Backtracking Assignment Solutions (Java, Recursion Only)

Problem 1: Move Brackets (Codeforces 1374C)

```
import java.util.*; public class MoveBrackets { static int solve(String s, int i, int open, int moves) { if (i == s.length()) return moves; if (s.charAt(i) == '(') return solve(s, i + 1, open + 1, moves); else { if (open > 0) return solve(s, i + 1, open - 1, moves); else return solve(s, i + 1, open, moves + 1); } } public static void main(String[] args) { Scanner sc = new Scanner(System.in); int t = sc.nextInt(); while (t-- > 0) { String s = sc.next(); System.out.println(solve(s, 0, 0, 0)); } } }
```

Problem 2: Palindrome Partitioning (LeetCode 131)

```
import java.util.*; public class PalindromePartitioning { static boolean isPalindrome(String s, int l, int r) { if (l >= r) return true; if (s.charAt(l) != s.charAt(r)) return false; return isPalindrome(s, l + 1, r - 1); } static void backtrack(String s, int start, List current, List> res) { if (start == s.length()) { res.add(new ArrayList<>(current)); return; } for (int end = start; end < s.length(); end++) { if (isPalindrome(s, start, end)) { current.add(s.substring(start, end + 1)); backtrack(s, end + 1, current, res); current.remove(current.size() - 1); } } } public static void main(String[] args) { Scanner sc = new Scanner(System.in); String s = sc.next(); List> res = new ArrayList<>(); backtrack(s, 0, new ArrayList<>(), res); for (List list : res) System.out.println(list); } }
```

Problem 3: N-Queens II (LeetCode 52)

```
public class NQueensII { static int solve(int n, int row, boolean[] cols, boolean[] d1, boolean[] d2) { if (row == n) return 1; int count = 0; for (int col = 0; col < n; col++) { int id1 = col - row + n, id2 = col + row; if (cols[col] || d1[id1] || d2[id2]) continue; cols[col] = d1[id1] = d2[id2] = true; count += solve(n, row + 1, cols, d1, d2); cols[col] = d1[id1] = d2[id2] = false; } return count; } public static void main(String[] args) { java.util.Scanner sc = new java.util.Scanner(System.in); int n = sc.nextInt(); System.out.println(solve(n, 0, new boolean[n], new boolean[2 * n], new boolean[2 * n])); } }
```

Problem 4: Permutations II (LeetCode 47)

```
import java.util.*; public class PermutationsII { static void backtrack(int[] nums, boolean[] used, List curr, List> res) { if (curr.size() == nums.length) { res.add(new ArrayList<>(curr)); return; } for (int i = 0; i < nums.length; i++) { if (used[i] || (i > 0 && nums[i] == nums[i - 1] && !used[i - 1])) continue; used[i] = true; curr.add(nums[i]); backtrack(nums, used, curr, res); used[i] = false; curr.remove(curr.size() - 1); } } public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt(); int[] nums = new int[n]; for (int i = 0; i < n; i++) nums[i] = sc.nextInt(); Arrays.sort(nums); List> res = new ArrayList<>(); backtrack(nums, new boolean[n], new ArrayList<>(), res); for (List l : res) System.out.println(l); } }
```