1 Coin changing

a.

(1 quarter = 25 cents; 1 dime = 10 cents; 1 nickel = 5 cents; 1 penny = 1 cent)

Pseudocode:

```
_quarter = n / 25
n = n – 25 * _quarter
_dime = n / 10
n = n – 10 * _dime
_nickel = n / 5
n = n – 5 * nickel
_penny = n
```

Prove:

Assume the solution we have obtained above is

$$n = 25 * \_quarter + 10 * \_dime + 5 * \_nickel + \_penny \quad (*)$$

and

$$k = \_quarter + \_dime + \_nickel + \_penny.$$

Make an assumption that the (*) is not the optimal solution, the optimal solution is

$$n = 25 * \_quarter' + 10 * \_dime' + 5 * \_nickel' + \_penny'$$

and

$$k - 1 = \_quarter' + \_dime' + \_nickel' + \_penny'.$$

So there are 4 cases:

1._quarter' = _quarter - 1

To compensate the 25 cents and keep the number of coins minimum, obviously we should use dimes to compensate the 25 cents.

So we can easily obtain:

$$\_dime' = \_dime + 5$$
$$\_nickel' = \_nickel$$
$$\_penny' = \_penny$$

and

$$\_quarter' + \_dime' + \_nickel' + \_penny' = k + 4 > k$$

So it is not an optimal solution.

2._dime' = _dime - 1

Similar to 1, we can obtain:

$$\_quarter' = \_quarter$$
$$\_nickel' = \_nickel + 2$$
$$\_penny' = \_penny$$
$$\_quarter' + \_dime' + \_nickel' + \_penny' = k + 1 > k$$

So it is not an optimal solution.

3._nickel' = _nickel - 1
Similar to 1, we can obtain that it is not an optimal solution.

4. _penny' = _penny – 1
Similar to 1, we can obtain that it is not an optimal solution.

To sum up, the assumption is false. Similarly, all other possible assumptions are false either.
Therefore, (*) is the optimal solution.

b.
Set a_i be number of coins of denomination c^i.
We use greedy algorithm to get a solution:

$$n = \sum_{i=0}^{k} a_i c^i$$

If we deduct the number of c^{j+1} from a_j to a_j - 1, to compensate the c^{j+1} cents and keep the number of coins minimum, obviously we should use c coins of denomination c^j to compensate it. The total number of coins will increase by c-1. That means we can't find an optimal solution besides greedy solution.
Therefore, the greedy algorithm always yields an optimal solution.

c.
Set the coins of denomination are 1, 3 and 4. When n = 6, the greedy solution is one 4 coin and two 1 coins. But two 3 coins is better.

2 Genetic Algorithm

See in GA.cpp and README.pdf.

3 Bonus

Pseudocode:

```
Edit_distance (x, m, y, n, cost)
  //cost = { 0, 1, 2, 3, 4, 5, 6} means{none, copy, replace, delete, insert, twiddle, kill}
{
    Create 2 new 2D arrays c[m+1][n+1] and op[m+1][n+1]
    Create a new array d[7]
    c[0][0] = 0
    op[0][0] = 0
    for j =1 to n
        c[0][j] = j * cost [4]   //insert
        op[0][j] = 4
    for i = 1 to m
        c[i][0] = i * cost[3]    //delete
        op[i][0] = 3
    if n = 0 and cost[6] < c[m][0]
        c[m][0] = cost[6]        //kill
        op[i][0] = 6
        p = 0
    for i = 1 to m
        for j = 1 to n
            for k = 1 to 6
                d[k] =  ∞
            if x[i] = y[j]
                d[1] = cost[1] + c[i-1][j-1]      //copy
            else
                d[2] = cost[2] + c[i-1][j-1]      //replace
            d[3] = cost[3] + c[i-1][j-1]          //delete
            d[4] = cost[4] + c[i-1][j-1]          //insert
            if i  ⩾  2 and j  ⩾  2 and x[i-1] = y[j] and x[i] = y[j-1]
                d[5] = cost[5] + c[i-1][j-1]      //twiddle
            if i = m and j = n
                for k = 0 to m-1
                    if cost[6] + c[k][n] < d[6]
                        d[6] = cost[6] + c[k][n]    //kill
                        p = k
            c[i][j] =  ∞
            for k = 1 to 6
                if d[k] < c[i][j]
                    c[i][j] = d[k]
                    op[i][j] = k
    return c, op and p
}
```

```
/*
c is the array of optimal cost during transition.
op is the array of the operation of the final step during transition.
If the final step is kill, p is the number of chars that are solved.
*/

Print_op (op, p, i, j)
{
        if op[i][j] = 0
            return
        else if op[i][j] = 1
            Print_op (op, p, i-1, j-1)
            print " copy i -> j "
        else if op[i][j] = 2
            Print_op (op, p, i-1, j-1)
            print " replace i <-> j "
        else if op[i][j] = 3
            Print_op (op, p, i-1, j)
            print " delete i "
        else if op[i][j] = 4
            Print_op (op, p, i, j-1)
            print " insert j "
        else if op[i][j] = 5
            Print_op (op, p, i-2, j-2)
            print " twiddle i-1 -> j    i -> j-1 "
        else
            Print_op (op, p, p, j)
            print " kill p "
}
```

The time complexity and space complexity are both O(mn).