Ashley Taylor and Shreya Shankar (Based on handouts by previous CS106B instructors and TAs)

Recursion and Recursive Backtracking: Solutions

1. Digit Sum (Recursion)

```
int digitSum(int n) {
    if (n < 0) {
        return -digitSum(-n);
    } else if (n < 10) {
        return n;
    } else {
        return n % 10 + digitSum(n / 10);
    }
}
```

2. Edit Distance (Exhaustive Search)

```
int editDistance(const string& s1, const string& s2) {
    if (s1 == "") {
        return s2.length();
    } else if (s2 == "") {
        return s1.length();
    }t
    // try three possibilities for the "zeroth" character:
    int add = 1 + editDistance(s1, s2.substr(1, s2.length()));
    int del = 1 + editDistance(s1.substr(1, s1.length()), s2);
    int sub = editDistance(s1.substr(1, s1.length()), s2.substr(1, s2.length()));
    if (s1[0] != s2[0]) {
        sub += 1;
    }
    return min(add, min(del, sub));
}
```

3. List Twiddles (Backtracking)

```
void listTwiddles(const string& str, const Lexicon& lex) {
    listTwiddlesHelper("", str, /* index */ 0, lex);
}

void listTwiddlesHelper(const string& prefix, const string& str, int index,
```

```
        const Lexicon& lex) {
    if (!lex.containsPrefix(prefix)) return;
    if (index >= str.size()) {
        if (lex.contains(prefix)) cout << prefix << endl;
        return;
    }
    for (char ch = str[index] - 2; ch <= str[index] + 2; ch++) {
        if (isalpha(ch)) {
            listTwiddlesHelper(prefix + ch, str, index + 1, lex);
        }
    }
}
```

4. Longest Common Subsequence (Backtracking)

```
string longestCommonSubsequence(string s1, string s2) {
    if (s1.length() == 0 || s2.length() == 0) {
        return "";
    } else if (s1[0] == s2[0]) {
        return s1[0] + longestCommonSubsequence(s1.substr(1), s2.substr(1));
    } else {
        string choice1 = longestCommonSubsequence(s1, s2.substr(1));
        string choice2 = longestCommonSubsequence(s1.substr(1), s2);
        if (choice1.length() >= choice2.length()) {
            return choice1;
        } else {
            return choice2;
        }
    }
}
```

5. Spellable (Backtracking)

```
bool isElementSpellable(string text, Lexicon& symbols) {
    if (text == "") {
        return true;
    } else {
        for (int i = 1; i <= text.length() && i <= 3; i++) {
            if (symbols.contains(text.substr(0, i)) &&
                isElementSpellable(text.substr(i), symbols)) {
                return true;
            }
        }

        return false;
    }
}
```