Kevin Harvell
CS325-400

Homework #3

1.) a.) First, we must consider a cost array which is initialized to 0. Then we loop through the stops from the second stop to n. In each iteration, we set the minimum to the cost from the first stop to i, the current iteration count. Then, in each iteration we loop from the second stop to the current iteration – 1 and check to see if the last stored cost plus the cost from stop j to i is less than our previously set min. If so, we set the min equal to the stored cost plus the cost from stop j to i. Next, we save the cost for the inner iteration to the min value. The final stored cost represents the cheapest cost from i to j.

```
PSEUDOCODE
The following was modified from:
https://people.ucsc.edu/~ptantalo/cmps201/Spring10/hw9.doc

CanoeCost(R)
    n = number of rows in R
    C[1] = 0                            # Cost initialized to 0
    for i = 2 to n
        min = R[1, i]                   # Set minimum to i stop cost
        for j = 2 to i - 1
            if C[j] + R[j, i] < min     # If the previous cost + the cost from j to i
is less than min
                min = C[j] + R[j, i]    # Set min to the cost + cost from j to i
            C[i] = min                  # Store the minimum for next iteration
    return C[n]
```

The recursive formula for filling in the array is as follows:

$$C[i] = \begin{cases} 0 & i = 0 \\ \min(C[j] + R[j,i]) & 1 < i \leq n \\ 1 \leq j < i & \end{cases}$$

b.) The sequence of trading posts follows whenever the Cost of the current post plus the cost from the current post to the i stop is less than min, push the current stop to the Print array (P). The pseudocode below uses the CanoeCost algorithm and simply creates a print array(P) in order to keep track of lowest cost sequences as the canoe progresses from post to post. These lowest cost sequences are stored in an array of arrays. The last array in the sequence represents the cheapest sequence from i to j. It then iterates through that last array to print the cheapest sequence of trading posts for the canoe.

```
PSEUDOCODE
The following was modified from:
https://people.ucsc.edu/~ptantalo/cmps201/Spring10/hw9.doc

PrintSequence(R)
    n = number of rows in R
    C[1] = 0                            # Cost initialized to 0
    P[1] = []
    for i = 2 to n
        min = R[1, i]                   # Set minimum to i stop cost
        push 1 to P[i]                  # Print 1 for 1st stop; must be visited
        for j = 2 to i - 1
            if C[j] + R[j, i] < min     # If the previous cost + the cost from j to i
is less than min
                min = C[j] + R[j, i]    # Set min to the cost + cost from j to i
                push j to P[i]          # Everytime the min is changed, print the stop
            C[i] = min                  # Store the minimum for next iteration
```

```
    for i = 1 to P[P.length].length     # Go to last array in P to print each element
        print P[P.length][i]            # Print each element in last P array
```

c.) The running time for both algorithms is $\Theta(n^2)$ because each algorithm has nested loops that iterate to n

2.) a.) The following pseudocode creates a 2D array representing weight and items to keep track of the maximum benefit in the knapsack. It first initializes a row and column to 0 to represent no items and no weight, or in other words, an empty knapsack. Then, it loops through from weight 1 to max weight to see what can be stored in the knapsack to produce the maximum benefit.

```
PSEUDOCODE
This code was taken/modified from OSU 325 lecture slides regarding Knapsack 0-1

items = []
for w = 0 to W
   B[0,w] = 0     // 0 item's
for i = 0 to n
   B[i,0] = 0  // 0 weight
   for w = 1 to W
      if wi <= w // item i can be part of the solution
         if bi + B[i-1,w-wi] > B[i-1,w]
            B[i,w] = bi + B[i-1,w- wi]
            if w == W
                items.push(i)
         else
            B[i,w] = B[i-1,w]
            if w == W
                items.push(i)
      else B[i,w] = B[i-1,w]  // wi > w item i is too big
```

b.) O(FNM) because the algorithm loops through n times with a nested loop of M(or in my case above W). Then, the above algorithm will loop F times based on the number of family members.