0-1 knapsack & Subset Sum

Problem Statement

- A single resource
- Requests $\{1..n\}$ each take time w_i to process
- Can schedule jobs at any time between 0 to W

Objective: To schedule jobs such that we maximize the machine's utilization

$OPT(i)$ = value of the opt. sol. for requests $1 \ldots i$

if $i \notin O_i$, then $OPT(i) = OPT(i-1)$

if $i \in O_i$, then $OPT(i) = w_i + OPT(i-1)$

$OPT(i,w)$ = value of the opt. solution using a subset of the items $\{1..i\}$ with Max. allowed weight w.

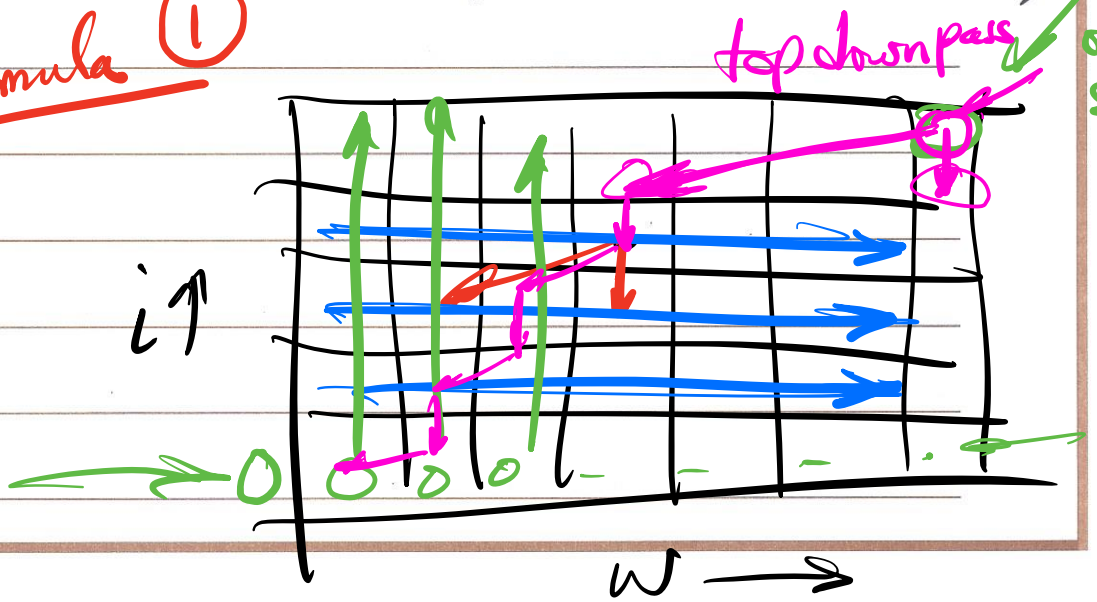
if $n \notin O$, Then $OPT(n,w) = OPT(n-1,w)$

if $n \in O$, Then $OPT(n,w) = v_n + OPT(n-1, w-w_n)$ ← for 0/1 Knapsack

If $w < w_i$, Then $OPT(i,w) = OPT(i-1,w)$

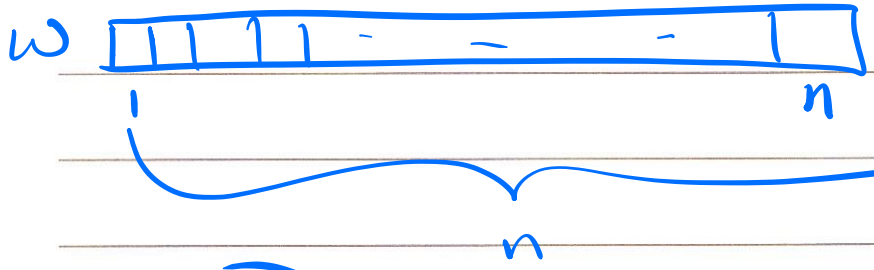
else, $OPT(i,w) = Max(OPT(i-1,w), w_i + OPT(i-1, w-w_i))$

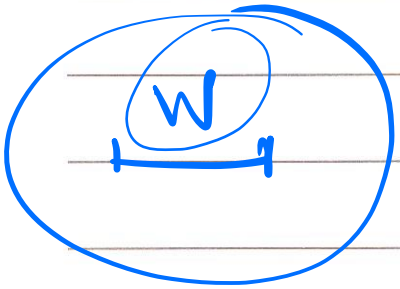
Rec. formula (1)



```
Subset-sum (n, w)
  array M[0,w] = 0  for each w = 0 to W
  for i = 1 to n
    for w = 0 to W
      use recurrence formula (1)
        to compute M[i, w]
    end for
  end for
  Return M[n, W]
```

This takes $\Theta(nW)$





$\log W$

$W = 2^{\log_2 W}$

Pseudopolynomial Complexity

$\Theta(n 2^{\log_2 W})$

Pseudo-polynomial time

An algorithm runs in pseudo-polynomial time if its running time is a polynomial in the numeric value of the input

Polynomial time

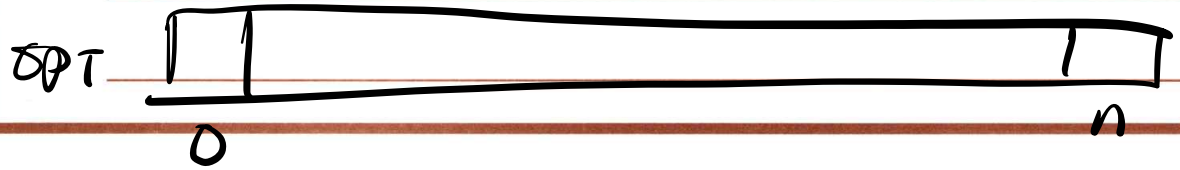
An algorithm runs in polynomial time if its running time is a polynomial in the length of the input (or output).

Austrian Schillings denominations

1
5
10
20
25

$OPT(i)$ = The min number of coins to pay for i schillings

$$OPT(i) = \min(OPT(i-1)+1,\ OPT(i-5)+1,\ OPT(i-10)+1,\ OPT(i-20)+1,\ OPT(i-25)+1)$$



$OPT(0) = 0$, $OPT(1) =$, . . . $OPT(24)$

for $i = 25$ ~~1~~ to n

end for

Dynamic Programming

Sequence Alignment Problem

A DNA strand consists of a string of molecules called bases.

4 Types of bases:

- Adenine A
- Cytosine C
- Guanine G
- Thymine T

$S_1 = ACCGGTCG$

$S_2 = CCAGGTGGC$

```
A CC - GGTCG -
- CC A GGTGGC
```

3 Gap

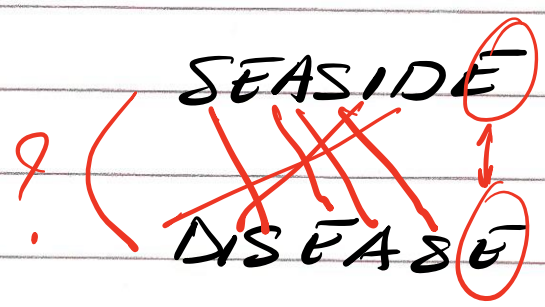
mismatch

Suppose we have 2 strings X & Y.

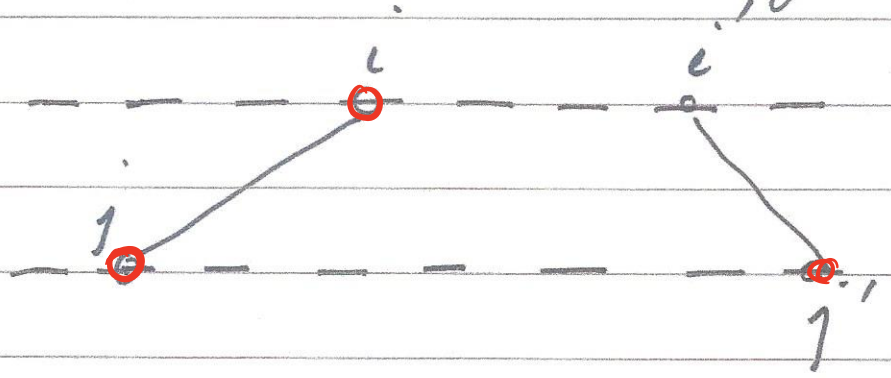
$X = \{x_1, x_2, \ldots, x_m\}$

$Y = \{y_1, y_2, \ldots, y_n\}$

Def. A matching is a set of ordered pairs with property that each item occurs at most once.



Def. A matching is an alignment if there are no crossing pairs.



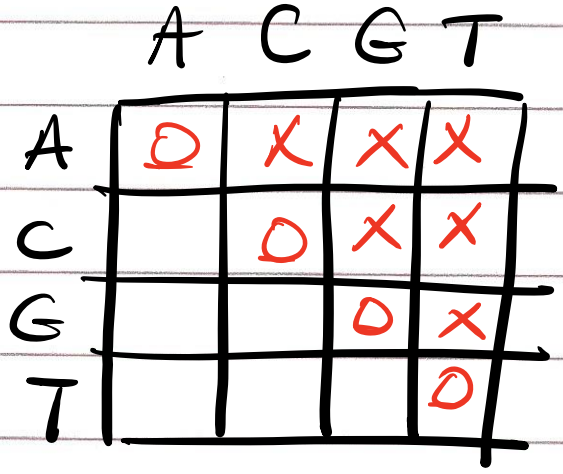
$(i, j), (i', j') \in M$

& $i < i' \rightarrow j < j'$

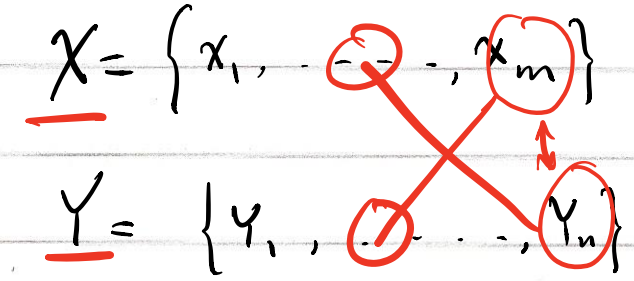
For an alignment M between X & Y

1- We incur a "gap penalty" of δ for each gap.

2- For each mismatch (of letters p & q) we incur a mismatch cost α_{pq}



Def Similarity between strings X & Y is the minimum cost of an alignment between X & Y.



Say M is an opt. solution.
either $(x_m, y_n) \in M$ or $(x_m, y_n) \notin M$

Define OPT(i, j) as the min cost of an alignment between $x_1 \dots x_i$ & $y_1 \dots y_j$

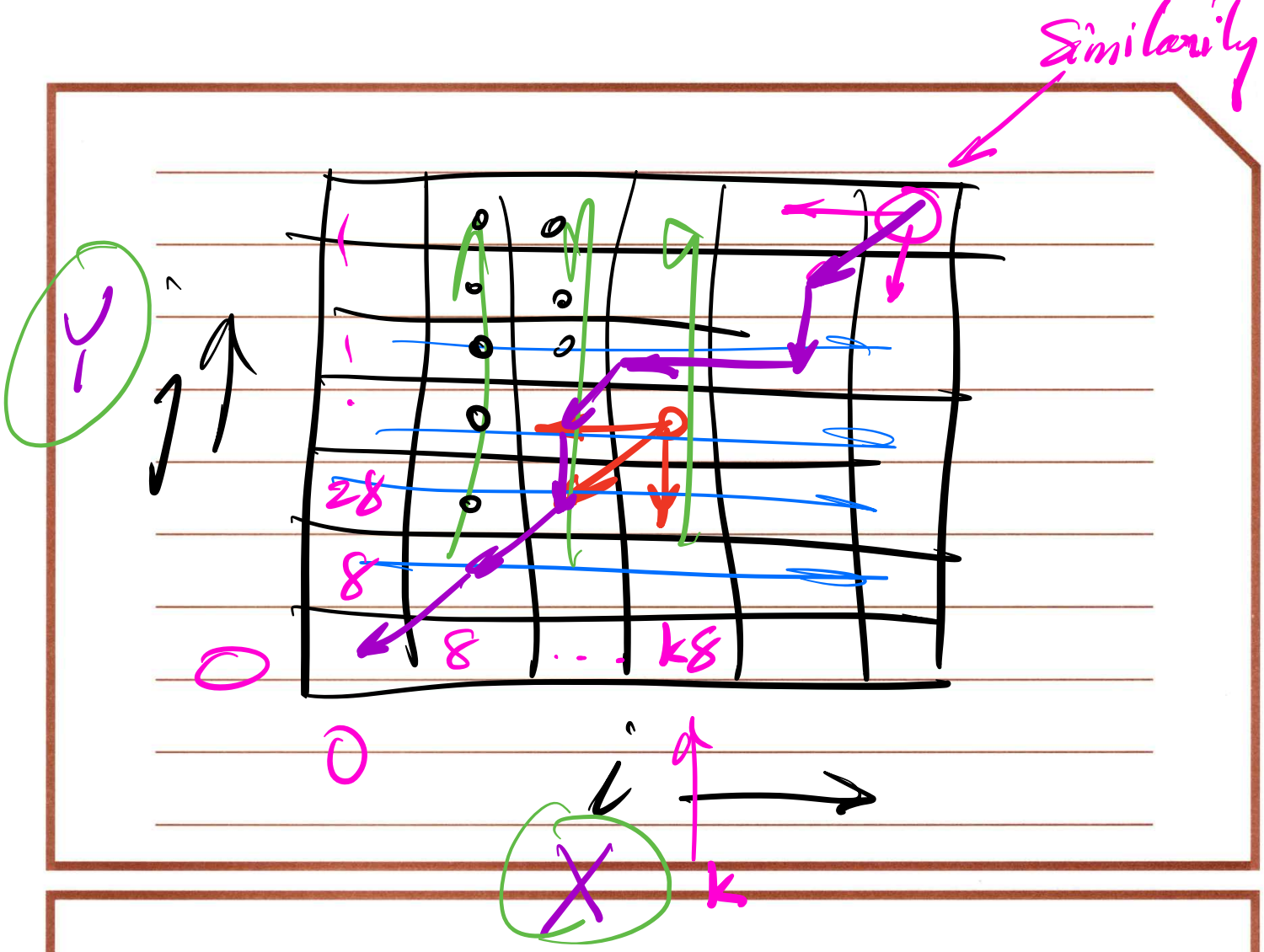
In an optimal alignment M, at least one of the following is true:

1) - $(x_m, y_n) \in M \Rightarrow OPT(m,n) = OPT(m-1, n-1) + \alpha_{x_m y_n}$

2) - x_m is not matched $\Rightarrow OPT(m,n) = OPT(m-1, n) + \delta$

3) - y_n is not matched $\Rightarrow OPT(m,n) = OPT(m, n-1) + \delta$

$$OPT(i,j) = \min \left[\alpha_{x_i y_j} + OPT(i-1, j-1), \right.$$
$$\delta + OPT(i-1, j),$$
$$\left. \delta + OPT(i, j-1) \right]$$

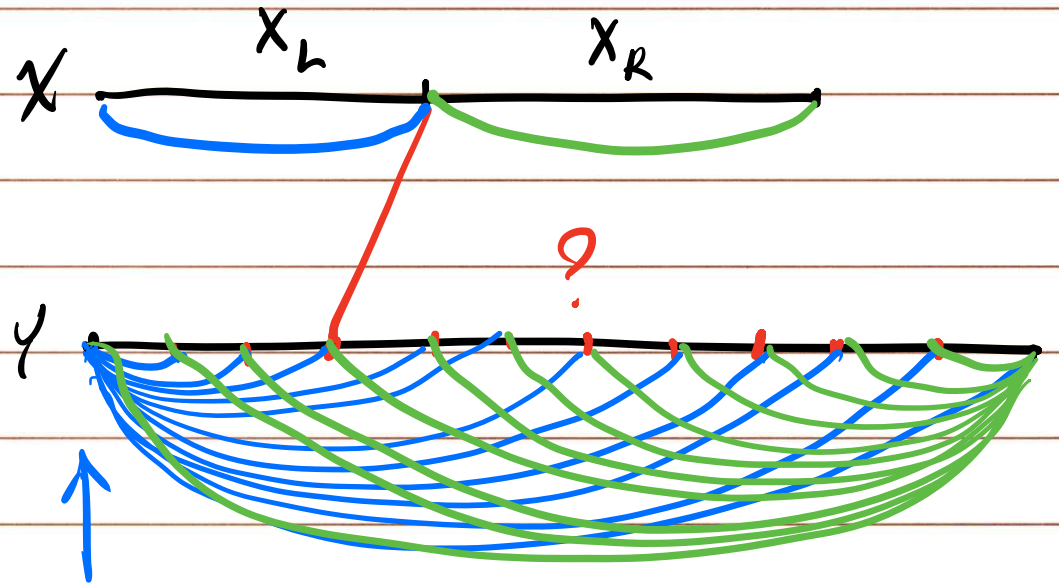


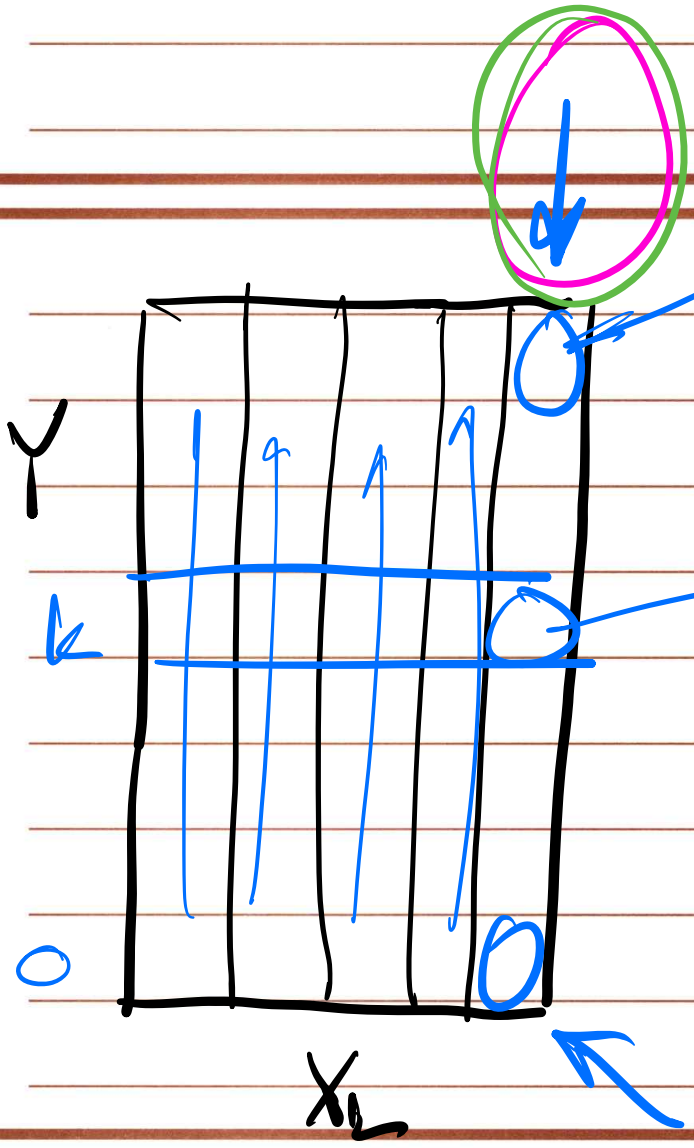
Cost of finding the similarity

$= \Theta(mn)$ ← efficient solution

Cost of finding the opt. alignment

$= \Theta(m+n)$

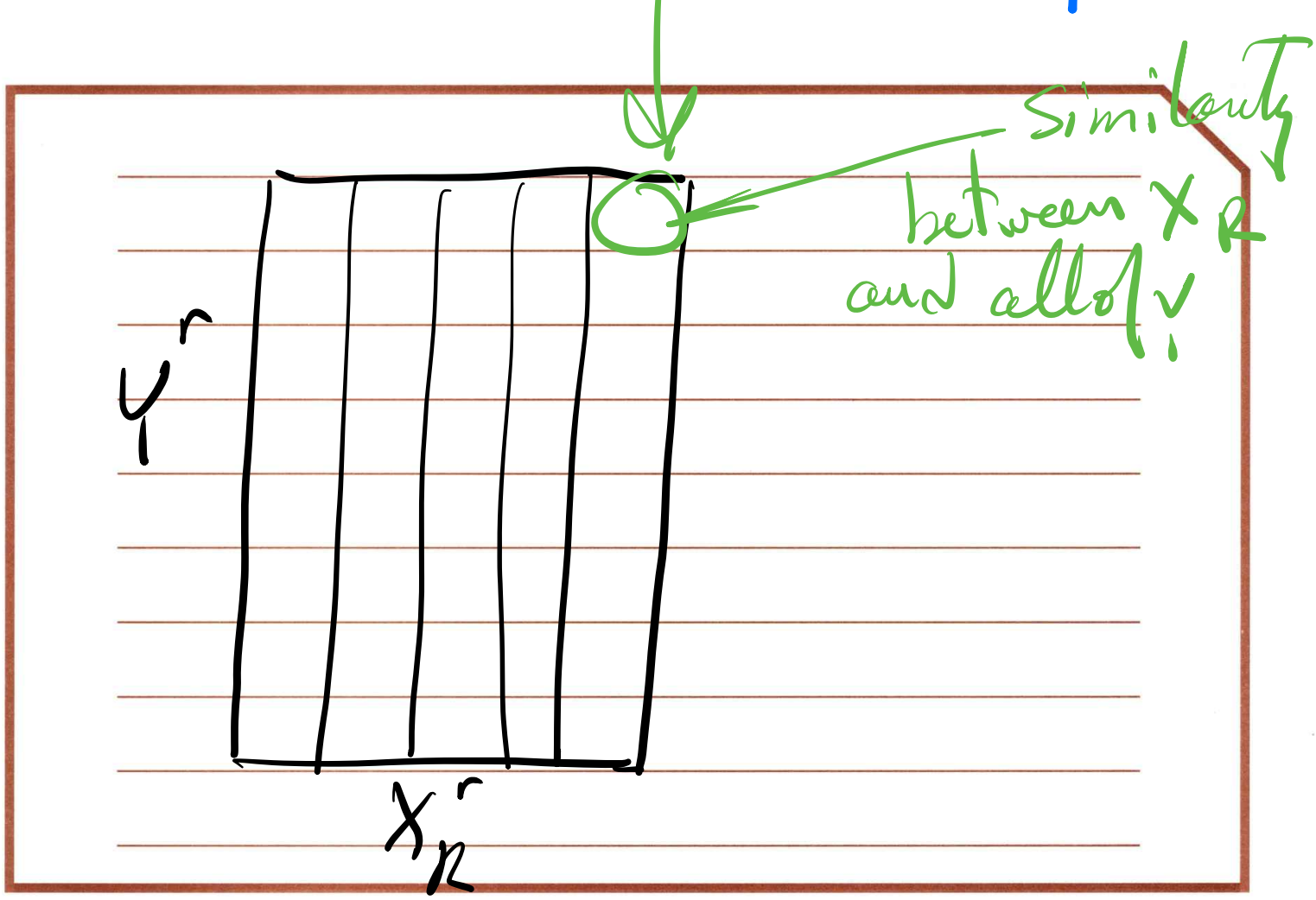


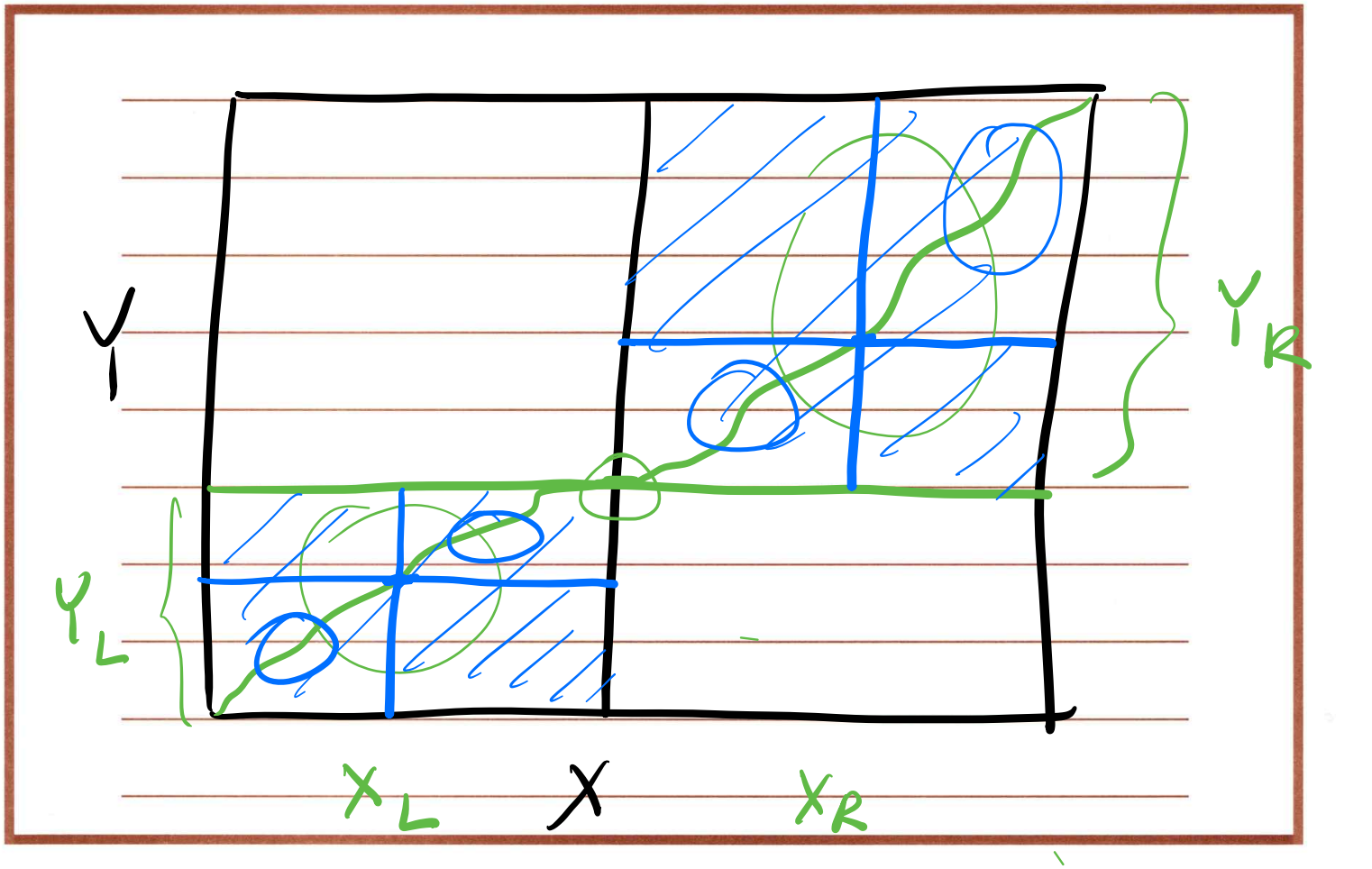


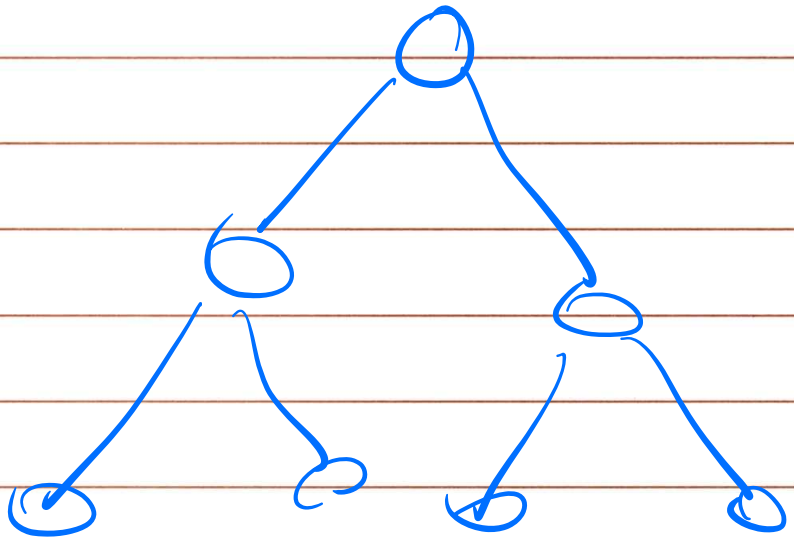
Similarity between X_L and all of Y

Similarity between X_L and $Y_1 .. Y_k$

similarity between X_L and none of Y


Similarity
between X_R
and allo!!
Y_R
X_R


Y
Y_R
Y_L
X_L
X
X_R



$$cmn + cmn/2 + cmn/4 + \cdots$$

$$\leq 2cmn = \Theta(mn)$$

Matrix Chain Multiplication

$A = A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n$

B. C. D

B is 2x10
C is 10x50
D is 50x20

$B.(C.D) = 10,400$

$(B.C).D = 3,000$

$m \left[\quad\right]_n \left[\quad\right]_k$

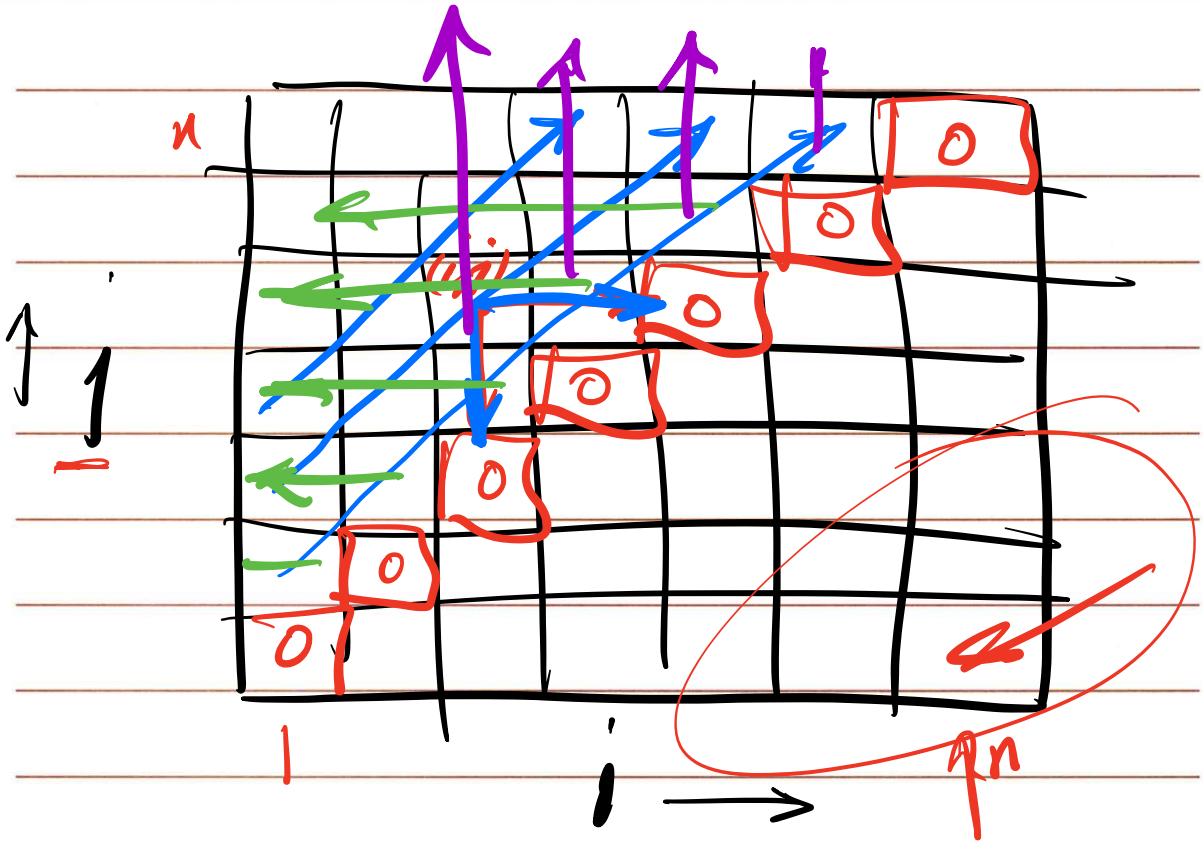
total op's $= m \times n \times k$

$$(A_i \dots A_k)(A_{k+1} \dots A_j)$$

$OPT(i,j)$ = opt. cost of multiplying matrices $i \dots j$

$$OPT(i,j) = \min_{i \le k < j} \{ OPT(i,k) + OPT(k+1,j) + R_i C_k C_j \}$$

recurrence formula (2)



for $i=1$ to n, $OPT(i,i)=0$

for $j=2$ to n
  for $i=j-1$ to 1 by -1
    use recurrence (2)
  endfor
endfor

Takes $O(n^3)$