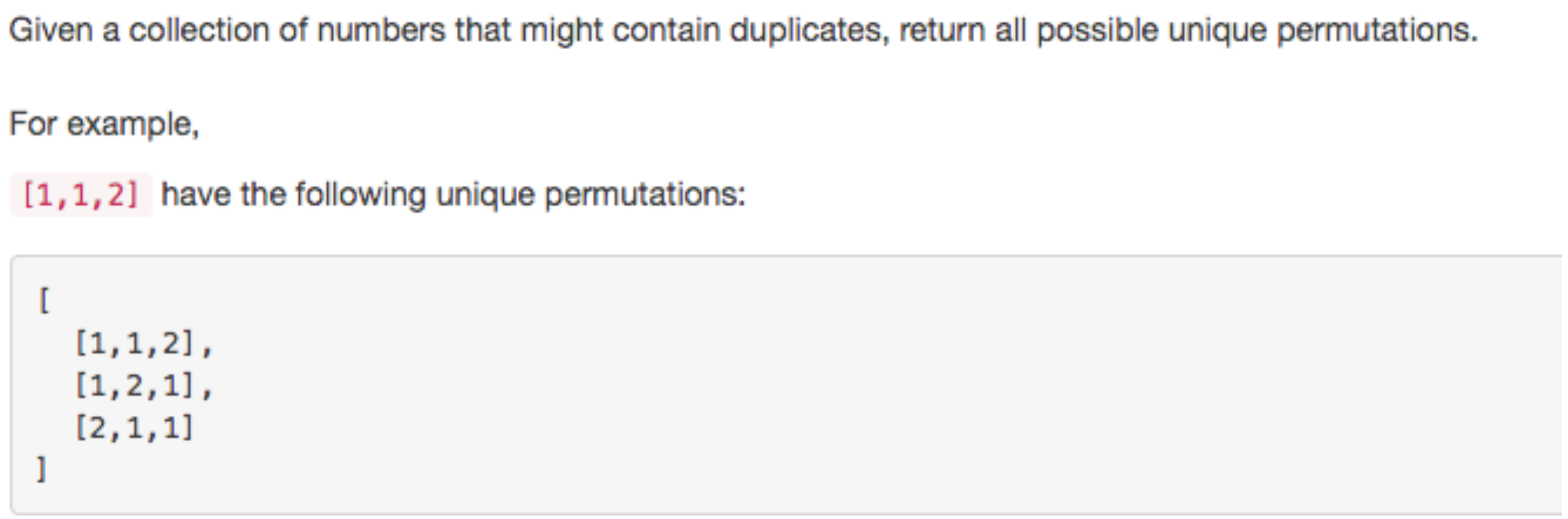
Why ???

从代码逆推 Recursion Tree



```java
public class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        if(nums == null || nums.length == 0){
            res.add(new ArrayList<Integer>());
            return res;
        }
        Arrays.sort(nums);
        helper(res, new ArrayList<Integer>(), nums, 0);
        return res;
    }
    public void helper(List<List<Integer>> res, List<Integer> path, int[] nums, int pos){
        res.add(new ArrayList<Integer>(path));

        for(int i = pos; i < nums.length; i++){
            path.add(nums[i]);
            // helper(res, path, nums, i);
            // helper(res, path, nums, pos);
            // helper(res, path, nums, 0);
            path.remove(path.size() - 1);
        }
        return;
    }
}
```

47. Permutations II



Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example,

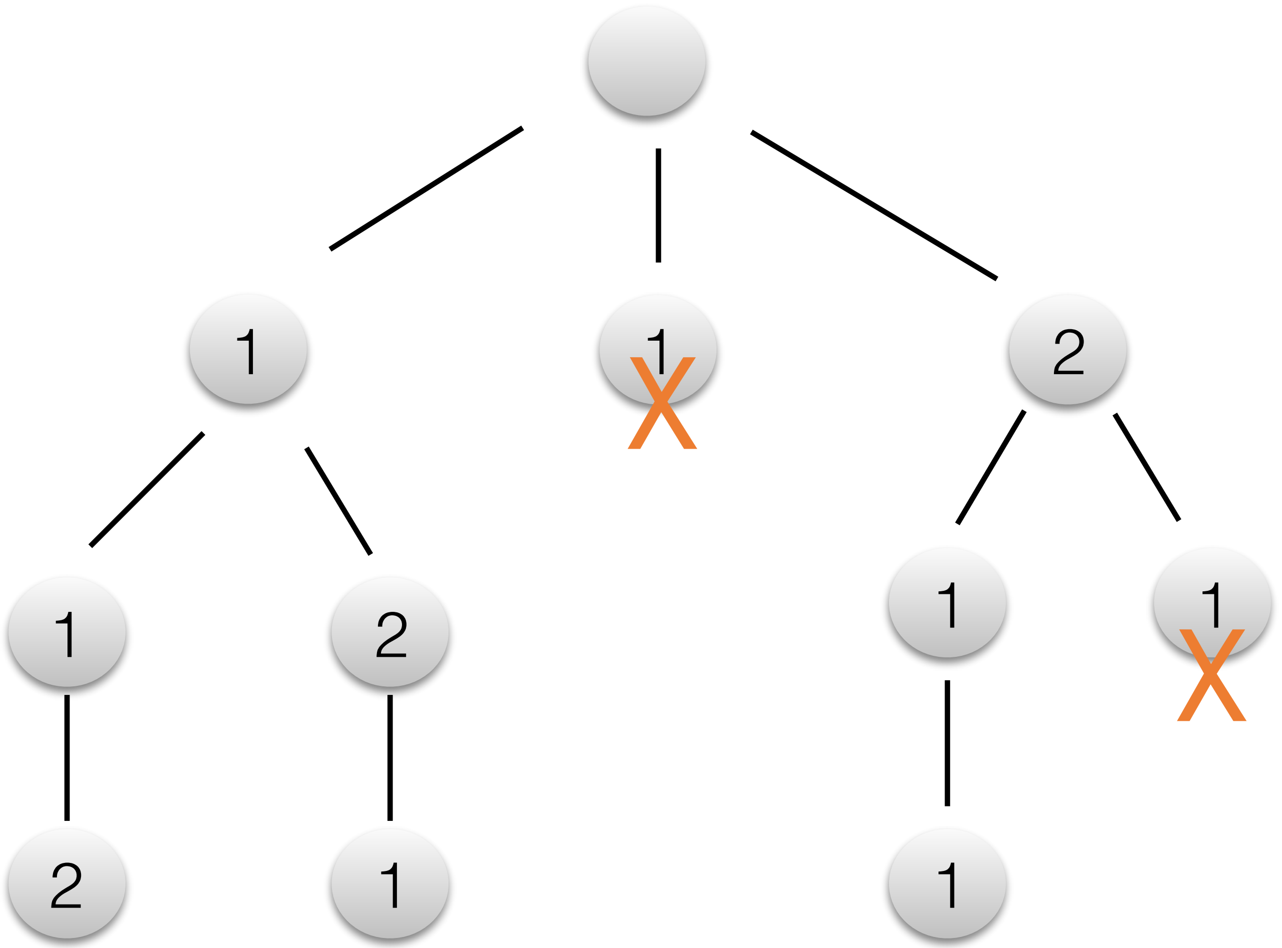
`[1,1,2]` have the following unique permutations:

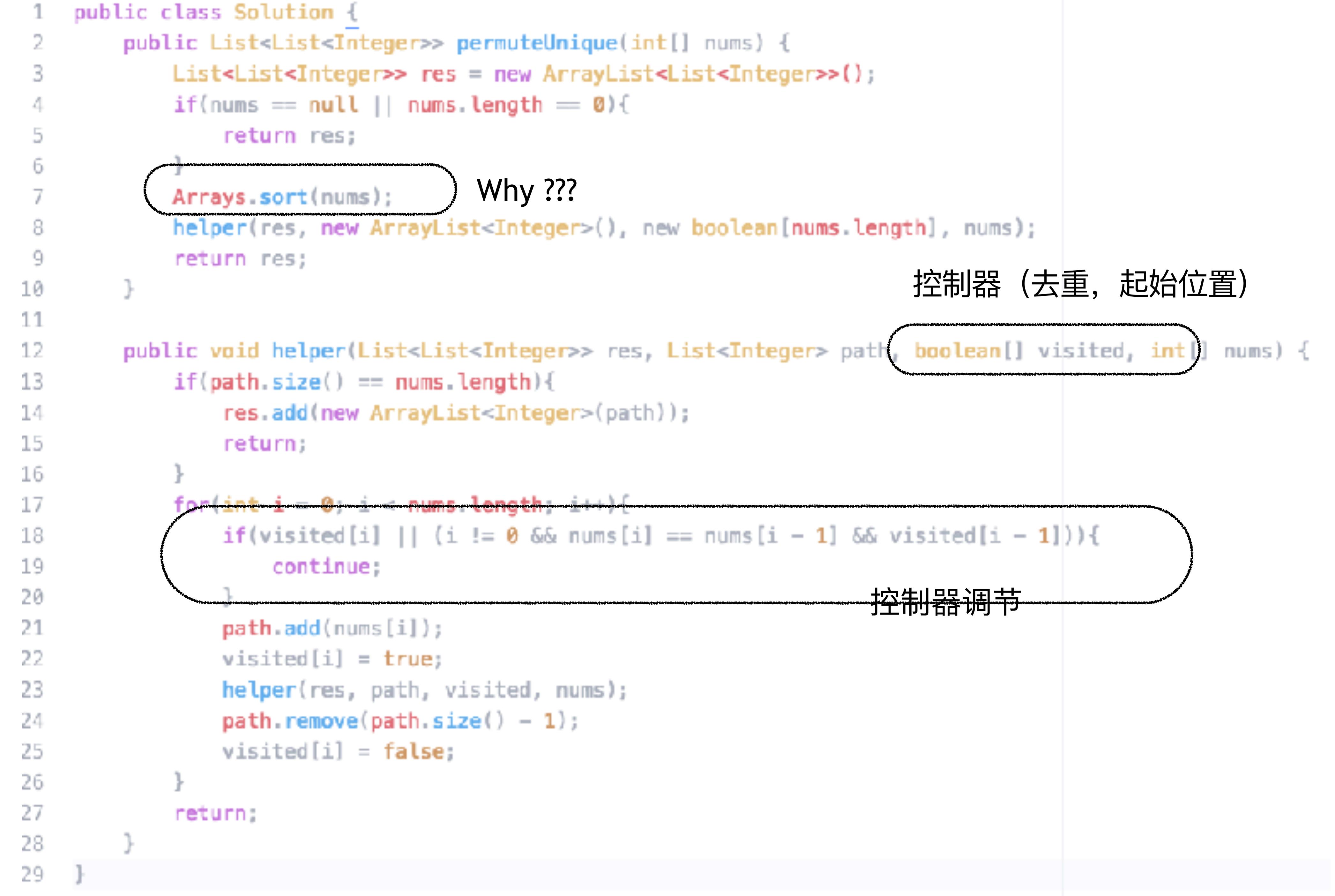
```
[
  [1,1,2],
  [1,2,1],
  [2,1,1]
]
```

10 min

```
public List<List<Integer>> permuteUnique(int[] nums) {

}
```

Recursion Tree





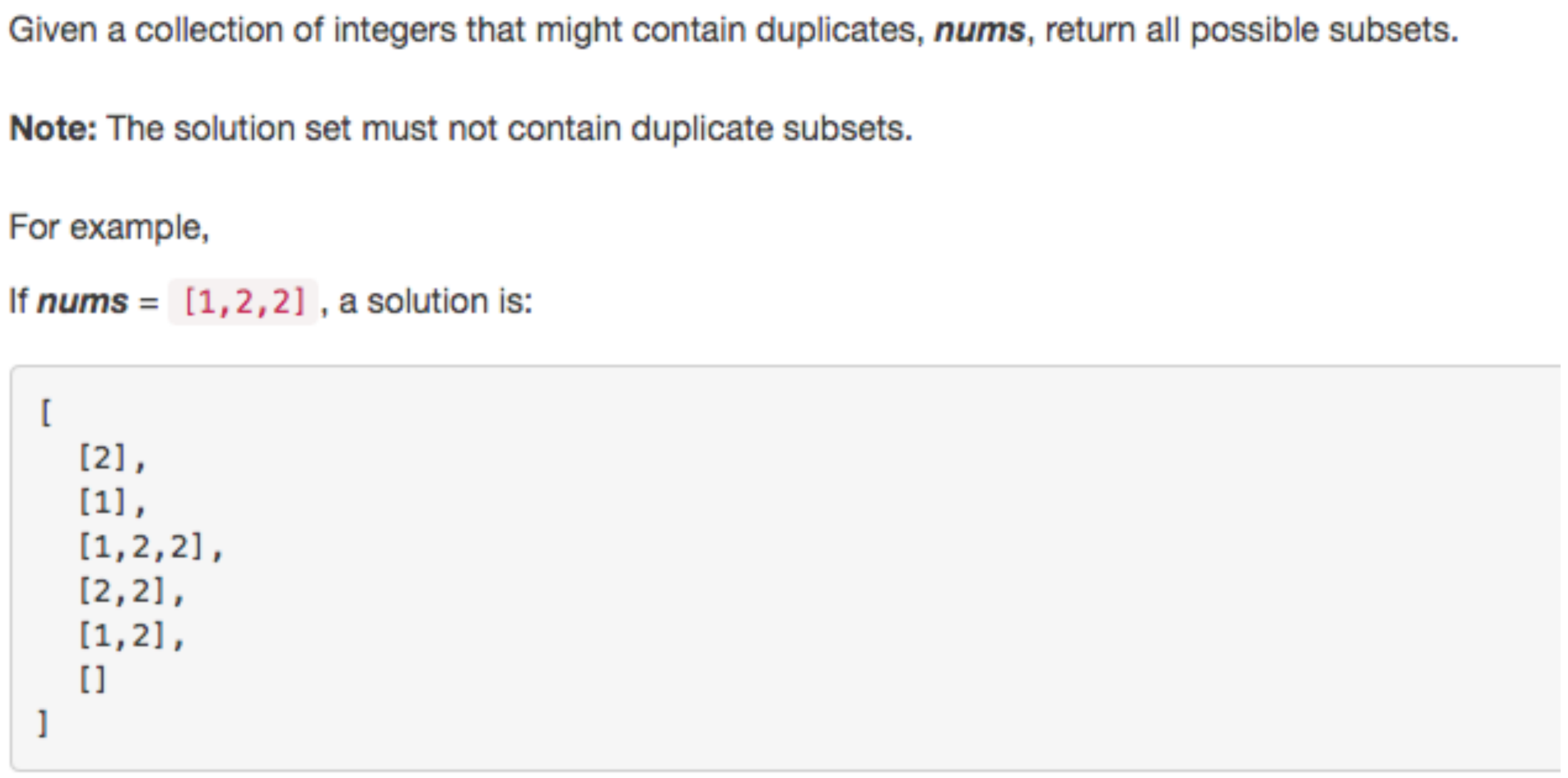
```java
public class Solution {
    public List<List<Integer>> permuteUnique(int[] nums) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        if(nums == null || nums.length == 0){
            return res;
        }
        Arrays.sort(nums);
        helper(res, new ArrayList<Integer>(), new boolean[nums.length], nums);
        return res;
    }

    public void helper(List<List<Integer>> res, List<Integer> path, boolean[] visited, int[] nums) {
        if(path.size() == nums.length){
            res.add(new ArrayList<Integer>(path));
            return;
        }
        for(int i = 0; i < nums.length; i++){
            if(visited[i] || (i != 0 && nums[i] == nums[i - 1] && visited[i - 1])){
                continue;
            }
            path.add(nums[i]);
            visited[i] = true;
            helper(res, path, visited, nums);
            path.remove(path.size() - 1);
            visited[i] = false;
        }
        return;
    }
}
```

Why ???

控制器（去重，起始位置）

控制器调节

90. Subsets II



Given a collection of integers that might contain duplicates, ***nums***, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

For example,

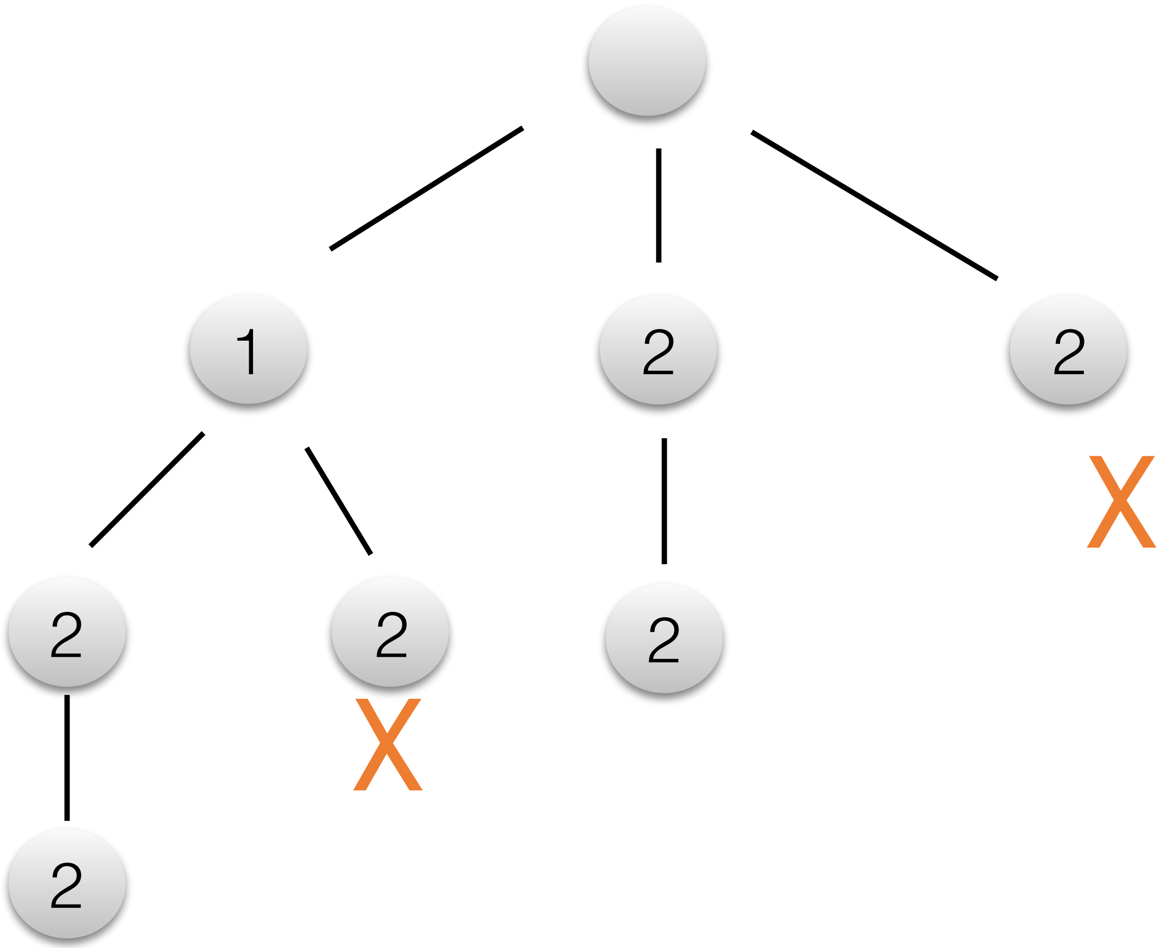
If ***nums*** = `[1,2,2]`, a solution is:

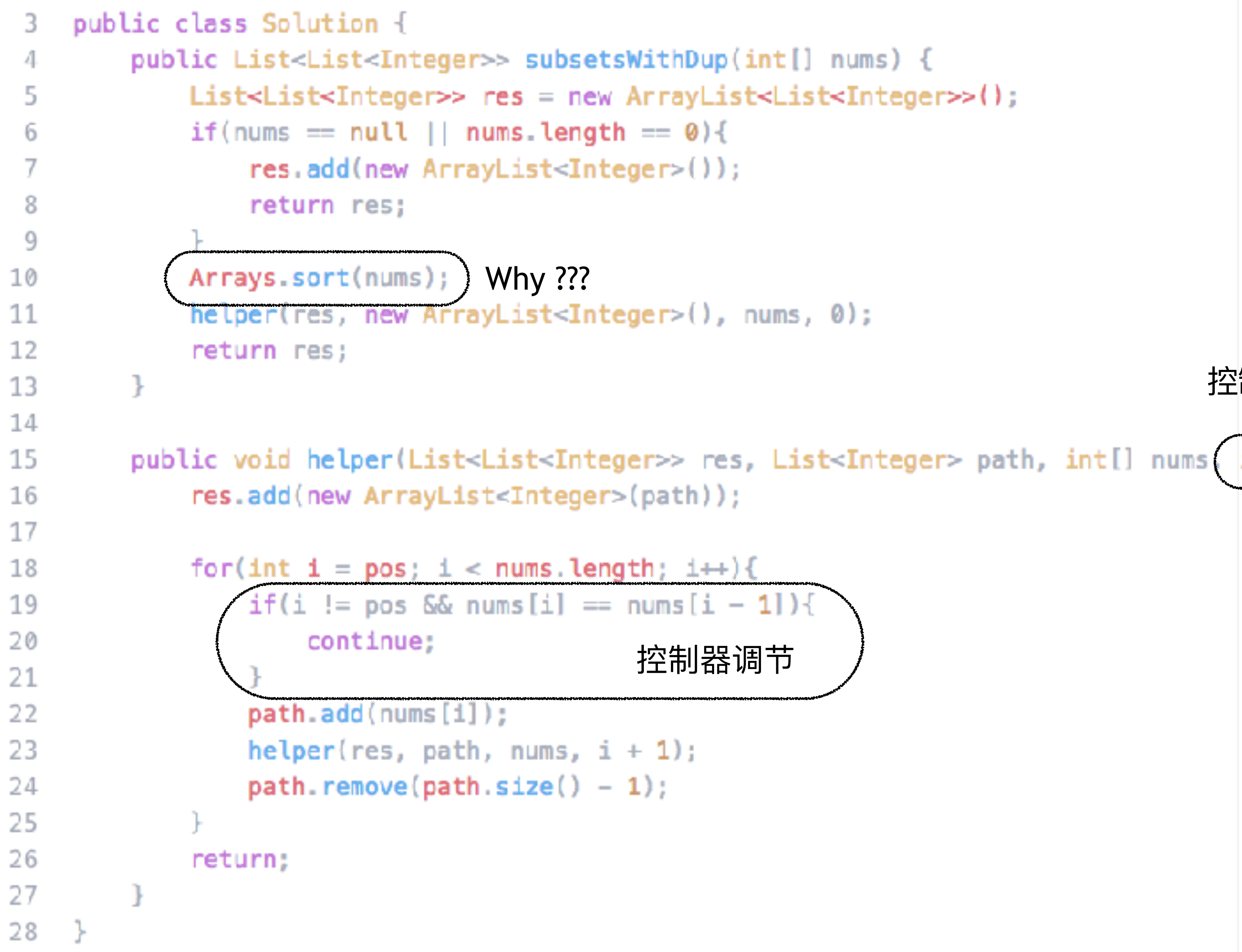
```
[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
```

10 min

```
public List<List<Integer>> subsetsWithDup(int[] nums) {

}
```

Recursion Tree





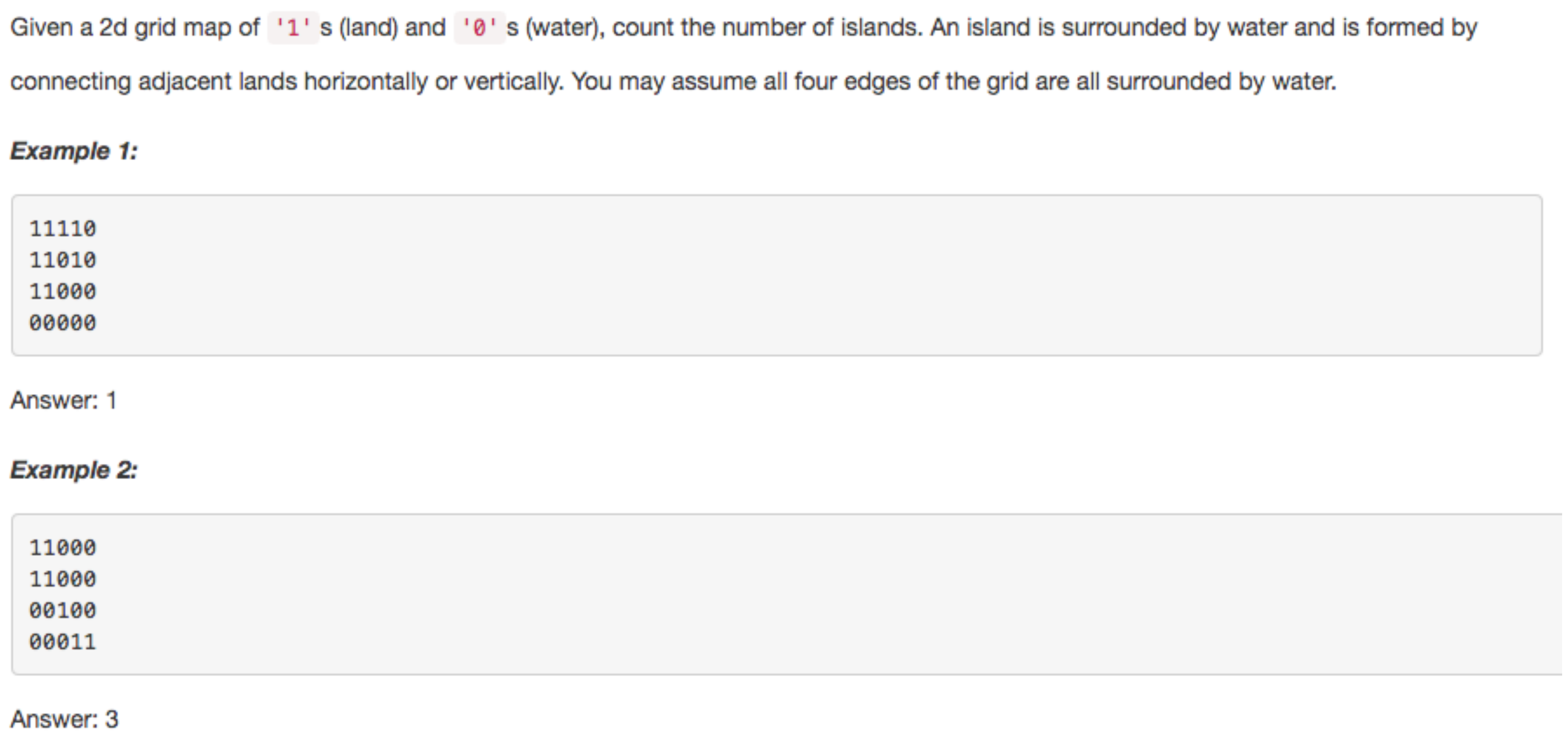
```java
public class Solution {
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        if(nums == null || nums.length == 0){
            res.add(new ArrayList<Integer>());
            return res;
        }
        Arrays.sort(nums);
        helper(res, new ArrayList<Integer>(), nums, 0);
        return res;
    }

    public void helper(List<List<Integer>> res, List<Integer> path, int[] nums, int pos){
        res.add(new ArrayList<Integer>(path));

        for(int i = pos; i < nums.length; i++){
            if(i != pos && nums[i] == nums[i - 1]){
                continue;
            }
            path.add(nums[i]);
            helper(res, path, nums, i + 1);
            path.remove(path.size() - 1);
        }
        return;
    }
}
```

Why ???

控制器（去重，起始位置）

控制器调节

200. Number of Islands



Given a 2d grid map of `'1'`s (land) and `'0'`s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
11110
11010
11000
00000
```

Answer: 1

Example 2:

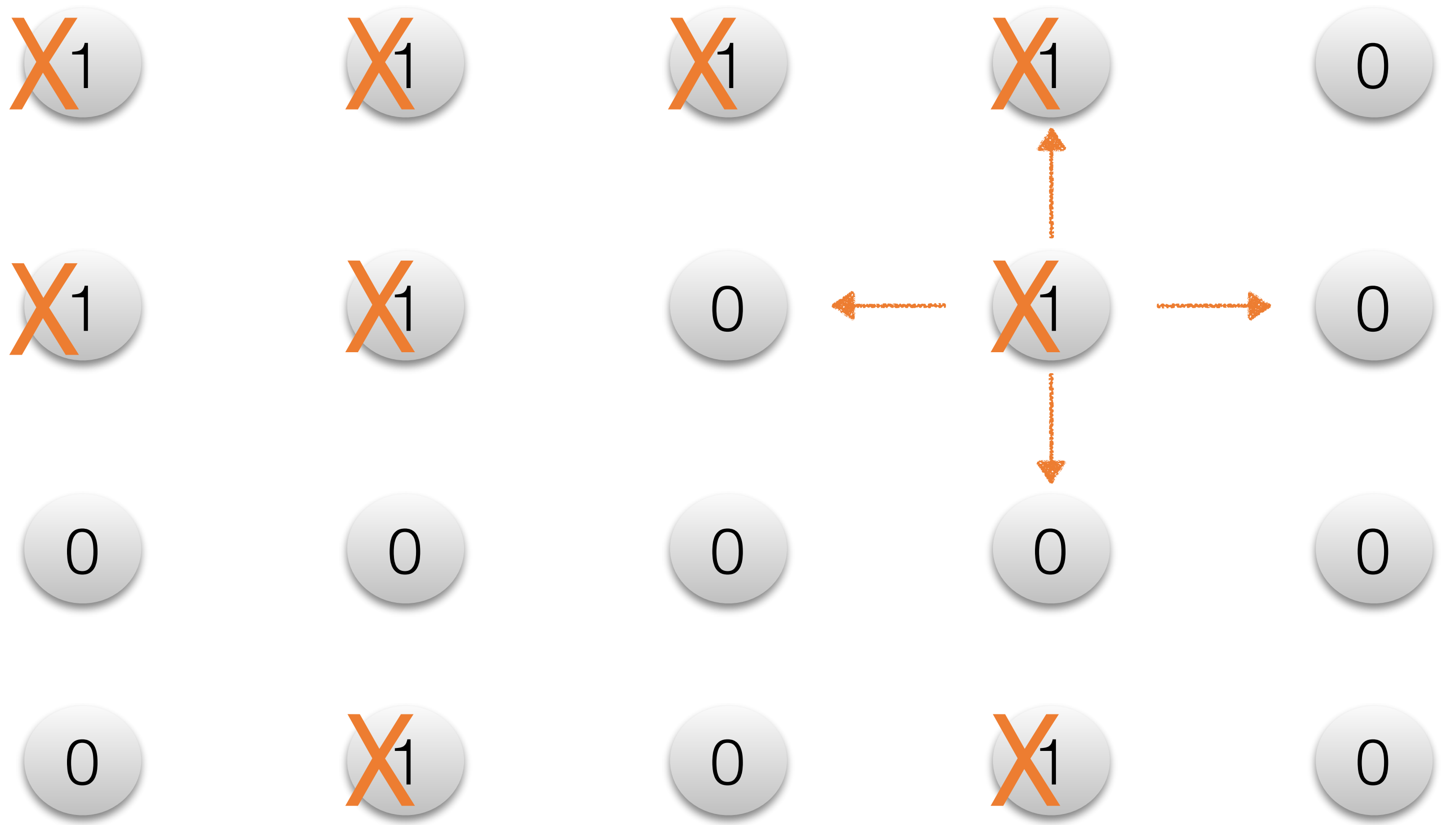
```
11000
11000
00100
00011
```

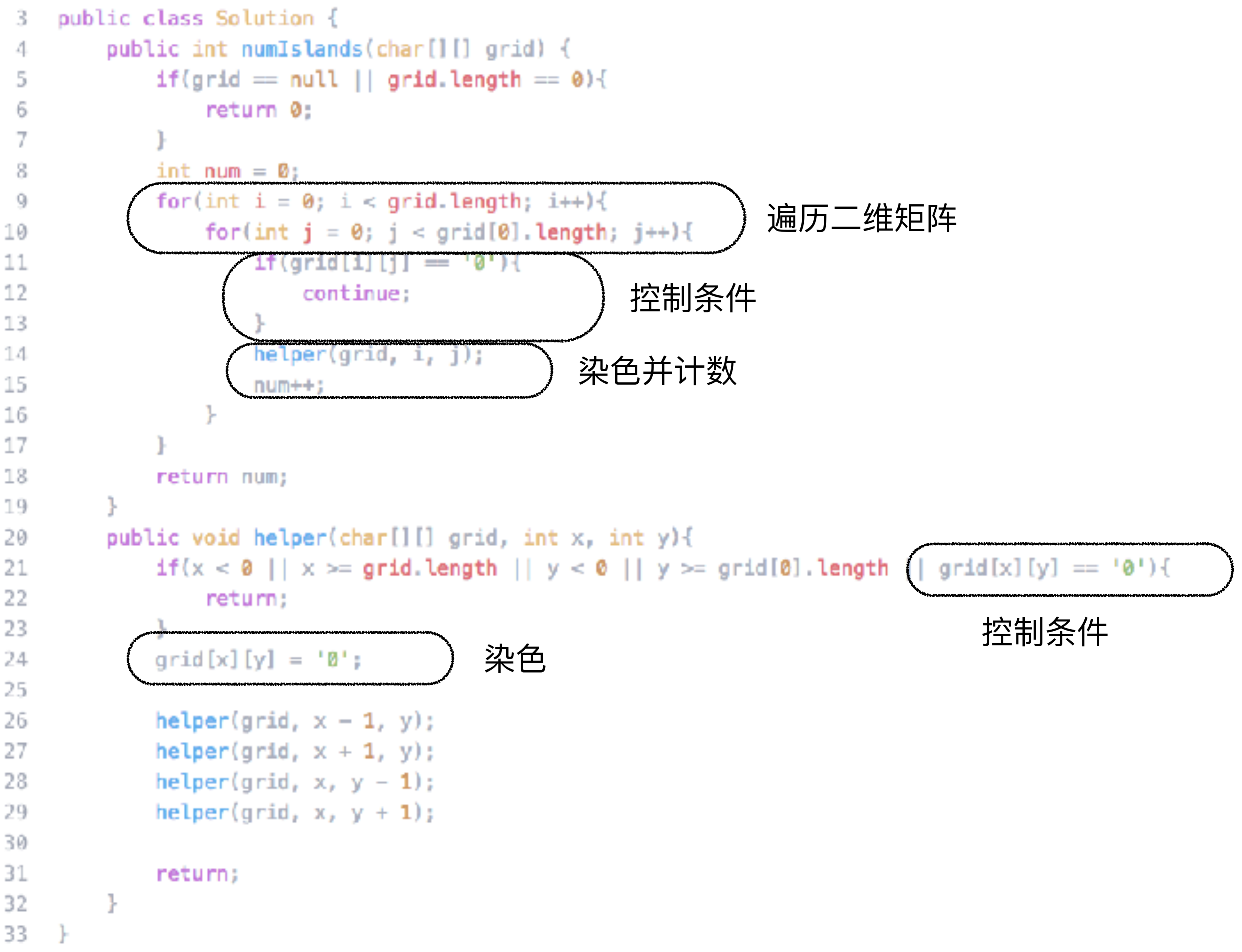
Answer: 3

10 min

```
public int numIslands(char[][] grid) {

}
```

DFS 染色





```java
public class Solution {
    public int numIslands(char[][] grid) {
        if(grid == null || grid.length == 0){
            return 0;
        }
        int num = 0;
        for(int i = 0; i < grid.length; i++){
            for(int j = 0; j < grid[0].length; j++){
                if(grid[i][j] == '0'){
                    continue;
                }
                helper(grid, i, j);
                num++;
            }
        }
        return num;
    }
    public void helper(char[][] grid, int x, int y){
        if(x < 0 || x >= grid.length || y < 0 || y >= grid[0].length || grid[x][y] == '0'){
            return;
        }
        grid[x][y] = '0';

        helper(grid, x - 1, y);
        helper(grid, x + 1, y);
        helper(grid, x, y - 1);
        helper(grid, x, y + 1);

        return;
    }
}
```

遍历二维矩阵

控制条件

染色并计数

控制条件

染色

Homework

BFS	***DFS***	***SORT***
103. Binary Tree Zigzag Level Order Traversal	*329. Longest Increasing Path in a Matrix*	*148 Sort List*
199. Binary Tree Right Side View	*394. Decode String*	*Quick Sort*
	199. Binary Tree Right Side View	*Merge Sort*
	542. 01 Matrix	*Insertion Sort*
		Selection Sort

Q & A

Thank you