CSE 331: recitation 13:

Dynamic programming:

⇒ Divide problem into small subproblem.
⇒ solve each subproblem.
⇒ Combine to get original solution.

looks familier? Divide & Conquer.

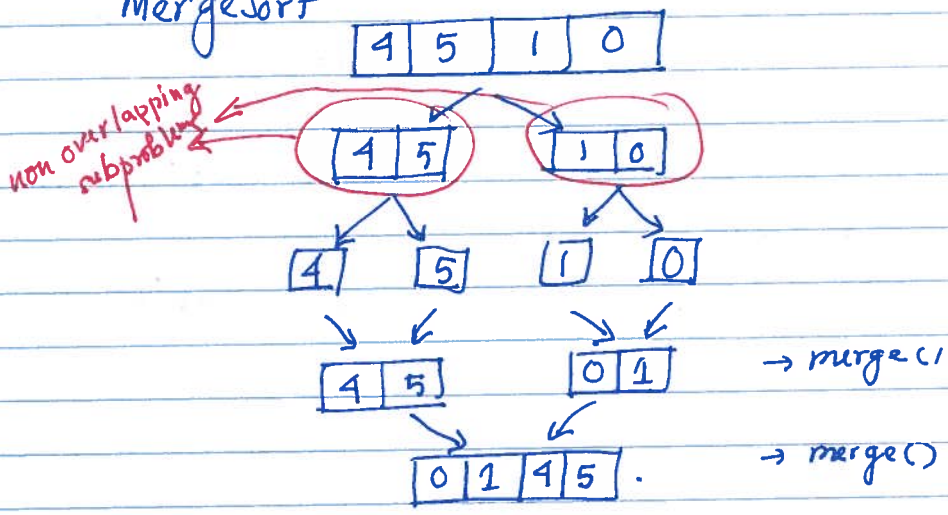
Divide And Conquer:

(1) independent subproblems.
(2) solve Top-down
(3) Usually recursive calls.

Dynamic Programming:

(1) Overlapping subproblems
(2) solve bottom-up
(3) Backtracking.

MergeSort



Fibonacci series – 1 1 2 3 5 8 13

$fib[n] = fib[n-1] + fib[n-2]$.

assume, $fib[0] = 0$

$fib[1] = 1$.

$Fib[5] = (Fib[4]) + (Fib[3])$

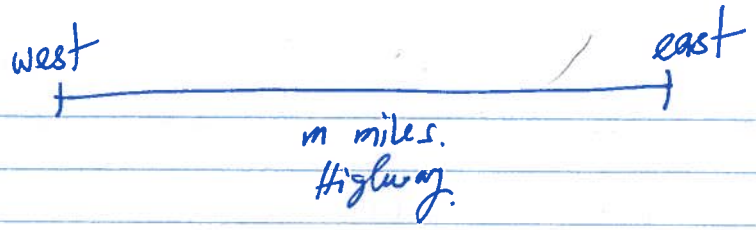
$= (Fib[3] + Fib[2]) + (Fib[2] + Fib[1])$

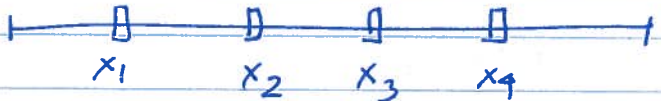
overlapping subproblems.

better strategy: solve smallest subproblems first. (store them)

then build largeso subproblem using small ones.

```
fibo (n):
    fib[0] = 0
    fib[1] = 1
    for i = 2 ... n:
        fib[i] = fib[i-1] + fib[i-2].
    return fib[n].
```





possible sites for billboard.

each site x_i generates r_i revenue if you place billboard there.

Constraint: $\min(\text{dist}(x_i, x_{i+1})) > 5$.

Goal: maximize revenue.

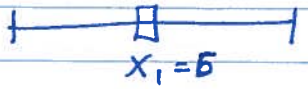
Example:

optimal soln?

place billboard in x_1, x_2 : revenue : 10
$r_1=5$, $r_2=5$.

Lets do it in bottom-up fashion:

(1) assume only one site



: optimal result: 5
OPT(1) = 5

(2) assume two sites

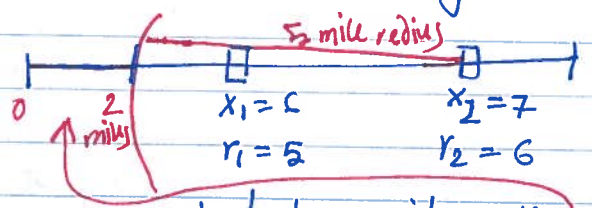
two options: either pick x_2 / not.

option 1:

if you don't pick x_2, result is ~~optimal~~ equal to optimal result of (1).

$\boxed{OPT(1).}$

option 2: if you pick x_2, eliminate all sites within 5 mile redius of x_2.

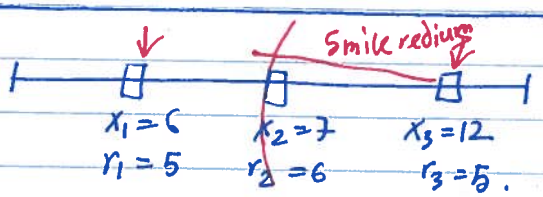


we don't have sites there.

$\boxed{r_2}$

so $OPT(2) = \max\{ OPT(1), r_2 \} = \{5, 6\} = 6.$

(iii)



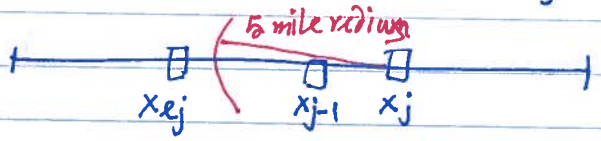
option 1: Don't pick x_3 : $\boxed{OPT(2)}$

option 2: pick x_3 : $\boxed{OPT(x_1) + r_3}$

$$OPT(3) = \max\{ OPT(2), (OPT(x_1) + r_3) \} =$$

$$= \max\{ 6, \quad 5 + 5 \} = 10.$$

Generic formula:



$$OPT(x_j) = \max\{ OPT(x_{j-1}), OPT(x_{ej}) + r_j \}$$

Algorithm:

M[ ] ← stores value of OPT from previous equation.

M[0] = 0

M[1] = r_1

for j = 2, ... n:

compute M[j] based on Generic equation.

return M[n].

Running time: O(n).

look up for x_{ej} can be done in constant time.

To do that:

initially for each x_i : define $x_i' = x_i - 5$

now we have

x_1 ..... x_n

x_1' ..... x_n'

Now merge two lists using merge routine (O(n))

Scan through merged list, if you get x_j'

x_{ej} is next to it.