Jonathan Tso
CS 401

1. Consider that a palindrome is like A, ACA, ACGCA, ... Then by looking at a subsequence, we know that our smallest is 1 characters and expands where the left and right are always the same character

1. Look at the string S from beginning character
2. Look at the subsequence S' from beginning character
3. Look at the ends of S', so if S' is from 1 to n, we look at 1 and n. If they are the same, we look at (1+1, n-1) and continue to check. We find the longest palindrome of this, with this method.
4. Repeat steps 2 and 3 until we reach the point where the character(s) we look at are the centers.
5. Repeat 1 to 4 after moving string S', beginning to i+1 (so string S is from (i, n))

This is basically a double for loop, so the run time is n^2 since we run through S' and S and S' is strictly $\leq S$.

2 a) 1, 10, 5, 8

In the above instance, we would get from this algo a max rising trend of 2. Later we are expecting 3 (1,5,8)

b) Run the same algorithm as the original, but include locations at each of the subsequent numbers from 2 to n

```
i = L = 1        K = 2
for K to n do
    for K to n do
        if P(j) > P(i) then
            L = L + 1
            run algorithm from (K, n) and return size
            ↑ for this, base case is it's singleton integer, it returns 1
        return size of longest trend
    end
```

3. For value V

Look at largest coin denomination, if V - coin > 0 then

rerun algorithm on V - coin and return T/F

If not, look at next largest denomination, and so forth and rerun

Base case 1 is for smallest coin denomination

if V - smallest coin > 0 then return True

else return false

Base case 2 is if V - any coin denomination == 0 return True

2)

4. Let's say that A[1...n] is ~~[illegible]~~

function func(A[n]) { int[]array maxVal

a) For int i=0; i<n; i++
   for int j=0; j<n; j++
   find the max value between the array of size A[i...j] and func(A[i...n])

b) Have the above return and add into the maxVal array the pathway or individual values that led to max value

The reason the above will work is it is effectively a double for loop, going through each iteration. However, it will remember the longest length and array in maxVal, so if we wanted to optimize this, we could potentially do a check on the next array's total val, and if maxVal's total > that total val, we can just ignore that array check.

5. For each node, have it recognize 3 values

1 its own min cost 3 num of minRuns

2 its best predecessor

Let us start with u, where cost = ∞ and 2 is itself

a) From u, look outward and find, for each connected node, the ~~shortest~~ costs of each, and update predecessor accordingly.

b) Repeat a but ~~for~~ each node onward from the newly updated nodes. Do this to fill the graph.

c) If we see that the minimum cost is already lower than if you were to touch it, no longer follow that path. If we see that by touching it, it is lower, update. If they are equal, add to num of minRuns.

Here, we will converge to v from u with the smallest possible paths. We also add to minRuns, which will result in our answer when looking at w.