I used this website to help me in this problem set:
http://stackoverflow.com/questions/2633848/dynamic-programming-coin-change-decision-problem

(a) **Recursive structure**
The solution for N depends on solutions for $N-N=0$, $N-3$, $N-10$, and $N-25$ (if they exist, i.e if $N=3$ then there is no solution for $N-10$ or $N-25$ since $N \geq 0$)
If we run our algorithm on an input **N** that **should** return **True**, that means that **N** is the **exact** sum of a certain number of 3's, a certain number of 10's and a certain number of 25's (a, b, c are integers greater than or equal to 0) **N = 3a + 10b + 25c**
That also means that either 3, 10 or 25 can be subtracted from **N**. If we subtract either 3, 10, or 25, we know that **at least one** of the differences **D** where **D** can be $\{0, N-3, N-10, N-25$ such that $\mathbf{D} \geq 0\}$ must be an **exact** sum of a certain number of 3's, a certain number of 10's and a certain number of 25's in order for the algorithm to return **True** on **N** (i.e running the algorithm on **D** should return **True**). This is where we can check if the output of running our algorithm on **D** will return **True**. If it does return **True** then you should be able to buy exactly **N** nuggets at this restaurant, if not, then you should **not** be able to buy exactly **N** nuggets at this restaurant. This is because if you currently have a sum that is not exactly composed of a certain number of 3's, a certain number of 10's and a certain number of 25's, then adding 3, 10, or 25 will not change the fact that it's not an exact sum of 3's, 10's and 25's.

(b) We define an array OPT of size $N+1$
$OPT[i]$, $i = 0, 1, 2, ..., N$
where $OPT[i]$ contains a boolean value indicating whether or not you can purchase exactly i Tofu Nuggets at this restaurant
In other words, $OPT[i]$ stores the output of running the algorithm on input i

(c) **Base case:** $OPT[0]$ = True
I can purchase exactly 0 nuggets by purchasing exactly 0 boxes of 3, 10, or 25 nuggets
General case: for $i = 1 ... N$
$OPT[i]$ = **False** **unless** $i \geq j$ and $OPT[i-j]$ = True for j = any value in the set {3, 10, 25}
If this is the case **then** $OPT[i]$ = **True**
We know from the structure described in part **(a)** that the algorithm must return **True** for a number **D** where **D** can be $\{0, N-3, N-10, N-25$ such that $\mathbf{D} \geq 0\}$ in order for the algorithm to return **True** for **N**. This is where the intuition for my recurrence relation came from.

(d) Dynamic(N)

```
1 OPT[0] = True
2   for i = 1 .. N
3       OPT[i] = False
4       for j = (3, 10, 25)
5           if i ≥ j and OPT[i − j]       //If i < j, then the first condition will be false and should
6               OPT[i] = True             //exit the if statement, in other words, we shouldn't get a negative
7   return OPT[N]                         //index for OPT for the second condition check
```

RunTime: $\Theta(n)$
Yes, it runs in polynomial time since:
for loop at line 2 runs N times
Nested **for** loop at line 4 runs 3 times
RunTime: $\Theta(3n) = \Theta(n)$