COSC 302: Analysis of Algorithms — Spring 2018
Prof. Darren Strash
Colgate University

Problem Set 8 — Greedy Algorithms and Dynamic Programming I
Due by 4:30pm Friday, April 6, 2018 as a single pdf via Moodle (either generated via LaTeX, or concatenated photos of your work). Late assignments are not accepted.

This is an *individual* assignment: collaboration (such as discussing problems and brainstorming ideas for solving them) on this assignment is highly encouraged, but the work you submit must be your own. Give information only as a tutor would: ask questions so that your classmate is able to figure out the answer for themselves. It is unacceptable to share any artifacts, such as code and/or write-ups for this assignment. If you work with someone in close collaboration, you must mention your collaborator on your assignment.

Suggested practice problems, from CLRS: 15.1-4, 15.1-5; 15.3-3; 15.3-5; 16.2-1; 16.2-4; 16.2-5

1. **(Worth 10 points)** Problem 16-1 from CLRS.

   (a) We suppose an algorithm that stores the coin values in an array in decreasing order. Then we iterate through the array and which each iteration, we divide the input by the coin value floored. And then add the result to another array called count, which stores the frequency of each coin. Such that quarter's frequency would be first followed by dimes then nickles and finally pennies. Usiing larger coins first will always lead to using less coins since smaller coins can be used to represent bigger coins. Suppose there existed a correct coin count that was equal to the amount we had found. Note that for coin values higher than 24 cents our algorithm will use a quarter. Using anything smaller would yield a count that's higher. For cent values between 24 and 10 inclusive we would use dimes, we could not use quarters because the value isn't high enough and we would not use a combination of nickles nor pennies since that would yield a higher count. The same goes for nickles which we would use for values lesser than 10 but higher than 4. Using a combination of pennies would always yield a higher coin count. Then finally we use pennies for the rest. This shows that we can consider the optimal solution and our own both the same.

   (b) The algoritm that makes change does so by finding c^i which is the denomination which is the greatest denomnation of the set which $c^i \leq n$ holds. We increment the count then do a recursive call on the subproblem $n - c^i$. Note that since we find the maximum every time which the inequality holds, we can use denominations that are less than c^i to make the same value but never with the same amount or less coins since it's the maximum.

   (c) If we were given a set of denominations such that $D = 1, 5, 7$. And the cent value is 10. A greedy algorithm would choose 7 then three 1 cent coins. When the optimal solution would be two 5 cent coins which is better than our greedy algorithm.

   (d) Let N be coin value. Let D an array of denominations in decreasing order. Let T be a two dimensional array. With one axis that represents the counts of each denomination and the other axis representing the coin value of all the denominations in the other axis. Not that this table is intialized so that each element has the value of ∞.

2. Problem 15.1-2 from CLRS. Suppose we have a rod of length 4. And three types of cuts which can be done on the rod. Rods of length 1 have a density of 1. Rods of length 2 have a

```
function MakeChange($N, D, T$)
    if $T[N][D] = \infty$ then
        $c = 1$
        for $c \leq D.length$ do
            if $N \neq 0$ and $c \leq N$ then
                if MakeChange($N - c, D, T_1$) $<$ MakeChange($N, D - c, T_1$) then
                    $T[c] = T[c] + 1$
                    return 1+MakeChange($N - c, D, T_1$)
                else
                    return MakeChange($N, D - c, T_1$)
            else
                $T[N][D] = 0$
                return 0
    else
        return $T[N][D]$
```

density of 1.50 and rods with a density of 3 have a density of 1.75. The greedy described in the problem would choose to first make a cut of length 3 then length 1 which has a value of 2.75 but this is not optimal since two cuts of length 2 would give a value 3.00 which is greater than the greedy algorithm.

3. Problem 15.1-3 from CLRS.

```
function MakeChange($p, n$)
    for $j \leftarrow 1$ to $n$ do
        $maxprofit = 0$
        for $k \leftarrow 1$ to $j$ do
            if $maxprofit < P_i + R[i = l_i] - c$ and $j - l_i \geq 0$ then
                $maxprofit = P_i + R[i = l_i] - c$
        $R[j] = maxprofit$
```