Week 4 Section

1. Big Oh
   a) O(N)
   b) O(N²)
   c) O(1)

2. diceRolls

   // private recursive helper to implement diceRolls logic
   void diceRollsHelper(int dice, Vector<int>& chosen) {
     if (dice <= 0) {
       cout << chosen << endl; // base case
     } else {
       for (int i = 1; i <= 6; i++) {
         chosen.add(i); // choose
         diceRollsHelper(dice - 1, chosen); // explore
         chosen.remove(chosen.size() - 1); // un-choose
       }
     }
   }
   
   // Prints all possible outcomes of rolling the given
   // number of six-sided dice in {#, #, #} format.
   void diceRolls(int dice) {
     Vector<int> chosen;
     diceRollsHelper(dice, chosen);
   }

3. diceSum

   void diceSumHelper(int dice, int sum, int desiredSum, Vector<int>& chosen) {
     if (dice == 0) {
       if (sum == desiredSum) {
         cout << chosen << endl; // base case
       }
     } else if (sum + 1*dice <= desiredSum && sum + 6*dice >= desiredSum) {
       for (int i = 1; i <= 6; i++) {
         chosen.add(i); // choose
         diceSumHelper(dice - 1, sum + i, desiredSum, chosen); // explore
         chosen.remove(chosen.size() - 1); // un-choose
       }
     }
   }
   
   // main function on next page

Thanks to Marty Stepp and other CS106B and X instructors and TAs for contributing problems on this handout.
void diceSum(int dice, int desiredSum) {
    Vector<int> chosen;
    diceSumHelper(dice, 0, desiredSum, chosen);
}

4. largestSum

int largestSum(Vector<int>& numbers, int limit) {
    if (limit <= 0 || numbers.isEmpty()) {
        return 0;  // base case
    } else {
        // grab last number for evaluation (first OK too)
        int last = numbers[numbers.size() - 1];
        numbers.remove(numbers.size() - 1);
        int largest = largestSum(numbers, limit); // explore rest w/out last
        if (last <= limit) {
            // explore rest with last
            int withLast = last + largestSum(numbers, limit - last);
            largest = max(largest, withLast);
        }
        numbers.add(last);  // put first number back in
    }
    return largest;
}

5. longestCommonSubsequence

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;
        }
    }
}
6. makeChange

```cpp
void makeChangeHelper(int amount, Vector<int>& coins, Vector<int>& chosen) {
    if (coins.isEmpty()) {
        if (amount == 0) {
            cout << chosen << endl;
        } else {
            int coin = coins[0];
            coins.remove(0);
            for (int i = 0; i <= (amount / coin); i++) {
                chosen.add(i);
                makeChangeHelper(amount - (i * coin), coins, chosen);
                chosen.remove(chosen.size() - 1);
            }
            coins.insert(0, coin);
        }
    }
}

void makeChange(int amount, Vector<int>& coins) {
    Vector<int> chosen;
    makeChangeHelper(amount, coins, chosen);
}
```

7. canBalance

```cpp
bool canBalance(int target, Vector<int>& weights) {
    if (target == 0) {
        return true;
    } else if (weights.isEmpty()) {
        return false;
    } else {
        int weight = weights[0];
        weights.remove(0);
        bool answer = canBalance(target, weights) ||
        canBalance(target + weight, weights) ||
        canBalance(target - weight, weights);
        weights.insert(0, weight);
        return answer;
    }
}
```