

$$\left. \begin{array}{l} \{ \\ \{ \end{array} \right\}$$

Power Set

```
function power_set(xs) {
  if (is_empty_list(xs)) {
    return list([]);
  } else {
    // Either you pick the number, or you don't
    var without_head = power_set(tail(xs));
    var use_head = map(function(x) {
      return pair(head(xs), 1);
    }, without_head);
    return append(without_head, use_head);
  }
}

function are_equal_sets(set1, set2) {
  if (length(set1) !== length(set2)) {
    return false;
  } else {
    return accumulate(function(e1, acc) {
      return accumulate(function(e2, acc) {
        return e2 === e1 || acc;
      }, false, set2);
    }, true, set1);
  }
}
```

10 Tower of Hanoi

Tower of Hanoi (Running time: $O(2^n)$)

```
function make_move(from, to) { return list(from, to); }
function hanoi(n, A, B, C) {
  if (n === 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));
  }
}
```

11 Sorting functions & Binary Search

Selection Sort ($O(n^2)$)

```
function selection_sort(xs) {
  if (is_empty_list(xs)) { return xs; }
  else {
    var x = smallest(xs);
    return pair(x, selection_sort(remove(x, xs)));
  }
}

// find smallest element of a non-empty list xs
function smallest(xs) {
  function sm(x, ys) {
    return is_empty_list(ys)
      ? x
      : x < head(ys) ? sm(x, tail(ys))
      : sm(head(ys), tail(ys));
  }
  return sm(head(xs), tail(xs));
}
```

Insertion Sort (Input-dependent, best: $O(n)$, worst: $O(n^2)$)

```
function insert(x, xs) {
  return is_empty_list(xs)
    ? list(x)
    : x <= head(xs) ? pair(x, xs)
    : pair(head(xs), insert(x, tail(xs)));
}

function insertion_sort(xs) {
  return is_empty_list(xs)
    ? xs
    : insert(head(xs), insertion_sort(tail(xs)));
}
```

Merge Sort (Running time: $\Theta(n \log n)$)

```
function middle(n) { return math_floor(n / 2); }
function take(xs, n) {
  return n === 0 ? []
    : pair(head(xs), take(tail(xs), n - 1));
}

function drop(xs, n) {
  return n === 0 ? xs
    : drop(tail(xs), n - 1);
}

function merge(xs, ys) {
  if (is_empty_list(xs)) { return ys; }
  else if (is_empty_list(ys)) { return xs; }
  else {
    var x = head(xs); var y = head(ys);
    return (x < y) ? pair(x, merge(tail(xs), ys))
      : pair(y, merge(xs, tail(ys)));
  }
}

function merge_sort(xs) {
  if (is_empty_list(xs) || is_empty_list(tail(xs))) {
    return xs;
  } else {
    var mid = middle(length(xs));
    return merge(merge_sort(take(xs, mid)),
    merge_sort(drop(xs, mid)));
  }
}
```

Quicksort (Input-dependent, best: $O(n \log n)$, worst: $O(n^2)$)

```
function partition(xs, p) {
  function helper(lst, smaller, larger) {
    if (is_empty_list(lst)) {
      return pair(smaller, larger);
    } else {
      var cur = head(lst);
      return cur <= p
        ? helper(tail(lst), pair(cur, smaller),
        larger)
        : helper(tail(lst), smaller, pair(cur,
        larger));
    }
  }
  return helper(xs, [], []);
}

function quicksort(xs) {
  if (is_empty_list(xs)) { return xs; }
  else if (is_empty_list(tail(xs))) { return xs; }
  else {
    var cur = head(xs);
    var partitioned = partition(tail(xs), cur);
    var sorted_smaller = quicksort(partitioned[0]);
    return append(sorted_smaller, pair(cur,
    sorted_larger));
  }
}
```

Binary Search ($O(\log n)$), returns true/false

```
function binary_search(a, v) {
  function search(low, high) {
    if (low > high) {
      return false;
    } else {
      var mid = math_floor((low + high) / 2);
      return (v === a[mid])
        ? search(low, mid - 1)
        : search(mid + 1, high);
    }
  }
  return search(0, array_length(a) - 1);
}
```

Binary Search on sorted rotated array, ($O(\log n)$), returns the index if found, -1 if not found

```
function find(target, arr) {
  function findPivot(low, high) {
    if (low > high) {
      return -1;
    } else {
      var mid = math_floor((low + high) / 2);
      if (mid < high && arr[mid] > arr[mid + 1]) {
        return mid;
      } else if (mid < low && arr[mid] < arr[mid - 1]) {
        return mid - 1;
      } else if (arr[mid] > arr[low]) {
        return findPivot(mid + 1, high);
      } else {
        return findPivot(low, mid - 1);
      }
    }
  }
}
```

```
function search(low, high) {
  if (low > high) {
    return -1;
  } else {
    var mid = math_floor((low + high) / 2);
    if (target === arr[mid]) {
      return mid;
    } 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);
}
```

12 Functions of questionable usefulness

DG Week 3

```
function sum_squares_two_larger(a, b, c) {
  return a > b ? a * a + ((b > c) ? b * b : c * c)
    : b * b + ((a > c) ? a * a : b * b);
}
```

DG Week 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);
  }
}
```

Additional

```
function addition(a, n) { return repeated(add_one, n)(a); }
function multiplication(a, n) {
  var addition_a = function(x) { return addition(a, x); };
  return repeated(addition_a, n)(0);
}

function exponentiation(a, n) {
  var mul_a = function(x) { return multiplication(a, x); };
  return repeated(mul_a, n)(1);
}

function tetration(a, n) {
  var exp_a = function(x) { return exponentiation(a, x); };
  return repeated(exp_a, n)(1);
}
```

13 Functions from Past Year Papers

Solution 1 { 3 marks }

```
function is_tree_of_numbers(x) {
  // Solution 1 { 3 marks }
  return is_list(x)
    && accumulate(function(a, b) {
      return (is_number(a) ||
      is_tree_of_numbers(a)) &&
      b;
    }, true, x);
}

// Solution 2 { 3 marks }
if (is_empty_list(x)) { return true; }
else if (is_pair(x)) {
  return (is_number(head(x)) ||
  is_tree_of_numbers(head(x))) &&
  is_tree_of_numbers(tail(x));
} else { return false; }
```

2048

```
function slideRowToLeft(arr) {
  for (var i = 0; i < 4; i = i + 1) {
    for (var j = i + 1; j < 4; j = j + 1) {
      if (arr[j] === arr[j + 1] && arr[j + 1] !== 0) {
        arr[j] = 2 * arr[j + 1];
        arr[j + 1] = 0;
        break;
      }
    }
  }
  return arr;
}

var n = -1;
for (i = 0; i < 4; i = i + 1) {
  for (j = 0; j < 4; j = j + 1) {
    if (arr[j] === 0 && n !== -1) {
      n = j;
    } else if (arr[j] !== 0 && n > -1) {
      arr[n] = arr[j];
      arr[j] = 0;
      n = -1;
    }
  }
}
return arr;
}
```

```
function rotateLeft(grid) {
  var ret = [];
  for (var i = 0; i < 4; i = i + 1) {
    ret[i] = [];
    for (var j = 0; j < 4; j = j + 1) {
      ret[3 - j][i] = grid[i][j];
    }
  }
  return ret;
}

function performSlide(grid, direction) {
  if (direction === "left") {
    return slideLeft(grid);
  } else if (direction === "up") {
    return rotateLeft(rotateLeft(rotateLeft(
    slideLeft(rotateLeft(grid)))));
  } else if (direction === "right") {
    return rotateLeft(rotateLeft(slideLeft(rotateLeft(
    rotateLeft(grid))));
  } else {
    return rotateLeft(slideLeft(rotateLeft(rotateLeft(
    rotateLeft(grid))));
  }
}
```

14 Memoization

1D Memoization Abstraction

```
function memoize(f) {
  var mem = [];
  return function(x) {
    if (mem[x] !== undefined) { return mem[x]; }
    else {
      var result = f(x);
      mem[x] = result;
      return result;
    }
  };
}

// example
var fib-memoize = function(n) {
  return n <= 1 ? n : fib(n - 1) + fib(n - 2);
};

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] = [];
  }
}
```

```
function cc(amount, coin_range) {
  if (read(amount, coin_range) !== undefined) {
    return read(amount, coin_range);
  } else {
    var result = amount === 0 ? 1
      : (amount < 0 || range === 0) ? 0
      : cc(amount, range - 1) + cc(amount,
      range);
    write(amount, coin_range, result);
    return result;
  }
}

Multi-dimensional Memoization (e.g. Tower of Hanoi)
var mem = [];
function read(w, x, y, z) {
  return (mem[w] === undefined) ? undefined
    : (mem[w][x] === undefined) ? undefined
    : (mem[w][x][y] === undefined) ? undefined
    : mem[w][x][y][z];
}

function write(w, x, y, z, value) {
  if (mem[w] === undefined) { mem[w] = []; }
  else {
    if (mem[w][x] === undefined) { mem[w][x] = []; }
    else {
      if (mem[w][x][y] === undefined) { mem[w][x][y] = []; }
      else {
        mem[w][x][y][z] = value;
      }
    }
  }
}

function make_move(From, To) { return list(From, To); }
function retriever(n, A, B, C) {
  var result = retriever(n, A, B, C);
  if (result !== undefined) { return result; }
  else {
    write(n, A, B, C, result);
  }
}

function 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 the_move = make_move(A, B);
      var moves_to_final = retriever(n - 1, C, B, A);
      return append(moves_to_helper,
      pair(the_move, moves_to_final));
    }
  }
}
```

Variation of Knapsack Problem

A thief has a getaway car that can hold at most m kg (m is an integer). 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 complexity:

Non-Memoized: $O(2^{\text{length}(\text{bar_masses})})$,
Memoized: $O(\text{length}(\text{bar_masses}) \cdot m)$

```
// Non-memoized version
function max_loot_mass(bar_masses, m) {
  if (is_empty_list(bar_masses) || head(bar_masses) > m) {
    return 0;
  } else {
    var cur = head(bar_masses);
    // Either take the current bar or not
    return math_max(cur +
    max_loot_mass(tail(bar_masses), m - cur),
    max_loot_mass(tail(bar_masses),
    m));
  }
}

// Memoized version
function max_loot_mass(bar_masses, m) {
  var mem = [];
  function write(len, m, value) {
    if (mem[len] === undefined) {
      mem[len] = [];
    } else {
      mem[len][m] = value;
    }
  }

  function read(len, m) {
    if (mem[len] === undefined) { mem[len] = []; }
    else {
      return mem[len][m];
    }
  }

  function helper(bar_masses, m) {
    if (is_empty_list(bar_masses) || head(bar_masses) >
    m) {
      return 0;
    } else {
      var cur = head(bar_masses);
      var len = length(bar_masses);
      var res = read(len, m);
      if (res === undefined) {
        res = math_max(cur + helper(tail(bar_masses),
        m - cur), helper(tail(bar_masses), m));
        write(len, m, res);
      }
      return res;
    }
  }

  return helper(bar_masses, m);
}
```

Knapsack Problem: Herbert the Clown

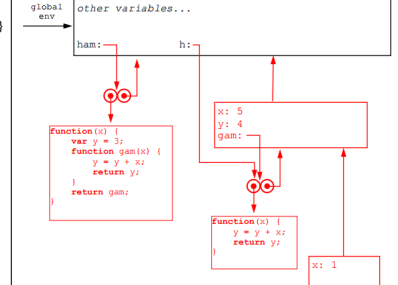
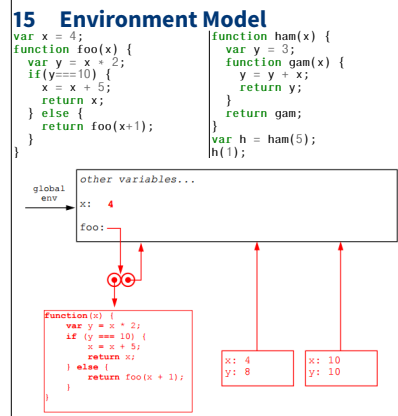
Most expensive combination with a given budget from a list of ingredients.

b = Budget
 i_1 = List of Ingredients Price List

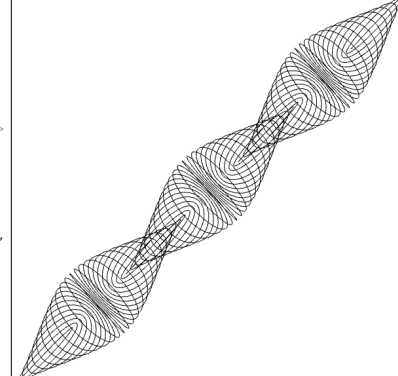
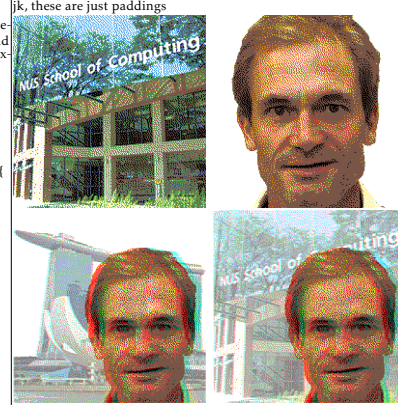
```
// Assume all sublists are sorted in ascending order
function is_sufficient_fund(b, i_1) {
  return is_empty_list(i_1)
    ? b >= 0
    : is_sufficient_fund(b - head(head(i_1)),
    tail(i_1));
}

if (is_sufficient_fund(b, i_1)) {
  if (is_empty_list(i_1) ||
  is_empty_list(head(i_1)) || head(head(i_1)) >
  b) {
    return 0;
  } else {
    var cur = head(head(i_1));
    var x = cur + getMaxSpend(b - cur, tail(i_1));
    var y = getMaxSpend(b, pair(tail(head(i_1)),
    tail(i_1)));
    return math_max(x, y);
  }
} else {
  return -1;
}
```

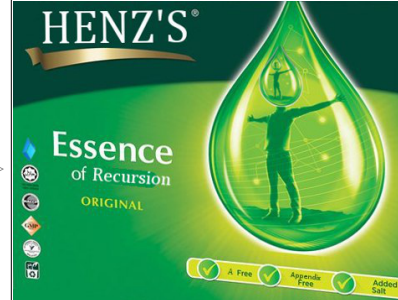
```
var i_1 = list(list(4, 6, 8), list(5, 10), list(1, 3, 5, 5));
getMaxSpend(11, i_1); // returns 10 (4+5+1)
getMaxSpend(20, i_1); // returns 19 (8+10+1)
```



16 Contest Entries for "Inspiration"



17 Dank Meme



18 20th Anniversary, from IC1101S to CS1101S, from 1997 to 2017

