Ramaneek Gill 1000005754
CSC373 Problem Set 3

a)
recursive structure:
return_value = 3a+10b+25c where a,b,c are natural numbers. Then there are three cases:
c > 0:
return_value = 3a+10b+25(c-1) + 25
OR b > 0:
Return_value = 3a + 10(b-1) + 10 + 25c
OR a > 0:
Return_value = 3(a-1) + 3 + 10b + 25c

#this is just showing that each 'equation' of people can be calculated recursively in a multiple of 3, 10, or 25.

b)
each array value will store a boolean on whether or not x amount of nuggets can be purchased for up to N people:
for x = -24, -23, -22,…, 0, 1, 2, …, N; A[x] = true or false.

c)
Observations:
For any x < 0; A[x] = FALSE
A[0] = TRUE #because 0 people implies 0 nuggets needed which is trivial case
#by simple boolean logic:
A[x] = A[x-3] OR A[x-10] OR A[x-25]   #try the mental math in your head to see it makes sense
This makes sense because if the number of people minus 3 OR 10 OR 25 can equally be bought exactly 1 nugget each then any multiple of another 3 OR 10 OR 25 people can equally get 1 nugget each.

d)
```
for x in range(-24, 0):   #python function 0 isn't included but -1 is
        A[x] = FALSE   #initializing trivial values
A[0] = TRUE #base case from c)
For x in range(1,N+1):
        A[x] = A[x-3] OR A[x-10] OR A[x-25]  #boolean ORs, also from c)
Return A[N] #true if N people can be bought exactly 1 nugget each in multiples of 3 or 10 or 25
```

Correctness:

Recursive proof is trivial refer to a, b and c and follow the algorithm to see it is correct. Pitt said the same thing in his lectures.

Polynomial Time:

Yes it does run in polynomial time because each iteration calls a growing constant number of recursive calls each until they reach the base cases (A[-24] all the way to and including A[0]), so A[x-3] gets called recursively in *one current* iteration at most O(3*(x-3 repeated recursively a constant amount of times until x-3 <= 0)). The 3 times is in that big-O notation because it is a very loose upper bound since A[x-3] can get called in an A[x-10] and A[x-25] less times than in an A[x-3] recursive call.