Recitation 13 (11/28 - 12/2)

- Remind them about the 480s timeout for HW 9

- Recap last week's recitation, answer any questions people may have: http://www-student.cse.buffalo.edu/~atri/cse331/fall16/recitations/Recitation12.pdf

- Dynamic Programming (source: Zulkar's recitation notes from last year, just typed out)
  - Divide problem into subproblems
  - Solve each subproblem
  - Combine to get original solution
  - This should look a lot like divide and conquer!
  - Differences below

Divide and Conquer	**Dynamic Programming**
Independent subproblems	Overlapping subproblems
Solve top down	Solve bottom up (or top down using memoization)
Usually recursive	Backtracking

Ex: Divide and Conquer vs. Dynamic Programming Solutions (same problem)

Fibonacci Series - 1 1 2 3 5 8 13

def: fib[n] = fib[n-1] + fib[n-2]

Assume fib[0] = 0 and fib[1] = 1
This is all you know and you want to get fib[5]

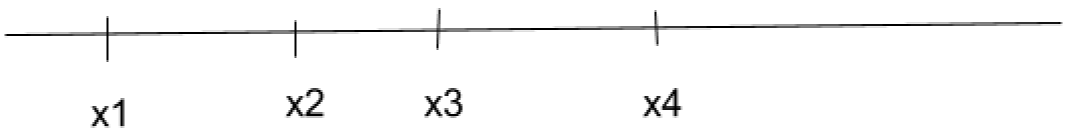
fib[5] = fib[4] + fib[3]
    = fib[3] + fib[2]
    = fib[2] + fib[1]

The above has overlapping subproblems. Better solution to the problem
→ Solve smaller subproblems first, store them and use them to solve the larger subproblems

fib(n):
    fib[0] = 0
    fib[1] = 1

```
for i in range(2,n):
        fib[i] = f[i-1] + f[i-2]
return fib[n]
```

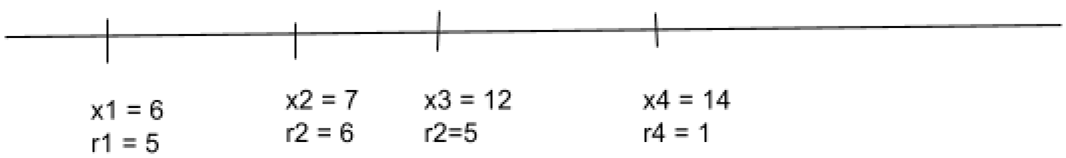
Billboard Problem:



sites for a billboard, each at a location x_i and each site generates r_i revenue

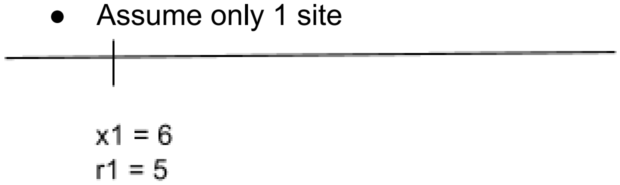
Goal: Maximize revenue with the constraint: $\min(\text{dist}(x_i, x_{i+1})) > 5$

Example:



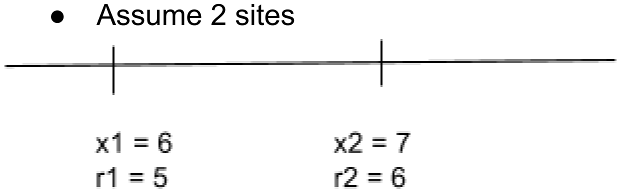
Bottom-Up (Dynamic Programming Style)

- Assume only 1 site



Optimal solution: opt(1) = 5

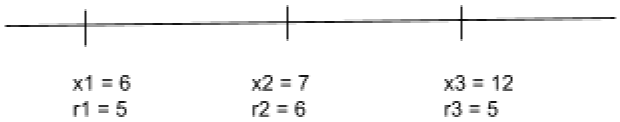
- Assume 2 sites



  - Don't pick x_2 - optimal solution is the same as above = 5
  - Pick x_2 - eliminate all sites within a 5 mile radius of x_2

opt(2) = max(opt(1), r_2) = max{5,6} = 6

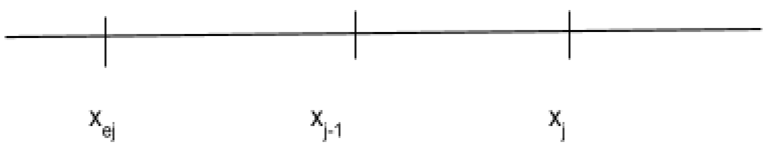
- Assume 3 sites - 2 options



    - Don't pick x_3: opt(2)
    - Pick x_3: opt(x1) + r3

max{opt(2), opt(x1) + r3} = {6, 5+5} = {6,10} = 10

Generic Formula (try to get students to come up with this for you)



$opt(x_j) = \max\{opt(x_{j-1}), opt(x_{ej})+r_j\}$ (where e_j is outside of the 5 mile radius)

Algorithm:
M[ ] ← stores value of opt from prev eq
M[0] = 0
M[1] = r_1
for j = 2...n:
    compute M[j] based on generic
return M[n] → will be the largest revenue

Runtime Analysis - O(n)

Look up for x_{ej} can be done in constant time
    Keep two lists - $x_i \ldots x_n$ & $x_i' .. x_n'$ where $x_i' = x_i - 5$
    Now we can merge the two lists O(n)
    Look for current element x_j' in list O(n) and x_{ej} is to the left of it O(1)
    Can preprocess and create a dict that contains the x_{ej} value for each x_j