Last Class: Divide + Conquer

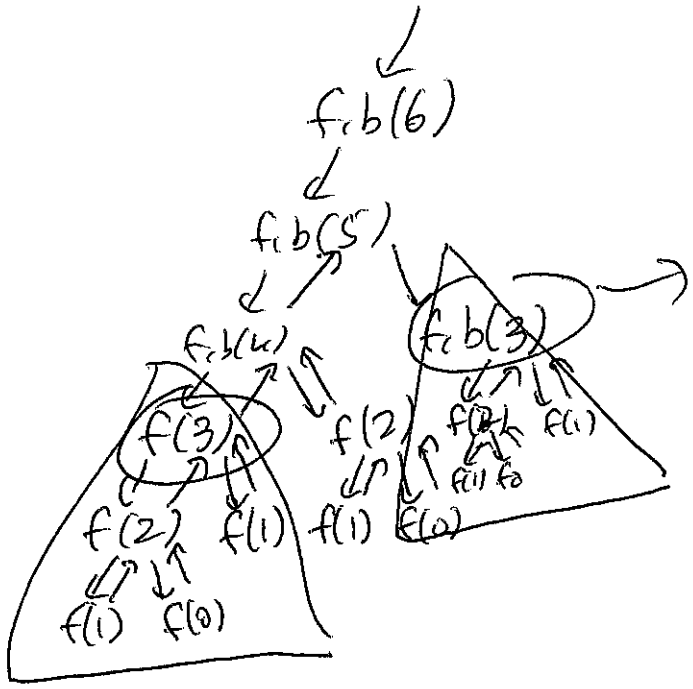
Int Mult
Tromino Tiling
LCS

Skipped
- Skyline (read this)
- Closest Pts (Pair) } not necessary
- Strassen's

really similar to Merge Sort

Dynamic Programming!

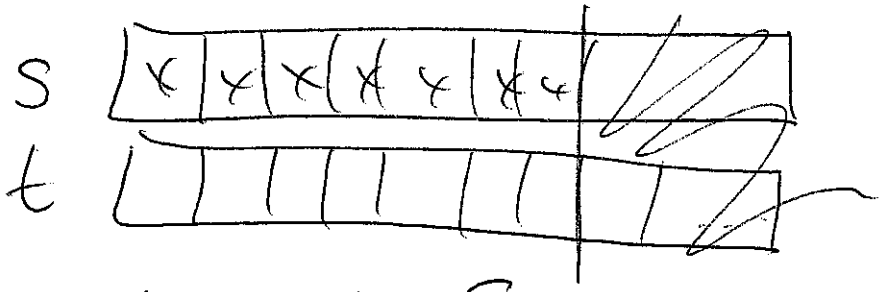
```
if (n < 2) return n;
return fib(n-1) + fib(n-2)
```



If the ant had better memory, it wouldn't have to recalculate f(3)!!!

KEY IDEA: ADD Memory (Array, HashMap) to store answers to possible recursive calls so you don't have to actually redo the WORK!

LCS (Longest Common Subsequence)




```
LCS (s,t) {
    if (last chars match)
        return 1+LCS(s-1char, t-1char)
    else
        return max(LCS(s, t-1char),
                   LCS(s-1char, t))
}
```

All recursive calls are to s[0..i] for all i, t[0..j] for all j

ALL PREFIXES of s can be the 1st param
ALL PREFIXES of t can be the 2nd param

```
// return the LCS of s[0...i], t[0..j]
LCS(int i, int j) {
    if (i<0 || j<0) return 0;
    if (s[i] == t[j])
        return 1+LCS(i-1, j-1);
    else
        return max(LCS(i, j-1),
                   LCS(i-1, j));
}
```

(1) Memoization - a slight edit to a slow recursive solution so that no recursive call gets made twice. (Top Down)

(2) Dynamic Programming - identify all possible recursive calls I might need to make, figure out the dependency btw those calls. and solve your cases in a bottom up order so that there's no recursion in the code. (Bottom Up)

(1) Fib
(2) LCS

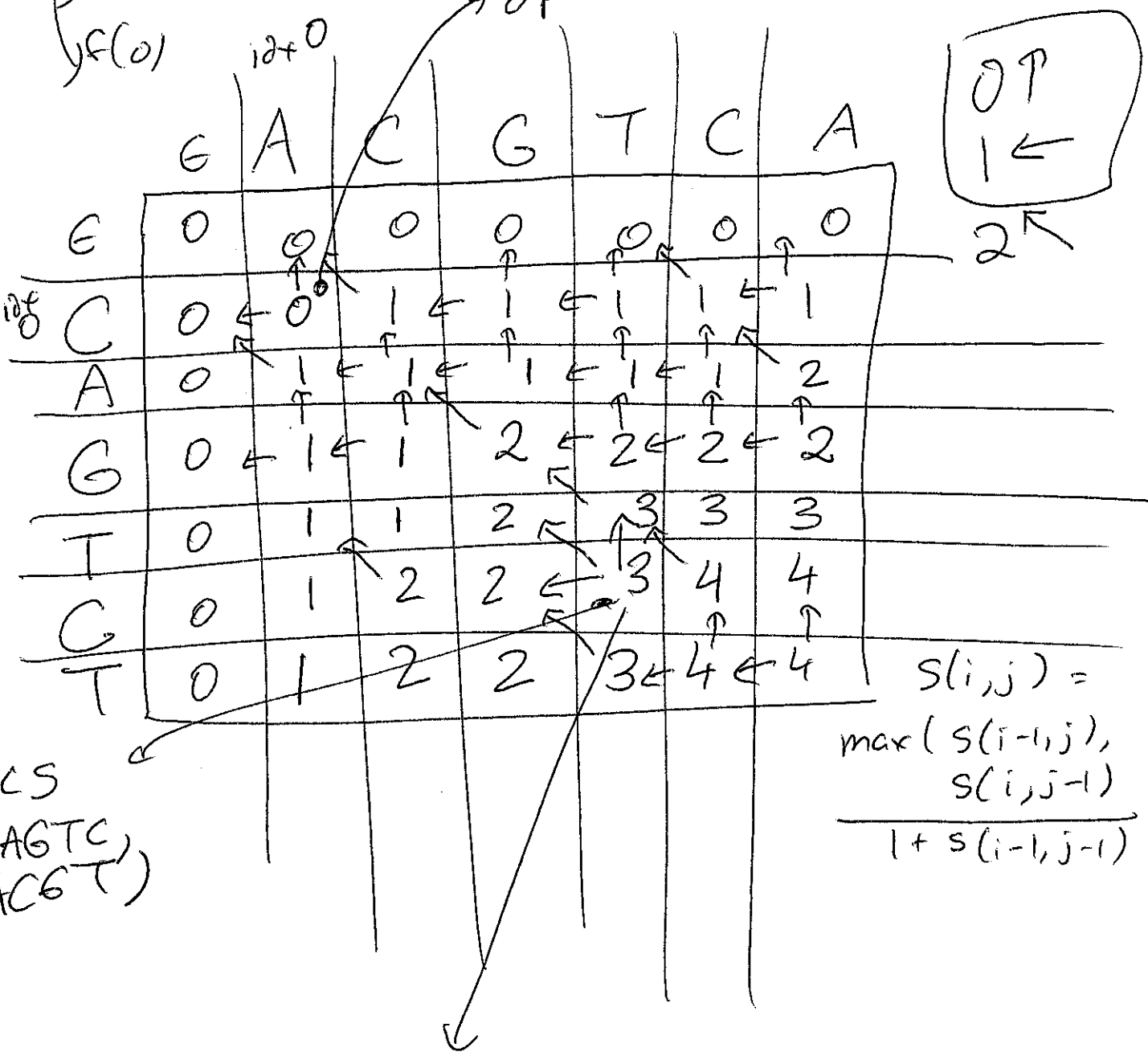
To MEMOIZE

1. Declare memory to store results of all rec calls.
2. Initial memory -1 (any sentinel) (unsolved)
3. In base case, check if we've done this call before
4. Before returning ans, store in memory.

f(5) Dependency chart

f(4) f(0) } Do manually
f(3) f(1)
f(2) f(2) ⟶ f(n)
f(1) no recursion is needed
f(0)

dp[1][1]



LCS (CAGTC, ACGT)

$$S(i,j) = \max\left(S(i-1,j),\ S(i,j-1),\ 1 + S(i-1,j-1)\right)$$