# ComS 311
# Recitation 3, 2:00 Monday
# Homework 6

Sean Gordon

December 7, 2019

1) Recurrence:
combo = null if remainder < 0
combo = [ ] if remainder == 0
combo = min( from(i = 0 → n) combo(set, remaining - set[i], list) )

**Algorithm 1** Find non-negative integers w1, ..., wn.

```
int[] comboIter(set, W){

//Store sub problem answers here
int[] sub = new int[W+1]
*Assign each index in sub to infinity

//Store the indices of the num used for each sum here
int[] indices = new int[W+1]

//Sum of 0 will always be an empty set
sub[0] = 0

for (int i = 1; i < W; i++) do
    for (int j = 0; j < set.length; j++) do
        int num = set[j]

        //If the current number can fit
        if (num <= i) then
            //Replace current answer if it works better
            int current = sub[i]
            int new = sub[i - num] + 1

            if (new < current) then
                //Update our records
                sub[i] = new
                //Save the index of the number that was used
                indices[i] = j
            end if
        end if
    end for
end for

//If the last cell is unused, there is no answer
if (sub[W] == infinity) then
    return []
end if


//Otherwise use the stored indices to build a list of [w1, ..., wn]
int[] w = new int[set.length]
*Assign each index in w to 0

//Backtrack through our lists
int i = W
while (i > 0) do

    //Increment the use count (w) of each number used in the end
    int index = indices[i]
    w[index]++

    //Go to the number used before this one
    i -= set[index]
end while

return w
}

Runtime: O(n*W)
```

2) Recurrence:
func = true: spaceLeft == 0 && remainingSum == 0
func = false:(spaceLeft == 0 && remainingSum != 0) || index == U.length
func = func(U, spaceLeft, remainingSum, index+1) ||
    func(U, spaceLeft-1, remainingSum - U[index], index+1);

**Algorithm 2** Test for subset of U of size k that adds to T.

```
boolean iterFunc(U, T, k){

n = U.length;

//Make a 2d array to map subsets
matrix[ ][ ] = new boolean[n][T+1];      //n, 0→T (not 1→T)

//Make hashmap to track the lengths of each subset that adds to a sum
//key = current sum, value = array of subset lengths
Hashmap legths = new HashMap<Integer, List<Integer>>();

//Set column 0 to true, as all sums == 0 use empty set
for int i = 0; i <= n; i++ do
    matrix[i][0] = true;
end for

//For each number in the set
for int i = 0; i < n; i++ do
    int number = U[i];

    //From 1→T
    for int sum = 1; sum <= T; sum++ do

        //If this number is too big, grab the val above
        if number > sum then
            matrix[i][sum] = matrix[i-1][sum];
            continue;
        end if


        //Decide if # can be added to a prev subset to fit current sum
        //Use typical subset-sum lookback
        result1 = matrix[i - 1][sum - number];

        if result then
            //We are adding this to the subset
            //Ex: if sum = 14, number = 9, and lengths@5 = [1, 3, 4],
            //lengths@14 will now = [2, 4, 5]
            //** This is a loop through an array **
            lengths@sum = lengths@(sum - number)++;
        end if

        //If this number won't fit, we don't add it to the subsets, but
        //this sum may still be possible, so the matrix should reflect that
        result2 = result || matrix[i-1][sum];
        matrix[i][j] = result2;
    end for
end for


if ! (matrix[n-1][T]) then
    return false;
end if

//If there is a possible subset, check that there is one of length k
list = lengths@T

for int i = 0; i < list.length; i++ do
    if(list[i]== k) return true;
end for
return false;
}


——————————— Runtime: O(n*T*k) ———————————
```

3) Recurrence:

```
traverse = score if x==M && y==N
traverse = max(traverse(maze, M, N, x,   y+1, score-2),
               traverse(maze, M, N, x+1, y,   score-2),
               traverse(maze, M, N, x+1, y+1, score-3))
```

**Algorithm 3** Maximize score in M x N maze

```
int iterTraverse(maze, M, N){

//Go from left→right, top→bottom,                            [X][X]
//looking at the max of cells to left, top left, and top [X][O]

for int x = 1; x <= M; x++ do
    for int y = 1; y <= N; y++ do
        //Find largest score for transitioning to this cell
        int score = max(maze[x-1][y] -2, maze[x-1][y-1] -3, maze[x][y-1] -2)

        //Replace maze slot with score + maybe diamond,
        // as we no longer need it
        maze[x][y] += score
    end for
end for

return maze[M][N]
}


———————————— Runtime: O(M*N) ————————————
```