a. Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. prove that your algorithm yields an optimal solution.

Solution:

If $n < 5$, use n pennies.

If $5 \leq n < 10$, use 1 nickel and $n - 5$ pennies

If $10 \leq n < 25$, use $n/10$ dimes, and then the rest will be either pennies or nickels.

If $25 \leq n$, use $[n/25]$ quarters, and then one of the 3 previous cases.

What the greedy algorithm does is to always use the greatest value coins for the existing amount and to use as many of those coins as possible without exceeding the existing amount. After deducting this sum from the existing amount, the remainder is used as the new existing amount and repeat the process.

To prove that this algorithm yields an optimal solution, firstly make sure that the greedy-choice property holds, i.e to get an optimal solution which is making change for n cents which contains c, where c is the largest coin (c<=n). If c is equal to n cents, then c will be the solution and the computations stop to look for other coins. Otherwise, the optimal solution doesn't include a coin of value c.

- If $1 \leq n < 5$, then $c = 1$. A solution may consist only of pennies, and so it must contain the greedy choice.

- If $5 \leq n < 10$, then $c = 5$. A solution may consist of a nickel and the rest will be pennies. In case there were just pennies, then the number of coins will increase thus making it less optimal.

- If $10 \leq n < 25$, then $c = 10$. By supposition, this optimal solution does not contain a dime, and so it contains only nickels and pennies. Some subset of the nickels and pennies in this solution adds up to 10 cents, and so we can replace these nickels and pennies by a dime to give a solution with fewer coins.

- If $25 \leq n$, then $c = 25$. An optimal solution consists of a quarter. But in case there are online dimes, nickels and pennies that adds up to 25 cents, then it can be replaced by a quarter to reduce the number of coins thus making it optimal.

b. Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are c0, c1, ..., ck for some integers c>1 and k>= 1. Show that the greedy algorithm always yields an optimal solution.

Solution:

Any optimal solution must use at most $c-1$ coins of denomination cj for $j \leq k$, because otherwise we could combine them to create a solution using fewer coins. Thus, the optimal solution to making change for any amount always uses at least one of the largest coin possible.

Let's consider the first two types of coins in this set, pennies and coins that are worth c cents each, which we will call c. At most, we can use $c - 1$ pennies because any number larger than c pennies would be replaced by at least one c coin. This operation would reduce the total coin number by $c - 1$. In other words, when the remainder is greater than c and if only pennies and c coins are to be used, then use as many c coins before considering pennies as that would reduce the total number of coins.

c. (Optional) Give an O(nk)-time algorithm that makes change for any set of k different coin denominations, assuming that one of the coins is a penny.

Solution:

The greedy algorithm takes O(nk) for any kind of coin set denomination, where k is the number of different coins in a particular set.

```
#include <iostream>

using namespace std;
int count;
int makeChange(int amount, int coins[4]) {
     const char* denomination[4] = {"penny","nickel","dime","quarter"};
     if (amount == 0) {
            return 0;
     }
    for (int i = 4; i > 0; i--) {
        int coin = coins[i-1];
        //If the next largest coin is found, print out its value.
        if (amount == coin || amount >= coin) {
                printf("coin: %i %s \n",coin,denomination[i-1]);
        count++;
     return 1 + makeChange(amount - coin, coins);
    }
 }
 printf("Cannot make change ");
 printf("cents remaining: %i", amount);
 return 0;
 }

int main()
{
    int coins[4] = {1 ,5 ,10 ,25};
    int amount = 75;
    printf("Amount: %i \n",amount);
    makeChange(amount ,coins);
    printf("Total number of coins: %i", count);
    return 0;
}
```

OUTPUT:

For amount 99:

```
Amount: 99
coin: 25 quarter
coin: 25 quarter
coin: 25 quarter
coin: 10 dime
coin: 10 dime
coin: 1 penny
coin: 1 penny
coin: 1 penny
coin: 1 penny
Total number of coins: 9
```

For amount 75:

```
Amount: 75
coin: 25 quarter
coin: 25 quarter
coin: 25 quarter
Total number of coins: 3
```