```
Evaluate: (twice (twice (twice))) (twice) (plus_one) (4) t^{2((2^2)^2)} = t^{2(4)^2} = t^{216} = t^{65536}twice does tetration (right-to-left).
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             if (is_empty(stack)) {
    return "stack underflow";
} else {
    var top = peek(stack);
    set_tail(stack, tail(tail(stack)));
    return top;
            CS1101S Finals Cheatsheet v1.2
                                                                                                                                                                                               input-->|1||-->|||-->||/|
           (2019-05-01)
by indocomsoft, page 1 of 2
                                                                                                                                                                                                                                                      23
                                                                                                                                                                                                                                                                               4|/|
             Source Week 11 Syntax
                                                                                                                                                                                                                                                                                                                                                                                                  → counter, a - starting point, b - endpoint

// recursive version

function sum(term, a, next, b) {
 return a > b ? 0 : term(a) + sum(term, next(a), next,
 b);
 }
                                                                                                                                                                                                                                                                                                                                                                                                         nn
Sum: term - the operator, next - (de/in)crement
counter, a - starting point, b - endpoint
Autable version of list functions
function mutable_append(xs, ys) {
   if (is_empty_list(xs)) {
      return ys;
   } else {
      set_tail(xs, mutable_append(tail(xs), ys));
      return xs;
}
                                                                                                                                                                                                                                                                                                                                                                                               contains
function contains(x, xs) {
   return !is_empty_list(member(x, xs));
pointer-pointing-to-which-memory-address mechanism). So, JS-to-
Cequivalent:
• in(arr) \( \ext{in(arr)} \) fin(\( \arrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrantarrant
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     unction mutable_reverse(xs) {
function help(ori, rev) {
   if (is_empty_list(ori)) {
      return rev;
   } else {
                                                                                                                                                                                                                                                                                                                                                                                                       }
return sum_iter(a, 0);
                                                                                                                                                                                                                                                                                                                                                                                                ///iterative alternative
function new_map(f, xs) {
  function helper(xs, acc) {
    return is_empty_list(xs)
    ? acc
    : helper(tail(xs), pair(f(head(xs)), acc));
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   var temp = tail(ori);
set_tail(ori, rev);
return help(temp, ori);
General Functions
• parseInt(string)
• alert(string)
Primitive Data Type Checks
                                                                                           display(value)prompt(string)
alert(string)

Primitive Data Type Checks

is boolean is string is object is number is function is_array

PS: is object(array) returns truex

Is tlibray

pair(x,y): Makes a pair from x and y.
head(p): Returns the taded (first component) of the pair x.
tail(p): Returns the taded (first component) of the pair x.
set _head(p,x): Sets the head (first component) of the pair p to be x; returns undefined.
tall (p): Returns the tail (second component) of the pair p to be x; returns undefined.
tall (p): Returns the tail (second component) of the pair p to be x; returns undefined.
tall (p): Returns the tail (second component) of the pair p to be x; returns undefined.
tall (p): Returns the tail (second component) of the pair p to be x; returns undefined.
tall (p): Returns true fix is a tail st with n elements. The first element is x1, the second x2, etc. Iterative process; time: O(n), space: O(n), since the constructed list data structure consists of n pairs, each of which takes up a constant amount of space.
is_pair(p): Returns true if x is a pair and false otherwise.
seement yis tall (x); Returns true if x is a list as defined in the lectures if the pair tall (x); Returns true if x is a list as defined in the lectures if the pair tall (x); Returns true if x is a list as defined in the lectures if the pair tall (x); Returns true if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the lectures if x is a list as defined in the 
                                                                                                                                                                                                     }
return reverse(helper(xs, []));
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 8
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 Coin Change
                                                                                                                                                                                                                                                                                                                                                                                                function dequeue(x, q){ return tail(q); }
function qhead(q){ return head(q); }
Trees
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    function count_data_items(tree) {
  return is_empty_list(tree)
                                                                                                                                                                                                                                                                                                                                                                                                                                  ? 0
: (is_list(head(tree))
? count_data_items(head(tree))
: 1)
+ count_data_items(tail(tree));
                                                                                                                                                                                             }
// filter with accumulate
function new_filter(pred, xs) {
   return accumulate(function(e, acc){
        return pred(e) ? pair(e, acc)
        : acc;
                                                                                                                                                                                                                                                                                                                                                                                                } 
return helper(amount, length(lst));
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    return nespet(amount) lengun(1st)),

// Given coins in list 1, return list of coin

// permutations that add up to x

unction makeup amount(x,1) {

if (is_pair(1)) {

return append(map{function(x){ return}

} makeup_amount(x - head(1), x); },

makeup_amount(x - tail(1)));

helse if (x === 0) {
  } else if (x === 0) {
    return list([]);
} else {
    return [];
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     unction make_node(data, prev, next) {
  return list("node", data, prev, next);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     unction get_prev(node) {
  return head(tail(tail(node)));
                                                                                                                                                                                                                                                                                                                                                                                                  function make_node(left_tree, number, right_tree) {
  return list(left_tree, number, right_tree);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      unction get_next(node) {
return head(tail(tail(tail(node))));
                                                                                                                                                                                                                                                                                                                                                                                                 \label{eq:continuous} \begin{cases} \text{yunction middle(n) } \{ \text{ return math\_floor(n } / 2); \} \\ \text{function take(xs, n) } \{ \text{ return n} == 0 ? [] : \text{pair(head(xs), take(tail(xs), r} \rightarrow - 1));} \end{cases}
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    unction get data(node) { return head(tail{node));}
unction empty_node() { return make_node([], [], []);}
unction is_empty_node(node) {
    return head(node) == "node" && equal(node,
    empty_node());
                                                                                                                                                                                                                                                                                                                                                                                                   function drop(xs, n) {
  return n === 0 ? xs : drop(tail(xs), n - 1);
                                                                                                                                                                                                                                                                                                                                                                                                 unction set_data(n, v) {
  var temp = list(v, get_prev(n), get_next(n));
  set_tail(n, temp);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     unction set_previous(n, v) {
  var temp = list(get_data(n), v, get_next(n));
  set_tail(n, temp);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      unction set_next(n, v) {
  var temp = list(get_data(n), get_prev(n), v);
  set_tail(n, temp);
      Recursive process; time: O(n), space: O(n), where n is the length of xs.

of x
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      unction insert_before(n1, n2) {
    set_prev(n2, get_prev(n1));
    set_next(get_prev(n1), n2);
    set_prev(n1, n2); set_next(n2, n1);
                                                                                                                                                                                                                                                                                                                                                                                                 Thutable Data Structure

Tortoise and Hare Algorithm (check if list has a loop)

function has_loop(lst) {
    function tortoise_and_hare(tortoise, hare) {
        if (is_empty_list(tortoise) || is_empty_list(hare)
     where the first element has index 0. Iterative process; time: O(n) | Variations of accumulate () | variations of accumulate ()
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      unction insert_after(n1, n2) {
    set_next(n2, get_next(n1));
    set_prev(get_next(n1), n2);
    set_next(n1, n2); set_prev(n2, n1);
                                                                                                                                                                                                                                                                                                                                                                                                           in (is_empty_list(tortoise) || is_en

→ || is_empty_list(tail(hare))) {
    return false;
                                                                                                                                                                                                                                                                                                                                                                                                              } else {
return tortoise === hare
? true
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      unction remove(n) {
  var x = get_prev(n);  var y = get_next(n);
  set_next(x, y);  set_prev(y, x);
                                                                                                                                                                                                                                                                                                                                                                                                                                             sc Functions

equal(x, y) - true if pairs and corrs leaves are ===
array_length(arr) = 1+(highest index used)
apply_in_underlying_javascript(f, xs):
calls the function f with arguments xs. For example:
function times(x, y) {
return x * y;
                                                                                                                                                                                                                                                                                                                                                                                                       }
return tortoise_and_hare(lst, tail(lst));
                                                                                                                                                                                                                                                                                                                                                                                                 ,
Circular List
                                                                                                                                                                                                   /accumulate_tree(plus,0,list(1,2,list(3,4),
                                                                                                                                                                                                                                                                                                                                                                                                      rcular List
unction make_circular_copy(xs) {
function inner(zs, ys) {
  if (is_empty_list(zs)) {
    return ys;
}
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      unction list_to_dlist(lst) {
var first_node = make_node(head(lst), empty_node(),
                                                                                                                                                                                                }
apply_in_underlying_javascript(times, list(2, 3));
// returns 6
                                                                                                                                                                                                                                                                                                                                                                                                             return ys;
} else {
return pair(head(zs), inner(tail(zs), ys));
// returns 6
OOP Inheritance, method declaration, constructor example code
function icsbot(name){ Player.call(this, name); }
icsbot.Inherits(Player);
icsbot.prototype._act = function(){
    Player.prototype._act.call(this);
                                                                                                                                                                                                                                                                                                                                                                                                     if (is_empty_list(xs)) {
    return [];
    else {
        var ys = pair(head(xs), []);
        set_tail(ys, inner(tail(xs), ys));
    return ys;
    }
}
                                                                                                                                                                                              unction reverse_dlist(dlst) {
    return list_to_dlist(reverse(dlist_to_list(dlst)));
                                                                                                                                                                                                                                                                                                                                                                                                    unction make_linear(xs) {
  function helper(lst) {
    if (tail(lst) === xs) {
        set_tail(lst, []);
    } else {
        helper(tail(lst));
    }
}
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         Permutations & Combinations
                                                                                                                                                                                                     }
return iter(initial, sequence);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     Permutations (O(n!)) and permutations_r (O\left(\frac{n!}{(n-r)!}\right))
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   Queue in O(1) using mutable data structure
                                                                                                                                                                                                                                                                                                                                                                                                  // Queue data structure
function make_queue() { return pair([], []); }
function is_empty_queue(q) {
    return is_empty_list(head(q));
}
2 Complexity
- Big Oh O() : Upper bound (worst case)
- Big Omega \Omega() : Lower bound (best case)
- Big Theta \Theta() : Both upper and lower bound
- Order :
\Theta(1) < \Theta(\log n) < \Theta(n) < \Theta(n\log n) < \Theta(n^2) < \Theta(n^3) < \Theta(2^n) < \Theta(2^n) < \Theta(n^n)
                                                                                                                                                                                                                                                                                                                                                                                                  function enqueue(q, item) {
    if (is_empty_queue(q)) {
        set_head(q, pair(item, []));
        set_tail(q, head(q));
    } else {
        set_tail(tail(q), pair(item, []));
        set_tail(q, tail(tail(q)));
    }
}
                                                                                                                                                                                                                           return ():
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   introduction permutations_r(s, r) {
    if (r === 0) { return list([]);
    } else if(is_empt_list(s)) { return list();
    } else {
        return accumulate(append, [], map(function(x) {
            return pair(x, p);
            }, permutations_r(remove(x, s),
            \limits r - 1));
    }
}
                                                                                                                                                                                                     }
return helper(x);
3 Box-and-Pointer Diagrams
                                                                                                                                                                                                         Higher Order Function
                                                                                                                                                                                                  Compose,Identity
Function compose(f, g){
return function(x){ return f(g(x)); };
var w1 = list(1,2); var w2 = list(3,4);
var w3 = append(w1,w2);
                                                                                                                                                                                                                                                                                                                                                                                                 function dequeue(q) {
  var front = head(head(q));
  set_head(q, tail(head(q)));
  return front;
w1->|1| |-->|2|/|
                                                                                                                                                                                                   unction identity(x) {
  return x;
+-
w2----
                                                                                                                                                                                                                                                                                                                                                                                                  function peek(q) { return head(head(q)); }
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  Combinations O\left(\frac{n!}{(n-k)!k!}\right) = O\left(n^{n+2}\right)
                                                                                                                                                                                                 } unction repeated(f, n) { return (n === 0) ? identity : compose(f, repeated(f, \hookrightarrow n - 1));
                                                                                                                                                                                                                                                                                                                                                                                               Stack
function make_stack() { return pair("stack", []); }
function is_empty_stack(stack) {
   return is_empty_list(tail(stack));
}
w3->|1||-->|2||-->|3||-->|4|/|
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    function combinations_k(xs, k)
if (k === 0) { return list([]);
else if (is_empty_list(xs)) { return [];
else {
var s1 = combinations(tail(xs), k - 1);
var s2 = combinations(tail(xs), k);
var x = head(xs);
var x = head(xs);
var x = head(xs);

}
function peek(stack) {
  return is_empty_stack(stack) ? "stack underflow"
  : head(tail(stack));
                                                                                                                                                                                                 function plus_one (x) {
return x + 1.
                                                                                                                                                                                                ,
function twice (f) {
  return function(x) { return f(f(x)); };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           var mas_^ - ..., .

⇔ s1);

return append(has_x, s2);
                                                                                                                                                                                                                                                                                                                                                                                                function push(stack, x) {
    set_tail(stack, pair(x, tail(stack)));
      |1| |-->|3|4|
```

function pop(stack) {

var input = list(1, pair(2,3), list(4)); var result = map(function(x){return x;}, input);

```
function search(low, high) {
  if (low > high) {
    return -1;
} else {
    var mid = math_floor((low + high) / 2);
    if (target === arr[mid]) {
        return mid;
} else {
}
             CS1101S Finals Cheatsheet v1.2
             (2019-05-01)
             by indocomsoft, page 2 of 2
Power Set
function power_set(xs) {
  if (is_empty_list(xs)) {
    return list([]);
                                                                                                                                                                                                                               return mig;
} else {
return target < arr[mid]
? search(low, mid - 1)
: search(mid + 1, high);
   }
                                                                                                                                                                                                                }
var len = array_length(arr);
var pivot = findPivot(0, len - 1);
if (pivot === -1) {
    return search(0, len - 1);
    else if (arr[0]) > target) {
        return search(pivot+1, len - 1);
    } else {
        return search(0, pivot);
    }
 if function are_equal_sets(set1, set2) {
   if(length(set1) === length(set2)) {
     return false;
   }
}
           ness
DG Week 3
                                                                                                                                                                                                              \label{eq:continuous_squares_two_larger} \begin{subarray}{ll} $\sf Meek 3 \\ $\sf uunction \ sum\_squares\_two\_larger(a, b, c) \{ \\ $\sf return \ a > b \ ? \ a * a + (\{b > c) \ ? \ b * b : c * c \} \\ $\sf : b * b + (\{a > c) \ ? \ a * a : b * b) \end{subarray}
10 Tower of Hanoi
Tower of Hanoi (Running time: O(2^{jt})) function make move(from, to) { return list(from, to); } function hanoi(n, A, B, C) { if (n == 1) { return list(make_move(A, B));
                                                                                                                                                                                                                unction hanol(n, A, D, C, t if (a == 1) { return list(make_move(A, B)); } else { var moves_to_helper = hanoi(n - 1, A, C, B); var the_move = make_move(A, B); var moves_to_final = hanoi(n - 1, C, B, A); return append(moves_to_helper, pair(the_move, → moves_to_final));
                                                                                                                                                                                                         }
DGWeek 4
// Pascal Triangle
function pascal(row, column){
   if(column === 1 || row === column){      return 1;
   } else {
      return pascal(row - 1, column - 1) + pascal(row - 1, column);
}
11 Sorting functions & Binary Search
Selection Sort (\Theta(n^2))
                                                                                                                                                                                                               // AddItIonal
// AddItIonal
// Indition(a,n){return repeated(add_one,n)(a);}
unction multiplication(a,n){
var addition_a = function(x){return addition(a,x);};
return repeated(addition_a,n)(0);
   function selection_sort(xs) {
  if (is_empty_list(xs)) { return xs;
  } else {
                  erse {
var x = smallest(xs);
return pair(x, selection_sort(remove(x, xs)));
                                                                                                                                                                                                                unction exponentiation(a,n){
  var mul_a = function(x){return multiplication(a,x);};
  return repeated(mul_a,n)(1);
  unction tetration(a,n){
var exp_a = function(x){return exponentiation(a,x);};
return repeated(exp_a,n)(1);
                                                                                                                                                                                                           13 Functions from Past Year Papers
           return sm(head(xs), tail(xs));
                                                                                                                                                                                                                unction is_tee o_limber(sty) / // Solution i [5 marks]

**Return is_list(x) / Fourier is_list(x) / Return (is_number(a) | |

**A accumulate(function(a,b){

**Return (is_number(a) | |

**A is_tree_of_numbers(a)) & & b;

**Iree_of_numbers(a)) & & b;

**If (is_empty_list(x) {

**Peturn (is_number(head(x))) | |

**Teturn (is_number(head(x))) & & b;

**Teturn (is_number(head(
| Insertion Sort (Input-dependent, function insert(x, xs) \ return is compty. If t(xs) ? If t(xs) ? Pair(x,xs) : t(xs) : pair(head(xs), t(xs) : insert(x,tai)
Insertion Sort (Input-dependent, best:\Omega(n), worst:O(n^2))
                                                                                                             insert(x,tail(xs)));
⇒ is_tree_of_numbers(head(x))) &&

⇒ is_tree_of_numbers(tail(x));

} else { return false;
Merge Sort (Running time: \Theta(n \log n))
                                                                                                                                                                                                               unction my_filter(pred, xs) {
  return accumulate(function(a, b) {
      return pred(a) ? pair(a, b) : b;
      }, [], xs);
function drop(xs, n) {
    return n === 0 ? xs : drop(tail(xs), n - 1);
                                                                                                                                                                                                                         return n ==- 

function merge(xs, ys) {
    if (is_empty_list(xs)) { return ys;
        | else if (is_empty_list(ys)) { return xs;
        | else if (is_empty_list(ys)) { return xs;
        | else if (is_empty_list(ys)) { return (x < y) ? pair(x, merge(tail(xs), ys)) return (x < y) ? pair(x, merge(tail(xs), ys));
        | pair(y, merge(xs, tail(ys)));
}
                                                                                                                                                                                                                        break;
} else { }
fvar n = -1;
for(i = 0; i < 4; i = i + 1) {
   if (arr[i]===0 && n === -1) {</pre>
                                                                                                                                                                                                                          11 (alt[i]==-0 && n ===-i) {
    n = i;
} else if (arr[i]!== 0 && n > -1) {
    arr[n]=arr[i];
    arr[i]=0;
                                                                                                                                                                                                                         i=n;
n=-1;
} else { }
Quicksort (Input-dependent, best: \Omega(n \log n), worst: O(n^2))
unction slideLeft(grid) {
                                                                                                                                                                                                                   for(var i = 0; i < 4; i = i + 1) {
    slideRowToTheLeft(grid[i]);
                                          unction rotateLeft(grid) {
                                                                                                                                                                                                                   ret = [];

for(var i = 0; i < 4; i = i + 1) {

  ret[i]=[];
                                                                                                                                                                                                                   }
for(var i=0; i < 4; i = i + 1) {
  for(var j=0; j < 4; j = j + 1) {
    ret[3-j][i]=grid[i][j];
    ret[3-j][i]=grid[i][j];
}</pre>
      } '
return helper(xs, [], []);
}
return ret;
                                                                                                                                                                                                              function performSlide(grid, direction) {

If (direction == "left") {
	return slideleft(grid);
	} else if (direction === "up") {
	return rotateleft(rotateleft(rotateleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sideleft(sidel
     inary Search (O(\log n)), returns true/false
                                                                                                                                                                                                                  } else {
  return rotateLeft(slideLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(rotateLeft(
function binary_search(a, v) {
  function search(low, high) {
    if (low > high) {
      return false;
    }
}
          } return search(0, array_length(a) - 1);
nary Search on sorted rotated array, (O(\log n)), returns the inde
                                                                                                                                                                                                             }
// example
var fib=memoize(function(n){
   return n <= 1 ? n : fib(n-1) + fib(n-2);
});</pre>
                                                                                                                                                                                                              Memoized Coin Change
                                                                                                                                                                                                            var mem = [];
function read(amount, coin_range) {
  return (mem[amount] === undefined) ? undefined :
                                                                                                                                                                                                                               mem[amount][coin_range];
                                                                                                                                                                                                              }
function write(amount, coin_range, value) {
  if (mem[amount] === undefined) {
    mem[amount] = [];
```

```
} else {}
mem[amount][coin_range] = value;
  ensional Memoization (e.g. Tower of Hanoi)
     If (unem[w][x] === underIned) { mem[w][x] = []; } if (mem[w][x][y] == []; } else {} mem[w][x][y][z] = value;
   tunction make_move(From, To) { return list(From, To); }
unction retriever(n, A, B, C);
unction retriever(n, A, B, C);
if (result == undefined) {
    result == undefined) {
    result == undefined);
    verite(n, A, B, C);
    write(n, A, B, C, result);
} else {
} return result;
  iunction mhanoi(n, A, B, C) {
    var result = retriever(n, A, B, C);
    if (result !== undefined) {      return result;
    } else {
        if (n === 1) {            return list(make_move(A, B));
        } else {
            var moves_to_helper = retriever(n - 1, A, C, B);
            var moves_to_final = retriever(n - 1, C, B, A);
            return append(moves_to_helper, append(moves_to_helper);
        }
    }
}
// mhanoi (8, 1, 2, 3);

Variation of Knapsack Problem

A thief has a getaway car that can hold at most m kg (m is an inte ger). Write a program that takes in a list of the gold bar masses and determines the total mass of gold bars he can steal. Time complex
  Non-Memoized: O(2<sup>length</sup>(bar_masses)),
  Memoized: O(length(bar_masses) \cdot m)
   / Memoized version
unction max_loot_mass(bar_masses, m) {
  var mem = [];
  function write(len, m, value) {
    if (memllen) === undefined) {
      mem[len] = [];
    } else {
      countw = countw + 1;
      mem[len][m] = value;
    }
}
     function read(len, m) {
  if (mem[len] === undefined) { mem[len] = [];
  } else { countr = countr + 1;
           return mem[len][m];
    }
return helper(bar_masses, m);
    napsack Problem: Herbert the Clown

fost expensive combination with a given budget from a list of in
     = Budget
1 = List of Ingredients Price List
   } else { return -1;
 }
var i_l = list(list(4,6,8), list(5,10), list(1,3,5,5));
getMaxSpend(11, i_l); // returns 10 (4+5+1)
getMaxSpend(20, i_l); // returns 19 (8+10+1)
```

```
15 Environment Model
                              unction ham(x) {
var y = 3;
function gam(x) {
 ar x = 4,
unction foo(x) {
var v = x * 2;
 var y = x *
if(y===10) {
x = x + 5:
                                y = y + x
return y;
 x = x + 0,
return x;
} else {
return foo(x+1);
                              }
return gam;
                            var h = ham(5);
h(1);
         x: 4
         foo:
             6
  global other variables...
            16 Contest Entries for "inspiration"
      e are just padding
NUS School of Computing
17 Dank Meme
    HENZ'S
     Essence
 (4)
        of Recursion
 A Free Append
                                              Added Salt
      20th Anniversary, from IC1101S to
      CS1101S, from 1997 to 2017
```