Part 1 – Continuation-Passing Style

To save space I am using the name top for the top-level continuation (lambda (x) x)

1. Give a tail-recursive continuation-passing-style function (rember-k a lat k) that removes the first occurrence (only the first) of atom a from lat and then applies k to the result. So (rember-k 'b '(a b a b a b b) top) returns '(a a b a b b)

2. Give a tail-recursive continuation-passing style function (index-k a lat k) that returns the 0-based index of the first occurrence of atom a in lat. So (index-k 'b '(a b a b b) top) returns 1.

3. Give a tail-recursive continuation-passing-style function (max-k L k) that returns the largest element of the not-necessarily-flat list L of numbers. For example, (max-k '(5 3 (4 7 2 (5) 1)) top) returns 7

4. Give a tail-recursive continuation-passing style function (replace-k old new L k) that replaces each instance of atom old with atom new in the general list L. For example, (replace-k 'a 'x '(a b c (b c (a))) top) produces (x b c (b c (x)))

Part 2 – Backtracking

You should submit a Racket file lab05.rkt with solutions for these.

5. Write a solution to the subset sum problem. This means you should write a function (subsetSum goal nums) that returns a subset of the numbers in list nums whose values sum to the goal. If no subset of nums does this, the function should return the empty list.
   - (subsetSum 23 '(2 10 5 7 3 8 6)) returns (10 5 8) or (10 7 6) or (2 10 3 8)
   - (subsetSum 23 '(2 10 5 7)) returns ()

Hint: you might call backtracking function (ss goal numssofar) that fits our backtracking pattern; this function returns an extension of the listsofar with elements of nums that sum to the goal.
6. A “no-repeat” sequence is a sequence containing only the digits 1, 2, and 3 that does not contain two identical adjacent subsequences. For example (2 1 3 1 2 1) is a no-repeat sequence, but (1 2 3 3 2 1) is not (because 3 is a repeated subsequence of length 1), and (1 2 3 2 3 1) is not (because the subsequence (2 3) is repeated in adjacent spots).

Write a function (noRepeat n) that returns a no-repeat sequence of length n.

Hint: This is very similar to the n-Queens problem. The backtracking is actually straightforward; the more difficult part is writing a function (ok x A) that says if A is a no-repeat sequence then (cons x A) is also one. Since we know A is a no-repeat sequence we only need to check subsequences starting with x. Suppose A is the sequence (x1 x2 x3 ... xn). You need to check if (x) is the prefix of (x1 x2 x3 ...), then if (x x1) is the prefix of (x2 x3 x4 ...) then if (x x1 x2) is the prefix of (x3 x4 ...) and so forth.