CSE 102
Homework Assignment 5

1. (This is a continuation of problem 6 on hw assignment 4, which was itself based on problem 8-5 on page 207 of CLRS 3rd edition.) Recall an array $A[1 \cdots n]$ is k-sorted if and only if the $(n-k+1)$ consecutive k-fold averages of $A[1 \cdots n]$ are sorted:

$$\frac{\sum_{j=i}^{i+k-1} A[j]}{k} \leq \frac{\sum_{j=i+1}^{i+k} A[j]}{k}$$

for $1 \leq i \leq n-k$. Observe that if $n = k$, then any array of length n is k-sorted, since there is only one k-fold average in this case. We may also *define* any $A[1 \cdots n]$ to be k-sorted if $n < k$, since in this case there are no k-fold averages. Modify Quicksort to produce an algorithm that k-sorts an array of any length. Write your algorithm in pseudo-code and call it $\text{Quick}[k]\text{sort}()$. Specifically, the call $\text{Quick}[k]\text{sort}(A, 1, n)$ will k-sort $A[1 \cdots n]$, but will not necessarily $(k-1)$-sort the same array. Prove the correctness of your algorithm, and analyze its worst case run time.

2. Read the Coin Changing Problem in the handout on Dynamic Programming. Its solution is presented below for reference. Recall this algorithm assumes that an unlimited supply of coins in each denomination $d = (d_1, d_2, \ldots, d_n)$ are available.

```
CoinChange(d, N)
1.  n = length[d]
2.  for i = 1 to n
3.      C[i, 0] = 0
4.  for i = 1 to n
5.      for j = 1 to N
6.          if i = 1 and j < d[1]
7.              C[1, j] = ∞
8.          else if i = 1
9.              C[1, j] = 1 + C[1, j − d[1]]
10.         else if j < d[i]
11.             C[i, j] = C[i − 1, j]
12.         else
13.             C[i, j] = min(C[i − 1, j], 1 + C[i, j − d[i]])
14. return C[n, N]
```

Write a recursive algorithm that, given the filled table $C[1 \cdots n; 0 \cdots N]$ as input, prints a sequence of $C[n, N]$ coin values whose total value is N. If it happens that $C[n, N] = \infty$, print a message to the effect that no such disbursement of coins is possible.

3. Read the Discrete Knapsack Problem in the handout on Dynamic Programming. A thief wishes to steal *n* objects having values $v_i > 0$ and weights $w_i > 0$ (for $1 \leq i \leq n$). His knapsack, which will carry the stolen goods, holds at most a total weight $W > 0$. Let $x_i = 1$ if object *i* is to be taken, and $x_i = 0$ if object *i* is not taken ($1 \leq i \leq n$). The thief's goal is to maximize the total value $\sum_{i=1}^{n} x_i v_i$ of the goods stolen, subject to the constraint $\sum_{i=1}^{n} x_i w_i \leq W$.

   a. Write pseudo-code for a dynamic programming algorithm that solves this problem. Your algorithm should take as input the value and weight arrays $v[\,]$ and $w[\,]$, and the weight limit *W*. It should generate a table $V[1 \cdots n; 0 \cdots W]$ of intermediate results. Each entry $V[i, j]$ will be the maximum value of the objects that can be stolen if the weight limit is *j*, and if we only include objects in the set $\{1, \ldots, i\}$. Your algorithm should return the maximum possible value of the goods which can be stolen from the full set of objects, i.e. the value $V[n, W]$. (Alternatively you may write your algorithm to return the whole table.)

   b. Write an algorithm that, given the filled table generated in part (a), prints out a list of exactly which objects are to be stolen.

4. **Canoe Rental Problem.**
   There are *n* trading posts numbered 1 to *n* as you travel downstream. At any trading post *i* you can rent a canoe to be returned at any of the downstream trading posts *j*, where $j \geq i$. You are given an array $R[i, j]$ defining the cost of a canoe that is picked up at post *i* and dropped off at post *j*, for i and j in the range $1 \leq i \leq j \leq n$. Assume that $R[i, i] = 0$, and that you can't take a canoe upriver (so perhaps $R[i, j] = \infty$ when $i > j$). Your problem is to determine a sequence of canoe rentals that start at post 1, end at post *n*, and which has a minimum total cost. As usual there are really two problems: determine the cost of a cheapest sequence, and determine the sequence itself.

   Design a dynamic programming algorithm for this problem. First, define a 1-dimensional table $C[1 \cdots n]$, where $C[i]$ is the cost of an optimal (i.e. cheapest) sequence of canoe rentals that starting at post 1 and ending at post i. Show that this problem, with subproblems defined in this manner, satisfies the principle of optimality, i.e. state and prove a theorem that establishes the necessary optimal substructure. Second, write a recurrence formula that characterizes $C[i]$ in terms of earlier table entries. Third, write an iterative algorithm that fills in the above table. Fourth, alter your algorithm slightly so as to build a parallel array $P[1 \cdots n]$ such that $P[i]$ is the trading post preceding i along an optimal sequence from 1 to i. In other words, the last canoe to be rented in an optimal sequence from 1 to i was picked up at post $P[i]$. Write a recursive algorithm that, given the filled table *P*, prints out the optimal sequence itself. Determine the asymptotic runtimes of your algorithms.

5. **Moving on a checkerboard** (This is problem 15-6 on page 368 of the 2nd edition of CLRS.) Suppose that you are given an $n \times n$ checkerboard and a single checker. You must move the checker from the bottom (1^{st}) row of the board to the top (n^{th}) row of the board according to the following rule. At each step you may move the checker to one of three squares:

   - the square immediately above,
   - the square one up and one to the left (unless the checker is already in the leftmost column),
   - the square one up and one right (unless the checker is already in the rightmost column).

   Each time you move from square *x* to square *y*, you receive $p(x, y)$ dollars. The values $p(x, y)$ are known for all pairs (x, y) for which a move from *x* to *y* is legal. Note that $p(x, y)$ may be negative for some (x, y).

   Give an algorithm that determines a set of moves starting at the bottom row, and ending at the top row, and which gathers as many dollars as possible. Your algorithm is free to pick any square along the bottom row as a starting point, and any square along the top row as a destination in order to maximize the amount of money collected. Determine the runtime of your algorithm.