Camilo Sarabia
Project 3 Report

1) To model the problem input, I will use a Directed graph. I will construct this graph by using an adjacency list. My vertices will be the buckets and all the possible combinations that I can have for these. For example, if I have the two buckets, one of size 5 and one of size 3, I will have 24 vertices (6*4, since we start counting at 0) and they will be (0,0)(0,1)(0,2)(0,3)(1,0)(1,1) so on and so forth. My edges will be the possible moves I can make at a certain vertex. Let's say for example that I am at vertex (0,0) the only moves I can make is to fill either one of these buckets, so my edges for this Vertex will be (5,0) and (0,3). My graph will be directed, unweighted, and vertex-labelled.

2) I will use Bread-First search as my graph algorithm

```
BFS(vector<Vertex> )
{
        Initialize sum to 0
        Initialize count to 0

        Initialize a map of integers to all 0's
        Initialize vector called moves to all 0's

        For my vector of vertex's, mark all the vertices as not visited

        Initialize a queue of vertices
        Mark the first vertex in my vector to visited and level to 0

        Push back the first vertex into my queue

        While Queue not empty
        {

                Create a new vertex called current vertex
                Assign queue.front to current vertex
                Queue.pop_front

                For int I to currents vertex size
                        Sum all the integers in my vertex //if I have (5,0) sum would be 5

                If moves at sum is equal to the current vertexs level do nothing
                Else moves at sum is equal to current vertexs level

                Map the index to the level
```

```
        For int I to my currents vertexs neighbors size
        {

                If my neighbor has not been visited
                {
                        My neighbors level will be its parents level+1
                        Mark the neighbor as visited
                        Push the neighbor into the queue
                }

        }

    Convert the vector moves to a max heap and extract the first element

    Return a vector that contains the number of moves and the amount of water that
    required that many moves

}
```