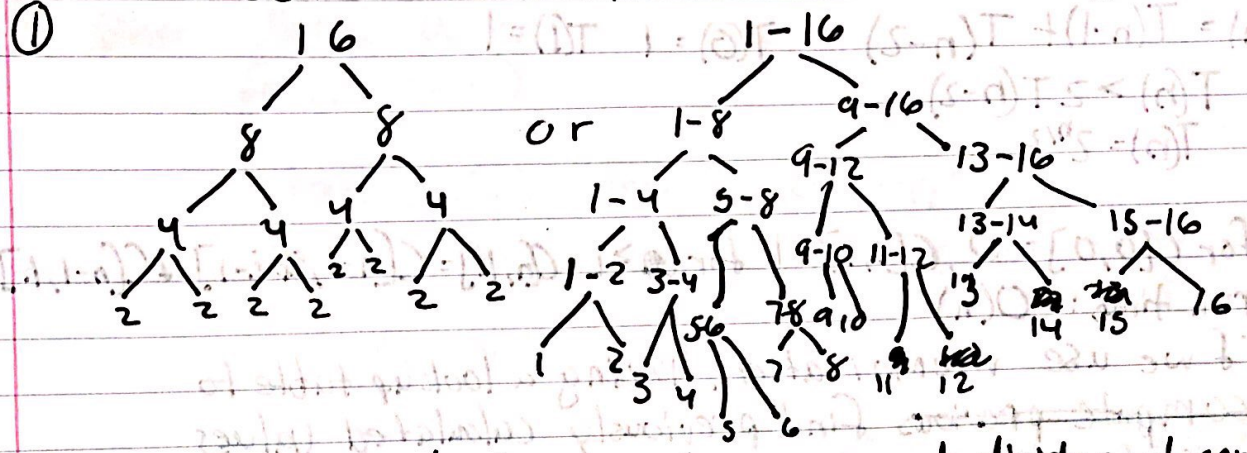
Jerome Schmidt 70497605 written HW#4

① 

memoization is not for speeding up a good divide and conquer because there is no overlapping.

② 8.1.3

a.) $T(n) = T(n-1) + T(n-2) + O(1)$

$T(n-1) = T(n-2) + T(n-3) + O(1)$

$T(n-2) = T(n-3) + T(n-4) + O(1)$

$T(n) = T(n-2) + T(n-3) + T(n-3) + T(n-4) + O(1) + O(1) + O(1)$

$T(n) = T(n-2) + 2(T(n-4) + T(n-5) + O(1)) + T(n-5) + T(n-6) + O(1) + O(1) + O(1) + \ldots$

so $T(n) = 2^k T(n-k)$

so $O(2^n)$

b.) Since for n coins, there are $\frac{C(2n,n)}{n+1}$ combinations of possible flips, and since combinatoral functions grow exponentially, the efficiency is at least exponential.

(3) a.) $T(n) = T(n-1) + T(n-2)$ $T(0) = 1$ $T(1) = 1$

$T(n) > 2T(n-2)$

$T(n) = 2^{n/2}$

b.) for $C[0,0] = 0$, $C[1,1] = 1$ for $i > 1$ $C[n,k] = C[n-1, i-1] + C[n-1,k]$

run time: $O(n)$

if we use memoization, having a lookup table to ~~compute previous~~ find previously calculated values significantly improves compute time.

(4) Input: array char X[], array char Y[]

```
int i = size of X[]
int j = size of Y[]
int k = 0 // placeholder for largest
for m = 0 → i {
    for n = 0 → j {
        if(m=0 or n=0) {
            temp[m][n] = 0
        }
        else if(X[m-1] = Y[n-1])
        {
            temp[m][n] = temp[m-1][n-1] + 1
            k = largest of k and temp[m][n]
        }
        else{
            temp[m][n] = 0
        }
    }
}
output k
```

⑤ Answer

```
Array[0][0] = 0
for(i=1 → n)
{
  Array[i][0] = Array[i-1,0] (1-p_i)
  for(j=1 → n)
  {
    for(k=1 to n)
    {
      Array[j][k] = Array[j-1][k](1-p_i) + Array[j-1][k-1] p_i
    }
  }
}
```