DP 5 Discrete Knapsack items (value, weights)

$v_1\ v_2 \ldots v_n$

$w_1\ w_2 \ldots w_n$

Z = Knapsack total weight

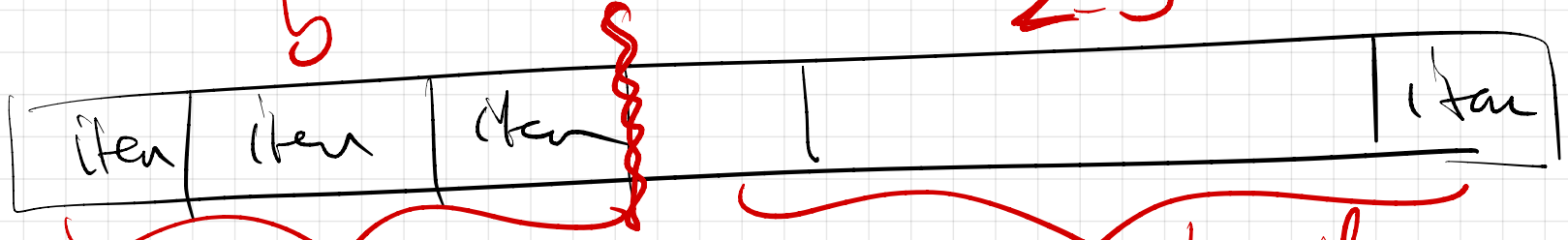
Whole or nothing

OBJ: max total value
$\sum$ weights $\leq Z$

- critical diff vs coins, rod-cuts:

table changes because each item can be used 0/1 times.

itemset (or similar) must be part of changing input



1 Knapsack

optimal C[b; all items]

optimal C[w-b; all items not on left]

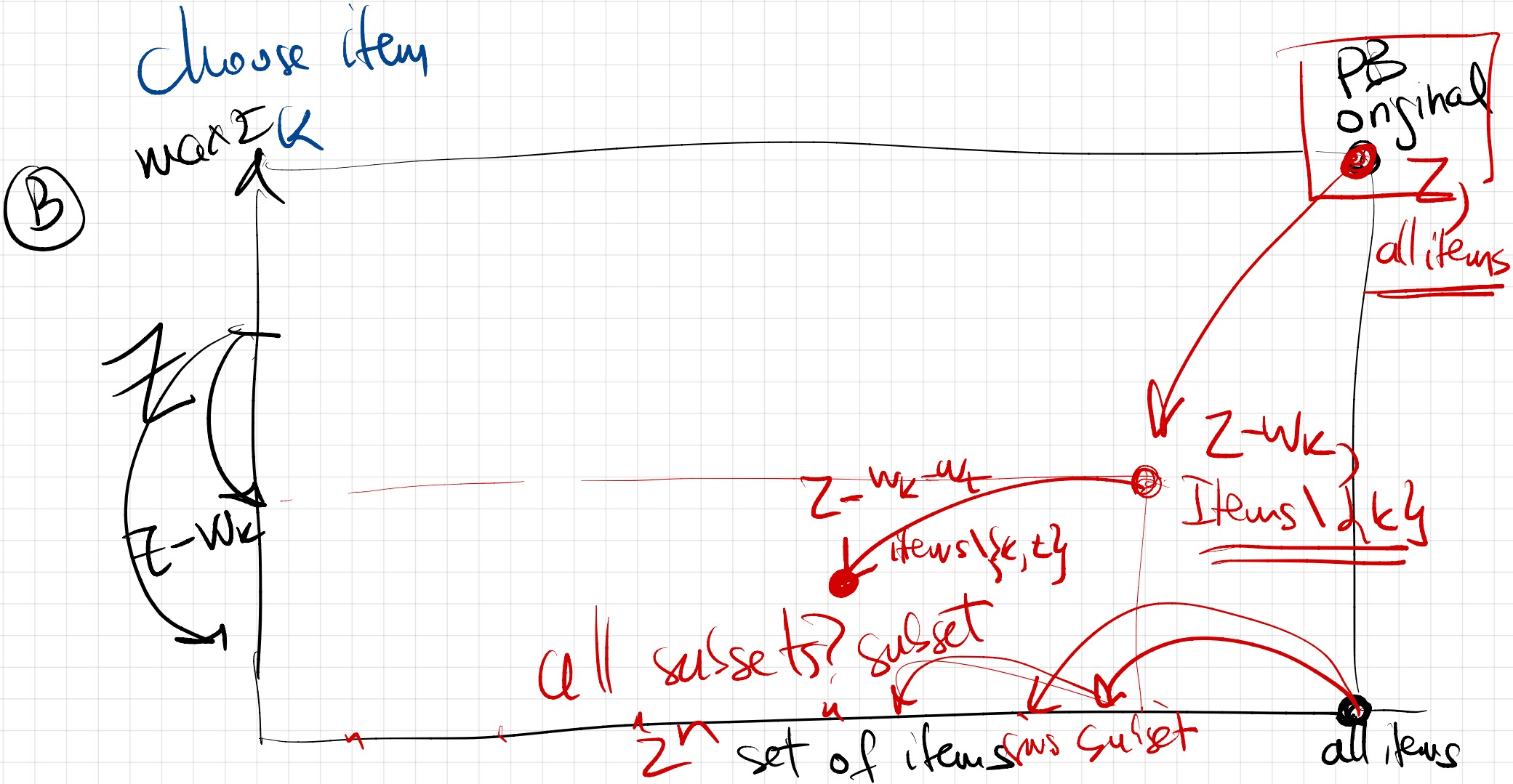
Lecture 7

all items = set

(2) tentative ? wishful thinking

A

$$C[Z, \text{all items}] = V_K + C[Z - W_K, \text{all items} \setminus \{K\}]$$



Indexing Trick: index elements global order

1, 2, 3, . . . n not changeable

not necess. input order, weight-sort, value-sort

Item set $I_n = \{1, 2, \ldots n\}$ $I[1:n]$

Partial sets $I_{n-1} = \{1, 2, \ldots n-1\}$ $I[1:n-2]$

$I_{n-2} = \{1, 2, \ldots n-2\}$ $I[1:n-2]$

$I_4 = \{1, 2, 3, 4\}$ $I[1:4]$

$\Theta(n)$ subsets (prefix)

step 1?

step 2A $C[Z, I_n] = \max($ ~~don't use item n~~ , $v_n + C[Z - w_n, I_{n-1}])$

knapsack, items available, item chosen, prefix set

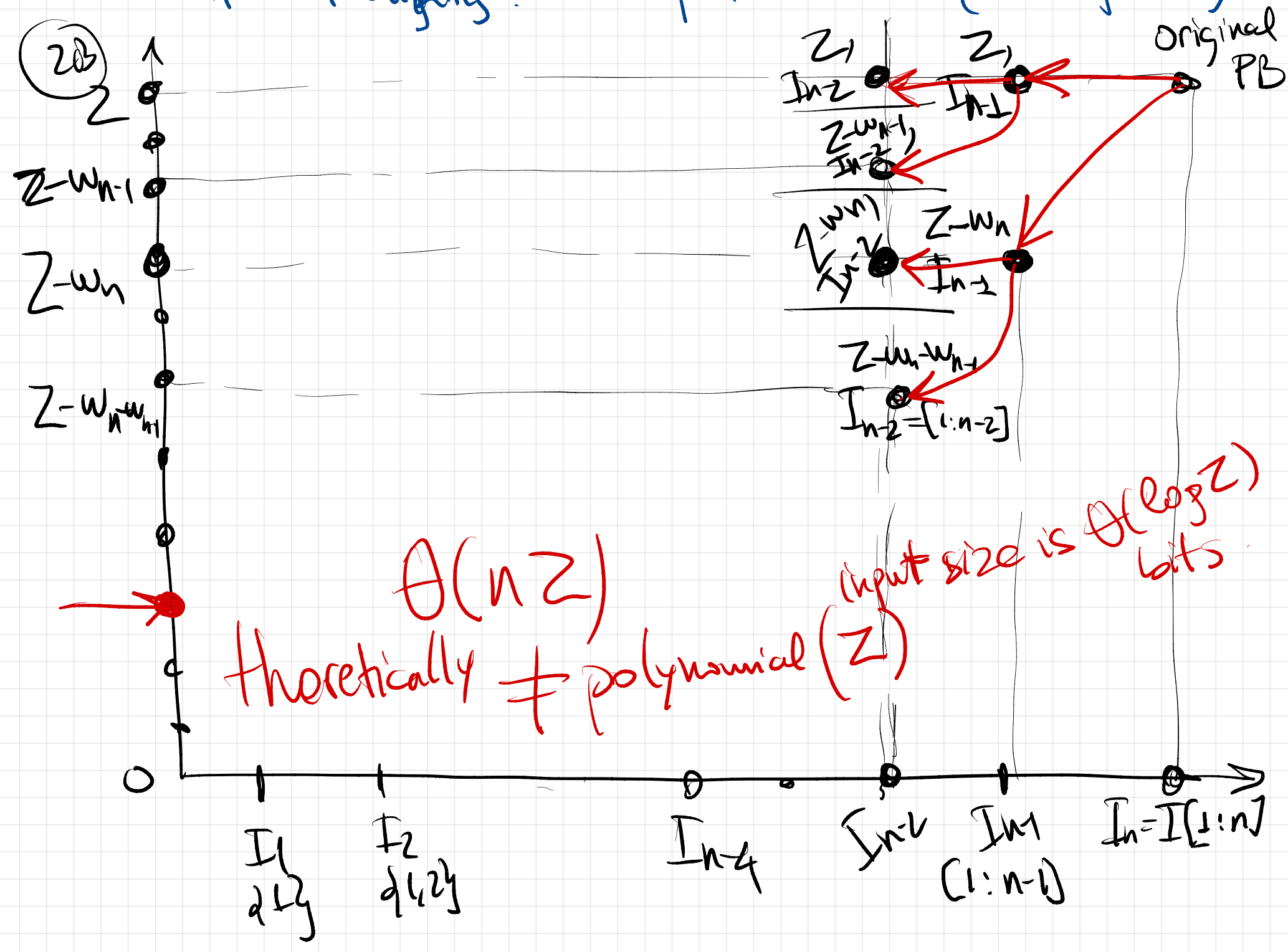
$\Theta(1)$

search (max)

$C[Z, I_{n-1}]$

$S[Z, I_n] = 1$ if n is used, 0 if not

Wishful thinking: use prefix subsets (indexing trick)



Y axis $Z, Z-w_n, Z-w_n-w_{n-1}, Z-w_{n-1}$

Z - subset of weights

- requirement: Z, w integers (discrete range)

example $I = [1:15]$

step 1:

OPTSOL $\boxed{i_1\ 2 \mid i_2\ 5 \mid i_3\ 6 \ldots \ldots \mid i_{m-1}\ 7 \mid i_m\ 9}$

assume OPTSOL its presented in index order (global)

OPTSOL $\Rightarrow$ OPTSOL $\Rightarrow$ i_m $\oplus$ OPTSOL $(Z-w_m, I_{i_m-1})$

Z, all items $Z, I[1:i_m]$ item i_m (excluded)

$1, 2, \ldots, i_m-1$

$1:8$

Matrix Chain Multiplication

ex $A \times (B \times C) = (A \times B) \times C$

$n=3$ 30×5 5×10 10×20

cel

$5\times10\times20$

5×20

$30\times5\times20$

$30\times5\times10$

30×10

$30\times10\times20$

last operation

total $1500 + 6000$

total $3000 + 1000$

STEP 1

$n=n$

optimal

$A_1 \cdot A_2 \cdot A_3 \cdots A_k \cdot A_{k+1} \cdots A_n$

optimal

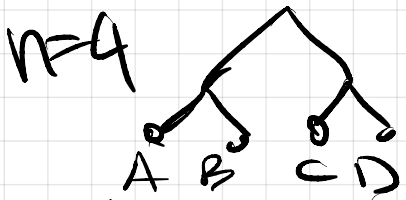
$P_0\times P_1$ $P_1\times P_2$ $P_2\times P_3$ $P_0 \times P_k$ $P_{k-1}P_k$ P_kP_{k+1} $P_k \times P_n$ $P_{n-1}\times P_n$

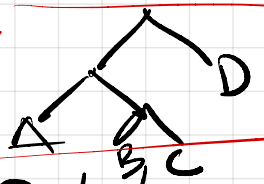
What is the best multip order? $\equiv$ putting parentheses

last multip (op) : $P_0 \times P_k \times P_n$

Brute force: Try all possible parenthesis

$(A(B))(C\,D)$ | $((A\,B)\,C)\,D$ | $(A\,(B\,C))\,D$ | $A(B(C\,D))$ | $A(B(C\,D))$





#ways to parenthesis = Catalan(n) $\simeq 4^n$ exact $\binom{2n}{n} - \binom{2n}{n-1}$

last

$(A((B\,C)\,D))(E\,F)$

step 2

$A_1 \cdot A_2 \cdots$

$C[i,j]$

$A_i \cdot A_{i+1} \cdots \underset{k \mid k+1}{} A_j \, A_{j+1} \cdots A_n$

$p_{i-1} \times p_k \quad p_k \times p_j$

$C[i,j]$ = best cost for multiplying $A_i \cdot A_{i+1} \cdots A_j$

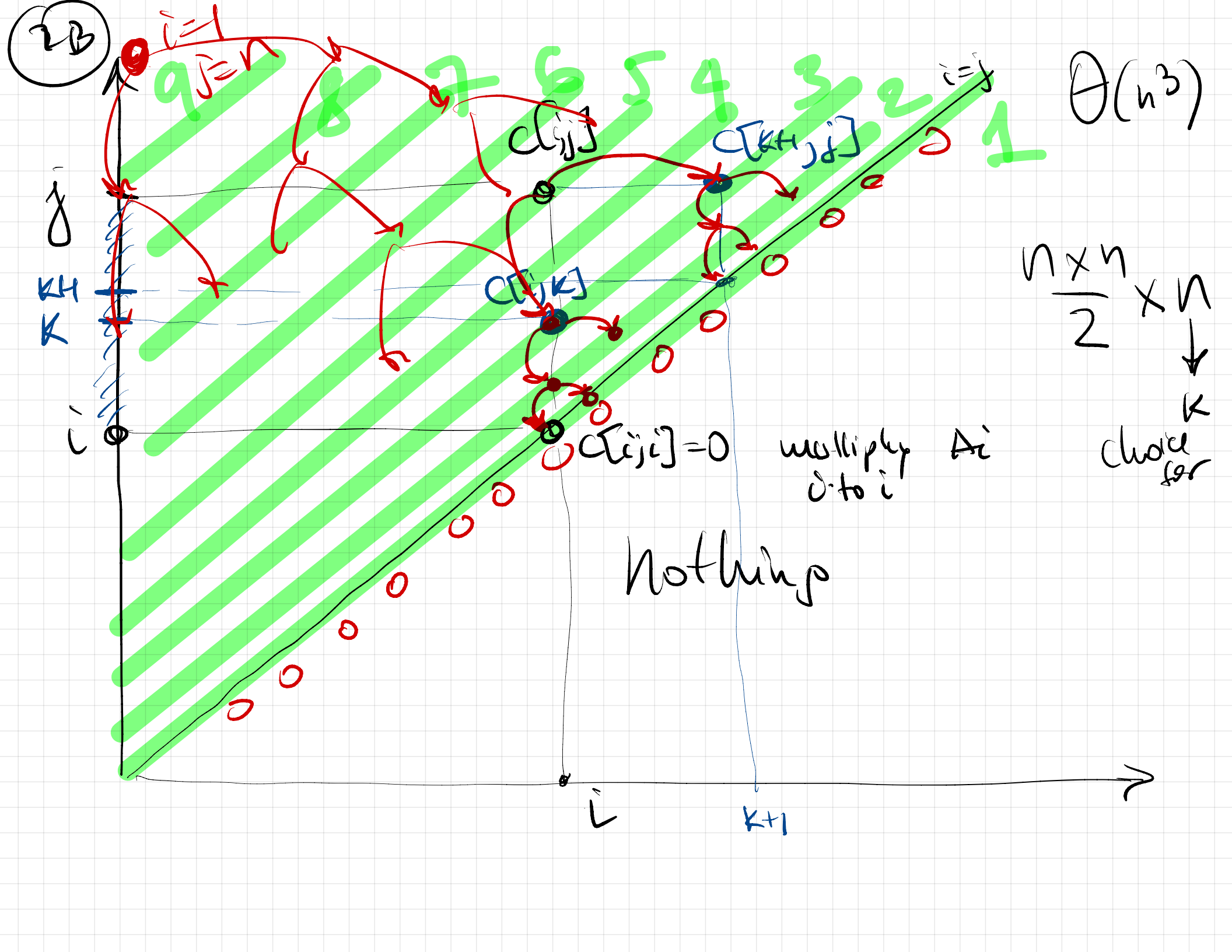
$i \le j$

look for last op (k)

cost last op $p_{i-1} \cdot p_k \cdot p_j + C[i,k] + C[k+1, j]$

$i < k < j$

$s[i,j] = k$


2B
i=1
j=n
8
7
6
5
4
3
2
1
i=j
θ(n³)
C[i,j]
C[k+1,j]
C[i,k]
j
k+1
k
i
C[i,i]=0
multiply Ai
i to j
n×n
2
×n
k
choice for
Nothing
i
k+1

```
MEMOIZED-MATRIX-CHAIN(p)
1  n = p.length − 1
2  let m[1..n, 1..n] be a new table
3  for i = 1 to n
4      for j = i to n
5          m[i, j] = ∞
6  return LOOKUP-CHAIN(m, p, 1, n)

LOOKUP-CHAIN(m, p, i, j)
1  if m[i, j] < ∞
2      return m[i, j]
3  if i == j
4      m[i, j] = 0
5  else for k = i to j − 1
6          q = LOOKUP-CHAIN(m, p, i, k)
               + LOOKUP-CHAIN(m, p, k + 1, j) + p_{i−1} p_k p_j
7          if q < m[i, j]
8              m[i, j] = q
9  return m[i, j]
```

The MEMOIZED-MATRIX-CHAIN procedure, like MATRIX-CHAIN-ORDER, maintains a table $m[1..n, 1..n]$ of computed values of $m[i, j]$, the minimum number of scalar multiplications needed to compute the matrix $A_{i..j}$. Each table entry initially contains the value ∞ to indicate that the entry has yet to be filled in. Upon calling LOOKUP-CHAIN(m, p, i, j), if line 1 finds that $m[i, j] < \infty$, then the procedure simply returns the previously computed cost $m[i, j]$ in line 2. Otherwise, the cost is computed as in RECURSIVE-MATRIX-CHAIN, stored in $m[i, j]$, and returned. Thus, LOOKUP-CHAIN(m, p, i, j) always returns the value of $m[i, j]$, but it computes it only upon the first call of LOOKUP-CHAIN with these specific values of i and j.

Figure 15.7 illustrates how MEMOIZED-MATRIX-CHAIN saves time compared with RECURSIVE-MATRIX-CHAIN. Shaded subtrees represent values that it looks up rather than recomputes.

Like the bottom-up dynamic-programming algorithm MATRIX-CHAIN-ORDER, the procedure MEMOIZED-MATRIX-CHAIN runs in $O(n^3)$ time. Line 5 of MEMOIZED-MATRIX-CHAIN executes $\Theta(n^2)$ times. We can categorize the calls of LOOKUP-CHAIN into two types:

1. calls in which $m[i, j] = \infty$, so that lines 3–9 execute, and
2. calls in which $m[i, j] < \infty$, so that LOOKUP-CHAIN simply returns in line 2.

Dependency (order subpb comp) ⇔ recurse stack

recursive

if m[ij] already computed not make recursive call

m = C

Memoization

- C = still a table
- C[] cache computed values
- never compute again C
- never recurse on computed value

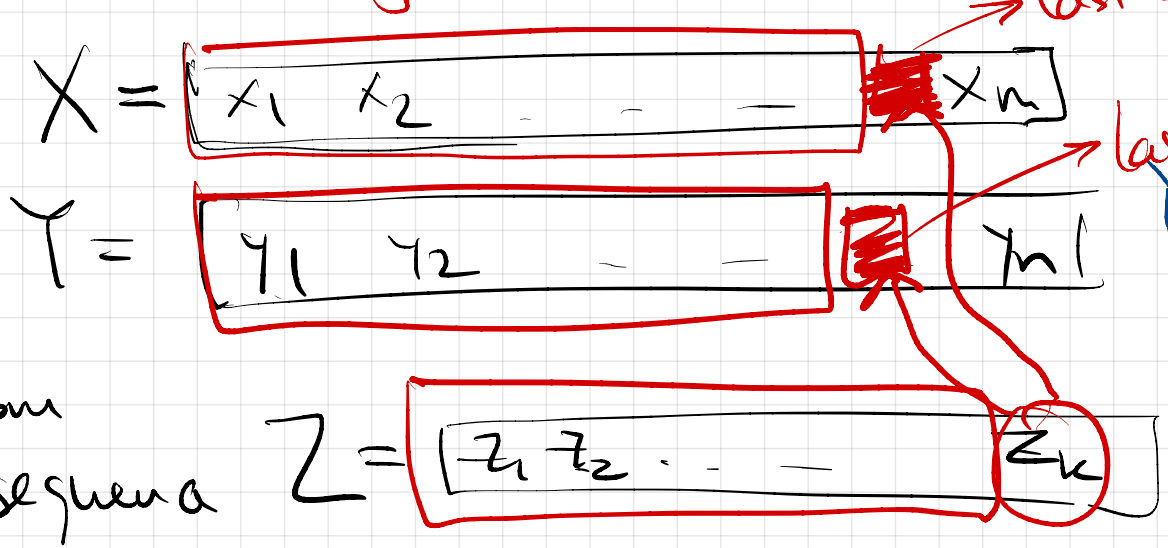
only recurse on non-computed C[] cell THAT ARE NEEDED

Memoization speeds up when it does not solve all subPBs in dependency table

PB	Memoization Must solve all subPBs anyway?
Rod Cut	
Coin Change	
CheckBoard	
0/1 Knapsack	
Matrix Chain	
LCS	
Optimal BST	

Exercise: fill this table

LCS = longest common subsequence



$X_i = X[1:i]$ prefixes based

$Y_j = Y[1:j]$ given index

Common subsequence

Z subseq of X

Z subseq of Y

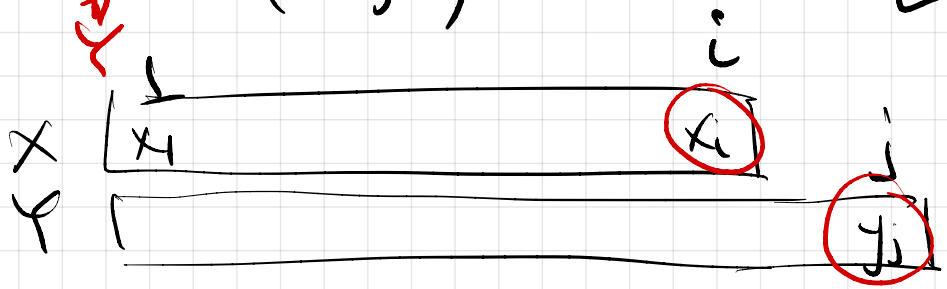
max k

① OPTSOL structure $= Z \Rightarrow Z_{k-1} = Z[1:k-1]$ opt solution to some $X[1:?]$, $Y[1:?]$ prefixes up to z_k value

assume

z_k occurs in X and Y

(2A) $C[i,j]$ = max (length) of common subseq $\begin{Bmatrix} X[1:i] \\ Y[1:j] \end{Bmatrix}$ LCS



$=$

z_k is last found $\iff x_i = y_j$

$C[i-1, j-1] + 1$

$x_i \neq y_j$ → cut x_i → $C[i-1, j]$

→ cut y_j → $C[i, j-1]$

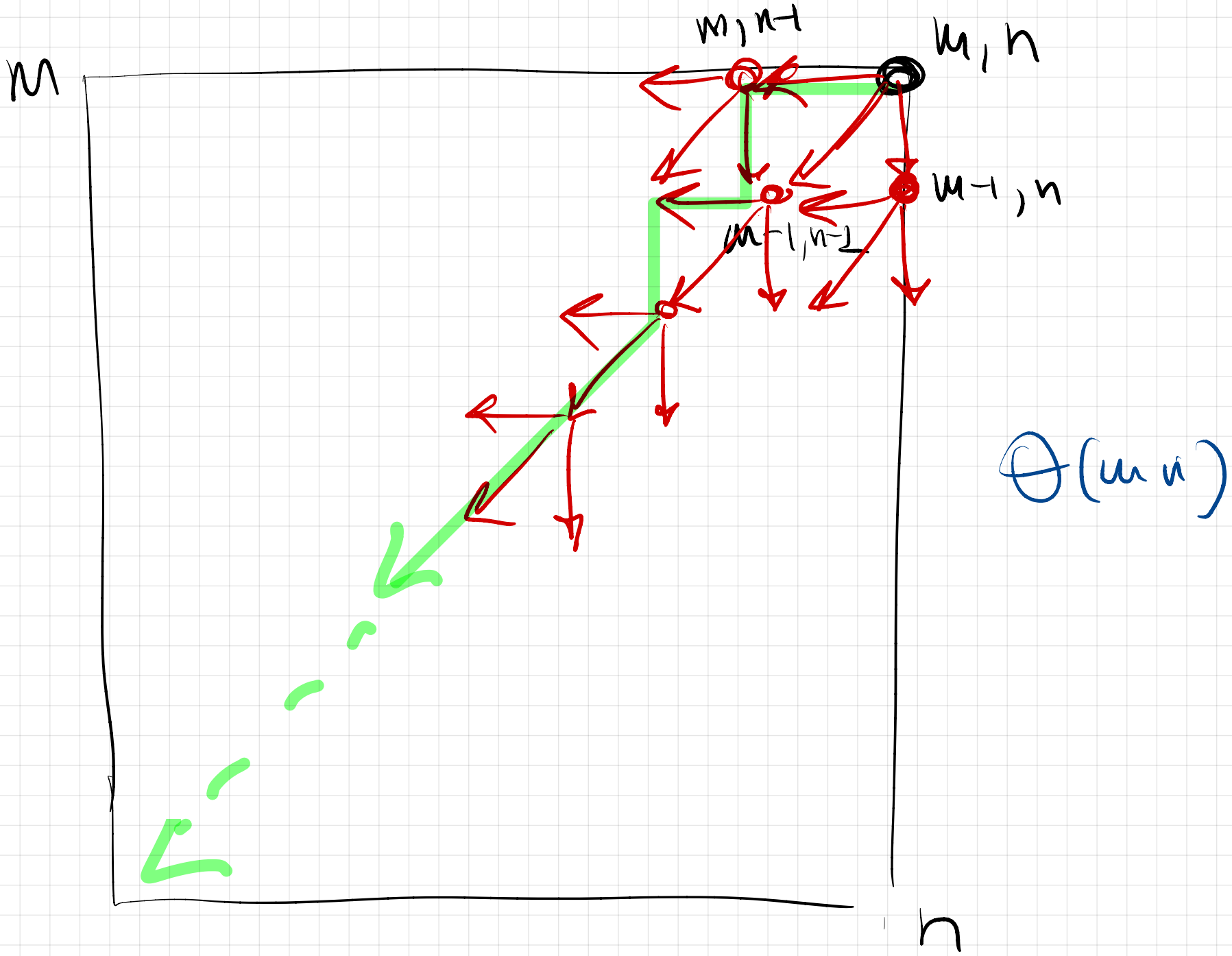
max of these

$\Theta(1)$

(2B)

$S[i,j] =$ { which wave is max-obj

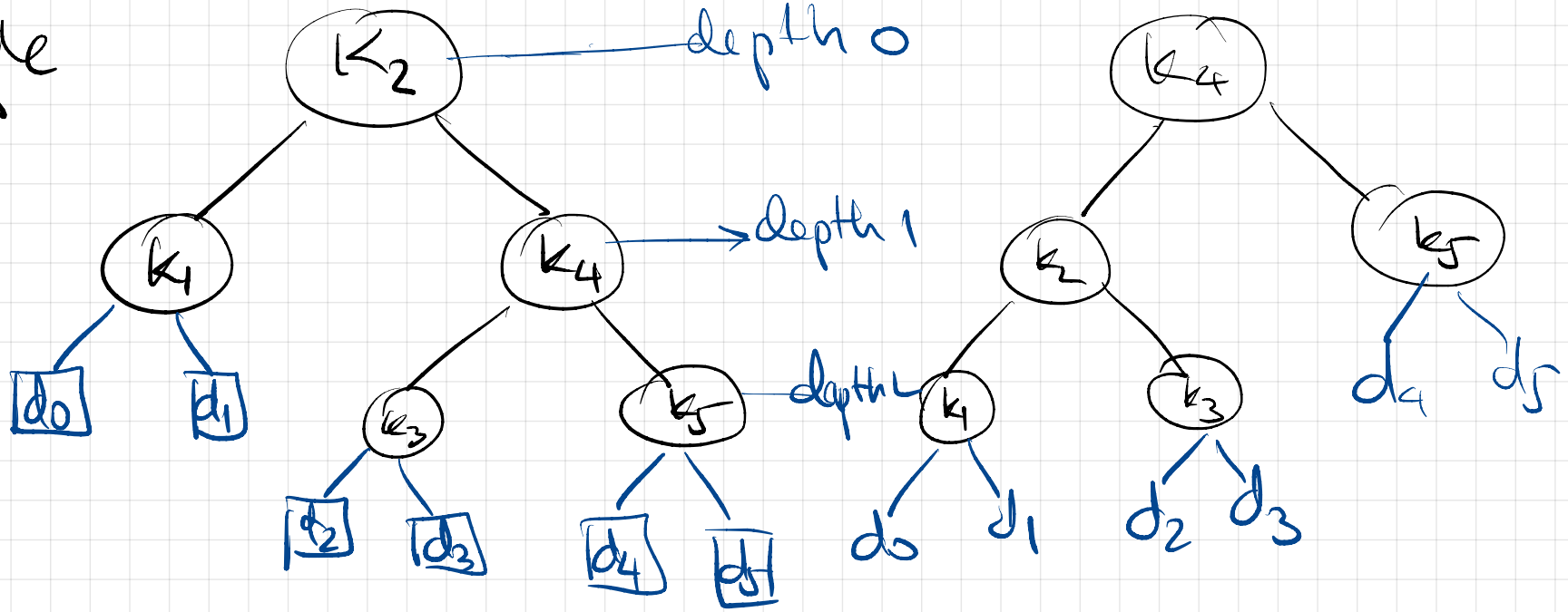
"←" "↙" "↓"



$\Theta(m n)$

optimal BST ordered values $k_1 \le k_2 \le \_ \_ \le$

example BST



Search probability: $Pr(k_i) = p_i$ not uniform

$Pr(d_i) = q_i$ d_i = searches for values $k_i < \text{val} < k_{i+1}$

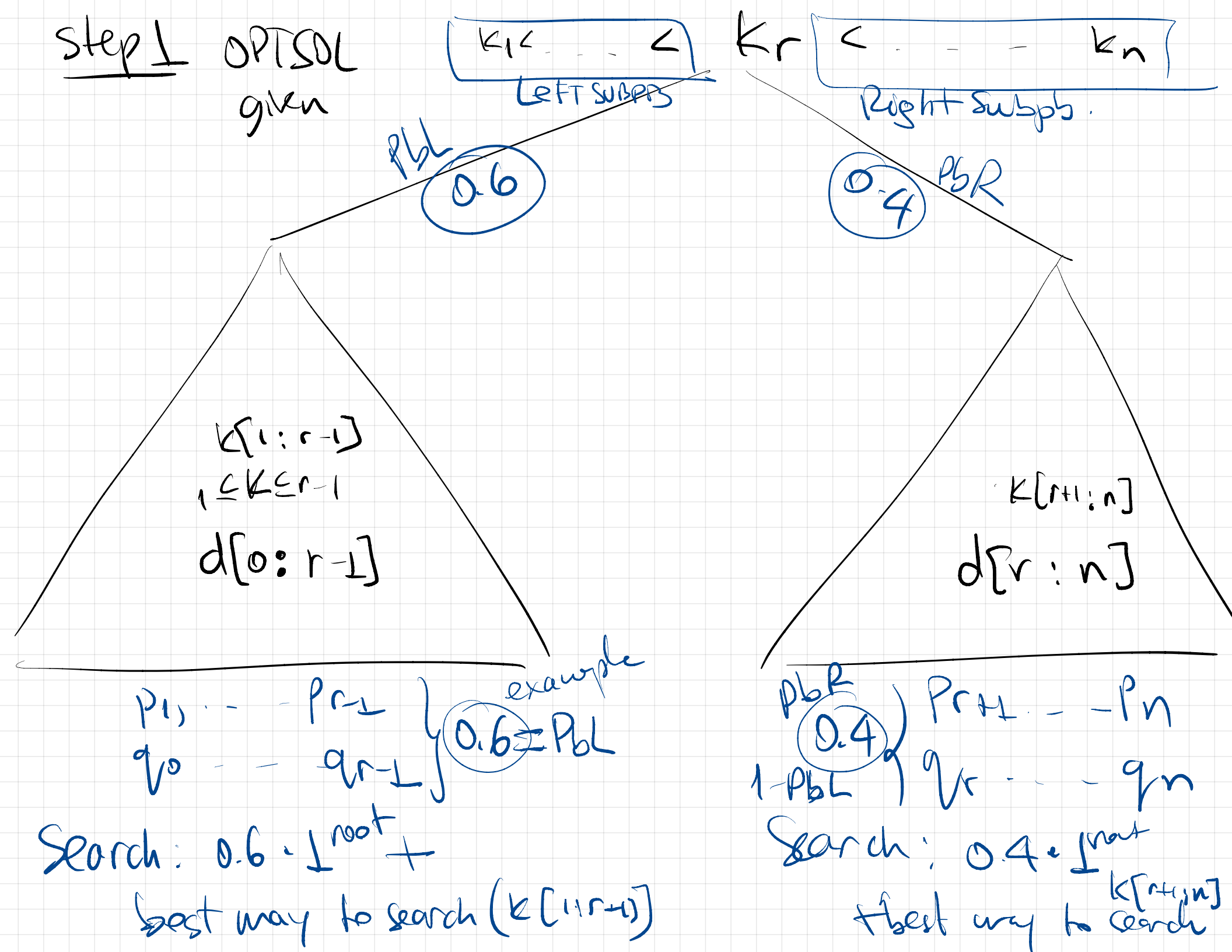
$\sum p_i + \sum q_i = 1$

depth = level

OPTIMALITY: min expected search cost

$$\sum_{i=1}^{n} [\text{depth}(k_i) + 1] p_i + \sum_{i=0}^{n} [\text{depth}(d_i) + 1] \cdot q_i$$

Step 1 OPTSOL given



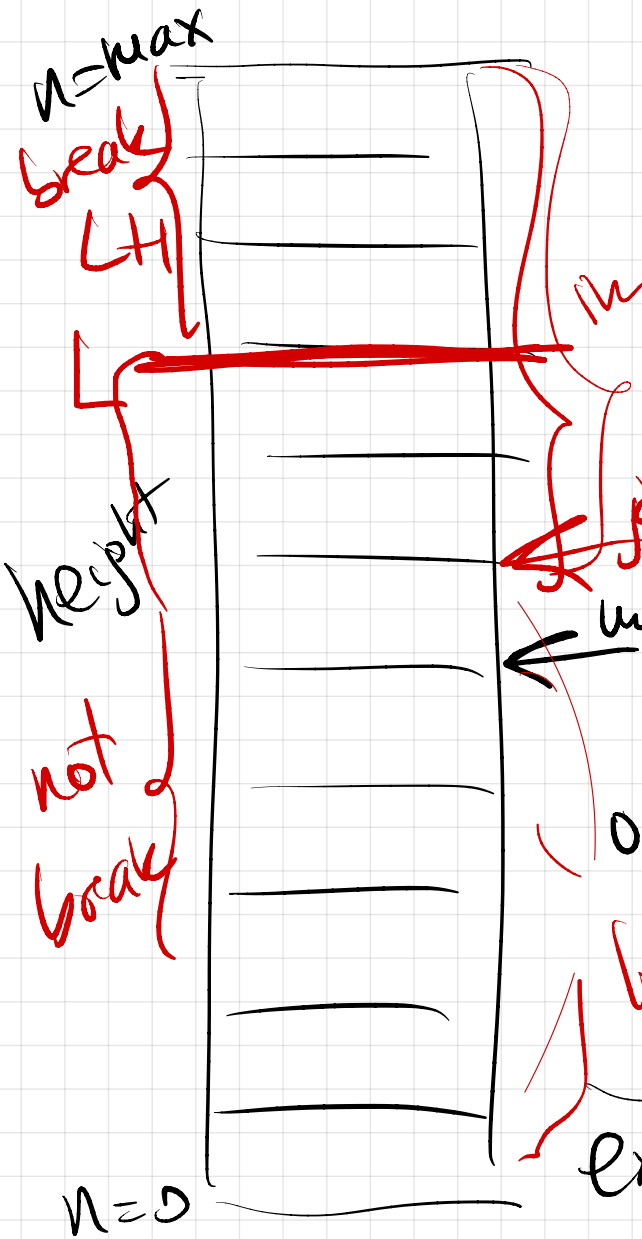
Search: $0.6 \cdot 1^{root} +$ best way to search $(k[1:r-1])$

Search: $0.4 \cdot 1^{root} +$ best way to search $k[r+1:n]$

Jars on Ladder

n steps k jars

Task: find highest level L where jars don't break (they break at $L+1$)



Trial jar at level ℓ

$\ell \le L \Rightarrow$ doesn't break $\Rightarrow$ can reuse it

$\ell \ge L+1 \Rightarrow$ jar breaks cannot reuse it

OBS: minimize # trials in worst case

You must 100% find L

ex. @ $k=1 \Rightarrow$ bottom up or str $\Theta(n)$

@ $k > \log n \Rightarrow$ binary search