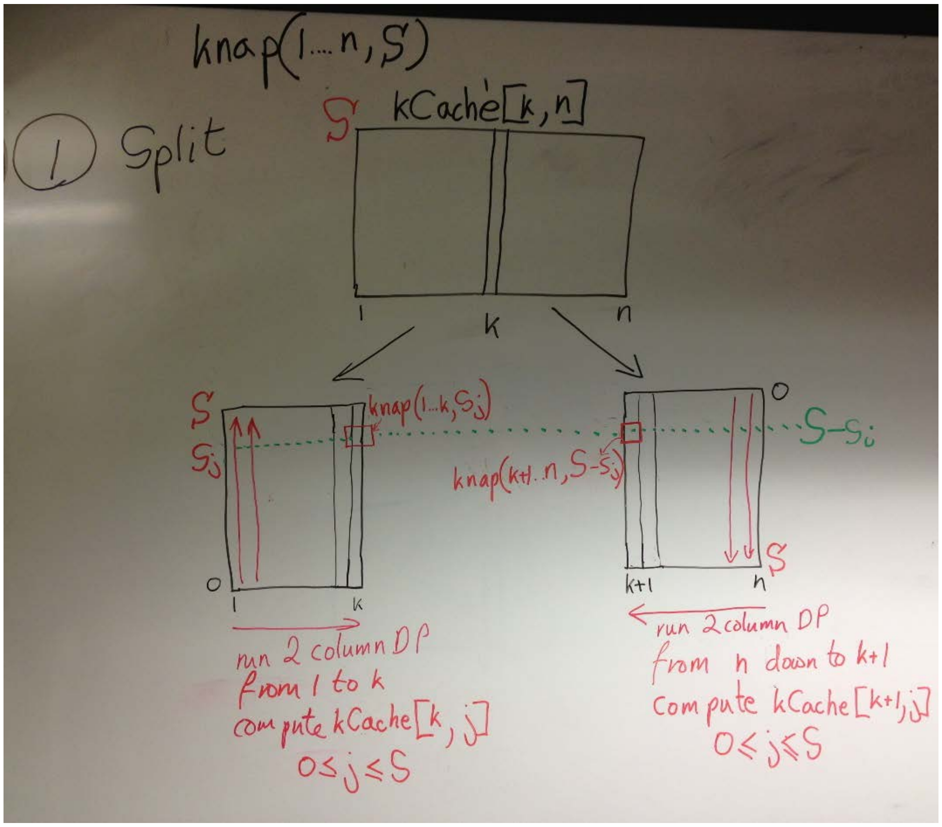
**CS 5050 Assignment Two 40 points**

1) Implement the trace-back routine that takes the 2D cache array and returns an assignment of true or false to each object in the problem description, where true means the object is selected.
2) Implement the linear space knapsack algorithm described in the nodes and summarized below to solve the same problem statement as Assignment 1. The linear-space divide and conquer algorithm should divide the problem in to equal halves (so k is n/2).
3) For a set of small problem verify that the three algorithms (linear, standard DP and caching) produce the same answer.
4) Once you have verified that the code is running correctly perform the following empirical studies:
    a. For randomized data generate a graph that measures the average cpu time of the **divide and conquer linear-space solution** as a function of the problem size. Given n is the problem size and m is the capacity of the knapsack for each experiment set m= 10*n. In each experiment, generate the sizes of the n objects need from a uniform random distribution between 1 and m/10 (the distribution does not matter since we are using DP). Start at size 64 and go as large as you can within reasonable running times, increasing the problem size by a factor of 2 each time. Produce a log-log graph where x is the log of the size of the problem and y is the log of the average running time.
    b. Compare the performance of the new D&C algorithm with your previous DP+traceback algorithm. What is the difference between the lines? How much larger problems can you solve with the D&C compared to the original algorithm.
5) Your graphs should be clearly captioned with the axis labeled. I recommend using gnuplot or some other professional quality graphing software. Whatever you use, you must have the correct log-log graph.
6) Write a brief (4-6 sentence) technical explanation of the behavior of the algorithms derived from the graphs.
7) The 40 points will be awarded as follows: 20 points fully working program that correctly implements the linear-space algorithm; 10 points for the empirical studies and line fitting; 10 points for the correct graphs and correct technical explanation of the behavior.
8) Submit your graphs and report, data files containing your raw data and your commented code along with instructions to run the code.

```
// sets the values in kUse
knapDC(lowIndex, highIndex, capacity)
     // single object, use it if it fits (value always > 0)
        if(lowIndex == highIndex)
                kUse[lowIndex] = (size[k] <= capacity)
        else
                midIndex = (lowIndex + highIndex)/2
                // linear space, scans from low to mid, returns last column 0 to capacity
                leftColumn = knapDPLinearLeft(lowIndex, midIndex, capacity)
                // linear space, scans from high down to midIndex+1, returns last column
                rightColumn = knapDPLinearRight(midIndex+1, highIndex, capacity)
                // find the crossing point, loop over 0<=i<=capacity
                bestSize = argMax(leftColumn[i]+rightColumn[capacity-i])
                // solve the two smaller problems
                knapDC(lowIndex, midIndex, bestSize)
                knapDC(midIndex+1, highIndex, capacity-bestSize)
        end
```

The split step where two smaller DP problems are created and solved using the 2 column technique




Once the k-1, k and k+1 columns are computed, then find the crossing point (bestSize) and whether the kth object should be used.