9.1 Letter Combination of a Phone Number (Backtracking)

```
import java.util.*;
class LetterCombinations {
   private static final Map<Character, String> phoneMap = new HashMap<>();

   static {
      phoneMap.put('2', "abc");
      phoneMap.put('3', "def");
      phoneMap.put('4', "ghi");
      phoneMap.put('5', "jkl");
      phoneMap.put('6', "mno");
      phoneMap.put('7', "pqrs");
      phoneMap.put('8', "tuv");
      phoneMap.put('9', "wxyz");
   }
   public List<String> letterCombinations(String digits) {
      List<String> result = new ArrayList<>();
      if (digits == null || digits.length() == 0) return result;
      backtrack(result, new StringBuilder(), digits, 0);
      return result;
   }
   private void backtrack(List<String> result, StringBuilder current, String digits, int
index) {
      if (index == digits.length()) {
         result.add(current.toString());
         return;
      }
      String letters = phoneMap.get(digits.charAt(index));
      for (char letter : letters.toCharArray()) {
         current.append(letter);
         backtrack(result, current, digits, index + 1);
         current.deleteCharAt(current.length() - 1); // Backtrack
      }
   }
   public static void main(String[] args) {
      LetterCombinations obj = new LetterCombinations();
      String digits = "23";
      List<String> combinations = obj.letterCombinations(digits);
      System.out.println("Letter combinations for " + digits + ": " + combinations);
   }
}
```

9.2 Regular Expression Matching using Backtracking

Coding:

```
class RegexMatching {
    public boolean isMatch(String s, String p) {
        return matchHelper(s, p, 0, 0);
    }
    private boolean matchHelper(String s, String p, int i, int j) {
        if (j == p.length()) return i == s.length();
        boolean firstMatch = (i < s.length() &&
                              (s.charAt(i) == p.charAt(j) || p.charAt(j) == '.'));
        if (j + 1 < p.length() && p.charAt(j + 1) == '*') {
            return matchHelper(s, p, i, j + 2) || (firstMatch && matchHelper(s, p, i + 1, j));
        } else {
            return firstMatch && matchHelper(s, p, i + 1, j + 1);
        }
    }
    public static void main(String[] args) {
        RegexMatching regex = new RegexMatching();

        System.out.println(regex.isMatch("aa", "a*"));  // Output: true
        System.out.println(regex.isMatch("mississippi", "mis*is*p*."));  // Output: false
        System.out.println(regex.isMatch("ab", ".*"));  // Output: true
        System.out.println(regex.isMatch("aab", "c*a*b"));  // Output: true
    }
}
```

9.3 Sequential Digits Using Backtracking

Coding:

```
import java.util.*;
class SequentialDigits {
    public List<Integer> sequentialDigits(int low, int high) {
        List<Integer> result = new ArrayList<>();
        for (int start = 1; start <= 9; start++) {
            generateNumbers(start, 0, low, high, result);
        }
        Collections.sort(result);
        return result;
    }

    private void generateNumbers(int digit, int num, int low, int high, List<Integer> result)
{
        if (num >= low && num <= high) {
            result.add(num);
        }
        if (num > high || digit > 9) {
            return;
        }
        generateNumbers(digit + 1, num * 10 + digit, low, high, result);
    }
    public static void main(String[] args) {
        SequentialDigits obj = new SequentialDigits();
        int low = 100, high = 300;
        System.out.println("Sequential digits in range: " + obj.sequentialDigits(low, high));
    }
}
```