Dynamic Programming

Motivation: Divide + Conquer can be a good algorithmic strategy

But: Can lead to inefficient solutions

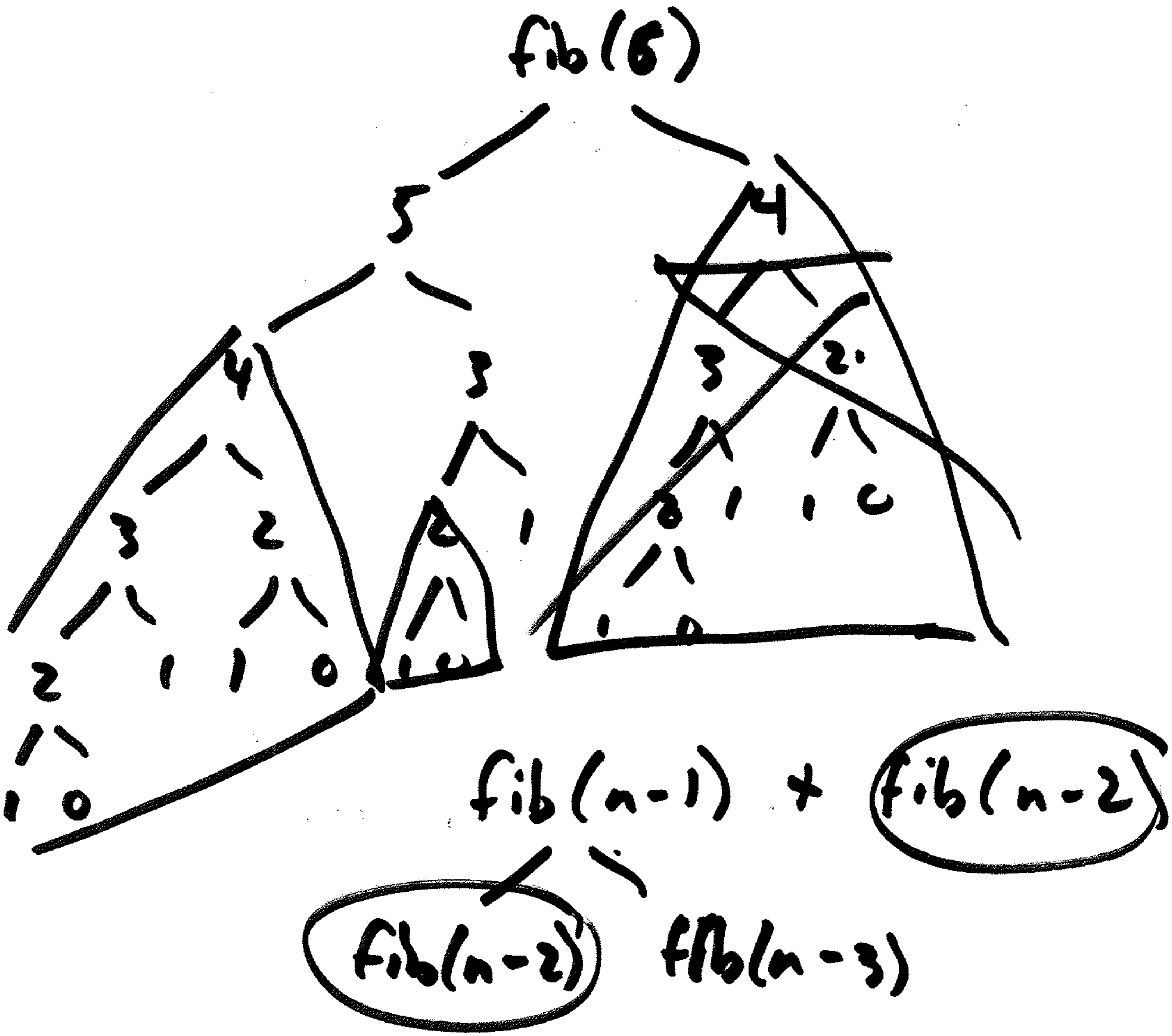
- Abuse of a call stack (recursion)
- Repeated Computation

Ex: Compute the Fibonacci numbers

$$F_n = F_{n-1} + F_{n-2} \qquad F_0 = F_1 = 1$$

$$1, 1, 2, 3, 5, 8, \underline{13}, \ldots$$

```
if (n = 0 OR n = 1)
    return 1
else
    return fib(n-1) + fib(n-2)
```


fib(6)
5
4
4
3
3
2
3
2
2
1
1
0
1
0
1
1
1
0
1
0
fib(n-1) + fib(n-2)
fib(n-2)
fib(n-3)

$\binom{n}{k}$ $\qquad\qquad$ $(x+y)^n =$

$$(x+y)^3 = (x+y)(x+y)^2$$

$$= (x+y)(x^2+2xy+y^2)$$

$$= x^3 + 2x^2y + xy^2 + x^2y + 2xy^2 + y^3$$

$$= \underset{\binom{3}{0}}{x^3} + \underset{\binom{3}{1}}{3}x^2y + \underset{\binom{3}{2}}{3}xy^2 + \underset{\binom{3}{3}}{y^3}$$

```
binomial(n, k):
    if(n = k or k = 0)
    L return 1
    else
        return binomial(n-1, k-1) +
                 binomial(n-1, k)
```

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!}$$ mults $n-2$

~~+ 1 division~~

$\approx n-3$ mults. $O(n)$ mults

Pascal:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{n}{0} = 1 \quad \binom{n}{n} = 1$$

	0	1	2	3	4 ...	k-1	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
⋮	⋮						
k	1						1
⋮	⋮						
n-1	1						
n	1						

— Pascal's Triangle

~~$\binom{2}{3}$ undefined~~

$\binom{n-1}{k-1} + \binom{n-1}{k}$

$\binom{n}{k}$

for $i = 0 \ldots n$

$\left.\begin{array}{l} C_{i,0} = 1 \\ C_{ii} = 1 \end{array}\right\}$ Base Cases

$A(n) \in O(n^2)$

for $i = 2 \ldots n$

  for $j = 1 \ldots (i-1)$

    $C_{ij} \leftarrow C_{i-1,j} + C_{i-1,j-1}$

FAST:

First Solution: Identify a Divide & Conquer
Approach/Solution
- Base Cases
- Division/sub calls/recursion

Analyze Solution: 1) Find an optimal substructure

2) Identify over<u>lapping</u> subsolutions

Subproblem Identification:

Memoize/cache

Turn Around: top-down solution → bottom up

1) Define/Derive a recurrence

2) Design a tableau:

- Identify base cases
- Identify dependencies
- Identify the final cell

3) Program a loop to fill out the table, consistent with

OBST = Optimal Binary Search Trees

goal: minimize average number of key comparisons for any search

note: necessarily a balanced tree.

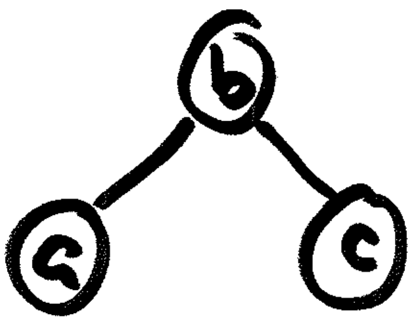
Application: • data is static (no inserts/deletion/updates); read-only tree

• you know the search probability of every key.

$a < b < c$

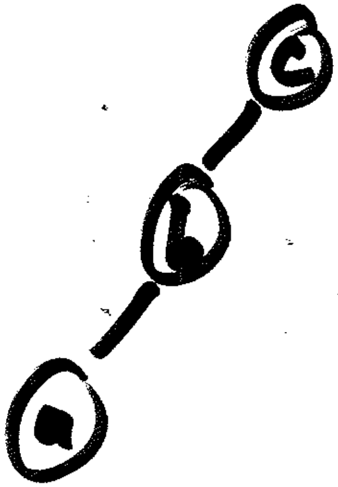
k	p(k)
a	.6
b	.3
c	.1

60% of the time a is searched
30% b is searched.
10% c searched

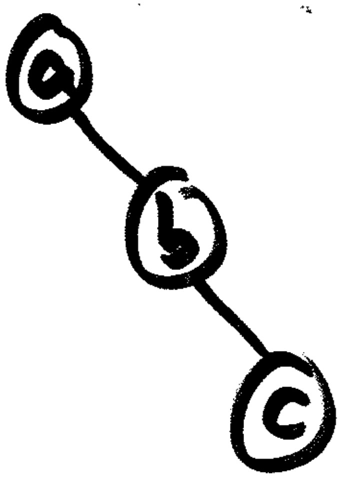


search for...	# comps	
a	2	x .6
b	1	x .3
c	2	x .1

1.7 comps on average.



key	# cmps		
a	3	× .6	1.8
b	2	× .3	.6
c	1	× .1	.1
			2.5



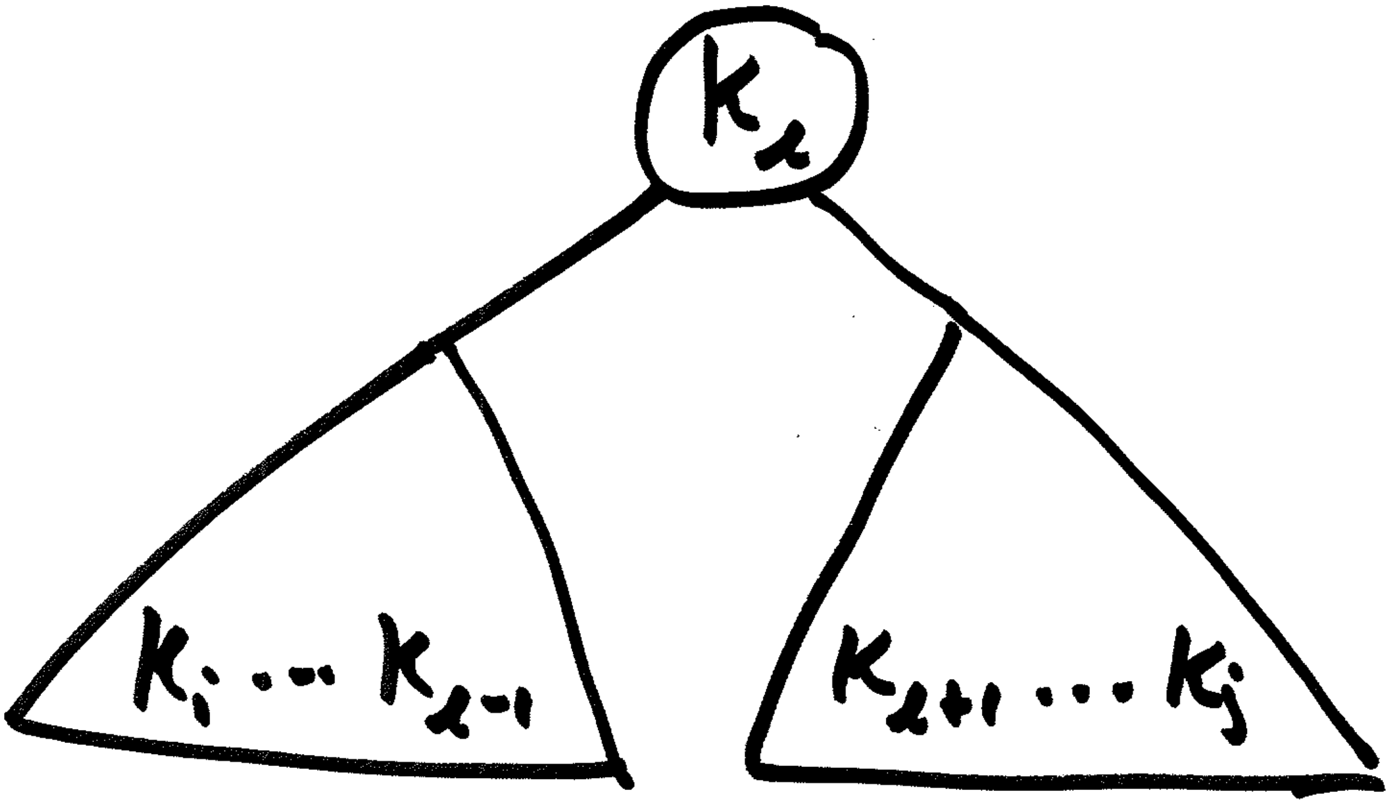
a	1	× .6
b	2	× .3
c	3	× .1
		1.5

< >

The number of different BST with n keys corresponds to the Catalan numbers:

$$C_n = \frac{1}{n+1}\binom{2n}{n} \in O\left(\frac{4^n}{n^{1.5}}\right).$$

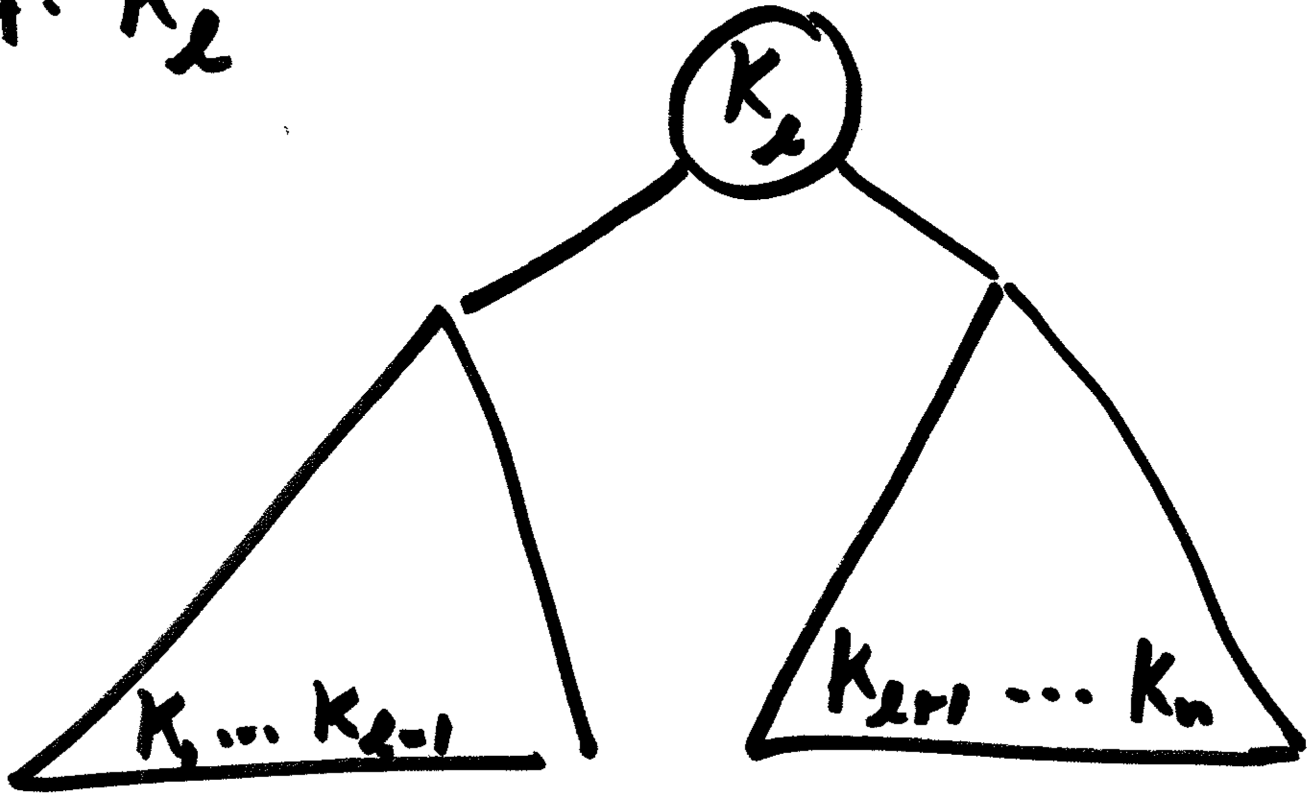
$K_i \ldots K_j$: find the best root K_r



Given keys $K_1\ K_2 \dots K_n$ $\quad K_1 < K_2 \dots < K_n$

Q: What is the best node to use as the root?

A: K_ℓ



$i \backslash j$	0	1	2	3	$\dots$	n
1	0	$p(k_1)$				C_{1n}
2		0	$p(k_2)$			
3			0	$p(k_3)$		
$\vdots$					$\ddots$	
n						$p(k_n)$
$n+1$						0

$k_i \ldots k_j$ $\quad$ $k_3\, k_4\, k_5$

Base Cases:

k_3

$k_4\, k_5$

$C_{i,i-1} = 0$ (empty tree)

$C_{ii} = 1 \cdot p(k_i)$

$= p(k_i)$ (single node tree)

Final Cell:

$C_{1,n}$

Input: A set of keys $k_1 \cdots k_n$, a prob. distribution p

Output: OBST Tableau

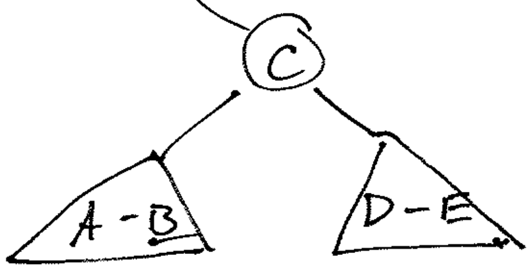
```
for i = 1...n
    C_{i,i-1} ← 0
    C_{ii} ← p(k_i)
    R_{ii} ← i    //root table
C_{n+1,n} ← 0
for d = 1...(n-1)    //dth-diagonal row
    for i = 1...(n-d)
        j ← d+i
        min ← ∞
        for l = i...j
            q ← C_{i,l-1} + C_{l+1,j}
            if (q < min)
                min ← q
```

$\lfloor$ $\lfloor$ $\lfloor$ $\lfloor$ $R_{ij} \leftarrow \ell$

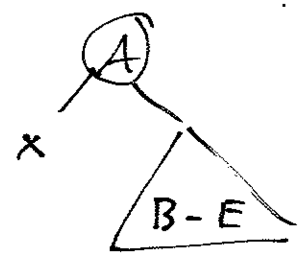
$\lfloor$ $\lfloor$ $C_{ij} \leftarrow \min + \sum_{s=i}^{j} p(k_s)$

output $\underbrace{C_{1,n}}_{\substack{\#\text{ of comparisons for the OBST} \\ \wedge \\ \text{average}}}$, R ← root table to build the tree.

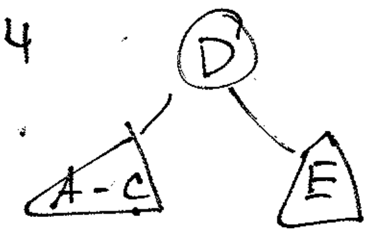
$$C_{1,5} = \min_{1 \le l \le 5} = \left\{ \begin{array}{lllll} l=1 & C_{1,0} & + & C_{25} & = 0 + 1.127 \\ l=2 & C_{1,1} & + & C_{35} & = .213 + 1.087 \\ l=3 & C_{12} & + & C_{45} & = .020 + .320 \\ l=4 & C_{13} & + & C_{55} & = 1.033 + .120 \\ l=5 & C_{14} & + & C_{65} & = 1.233 + 0 \end{array} \right. = \quad + \sum_{i=1}^{5} p(k_i)$$



$l = 1$



$l = 4$



Input: $n = 5$

Key	prob
A	.213
B	.020
C	.547
D	.100
E	.120

i \ j	0	1	2	3	4	5
1	0	.213	.020	1.033	1.233	$\boxed{1.573}$
2		0	.020	0.587	0.787	1.127
3			0	.547	0.747	1.087
4				0	.100	0.320
5					0	.120
6						0

$$C_{12} = \min_{1 \le l \le 2} \begin{cases} l = 1 & C_{1,0} + C_{2,2} = 0 + .020 = 0.020 \\ l = 2 & C_{1,1} + C_{3,2} = .213 + 0 = 0.213 \end{cases}$$

Defin: C_{ij} optimal # of comparisons in OBST with keys $K_i \ldots K_j$

$$C_{ij} = \min_{i \le l \le j} \left\{ C_{i,l-1} + C_{l+1,j} \right\} + \underbrace{\sum_{s=i}^{j} \overset{\text{prob. of } K_s}{p(K_s)}}_{\substack{\text{Captures the fact that the} \\ \text{new tree is deeper} \\ +1 \text{ comps for all keys } K_i \ldots K_j}}$$

Optimal Binary Search Trees

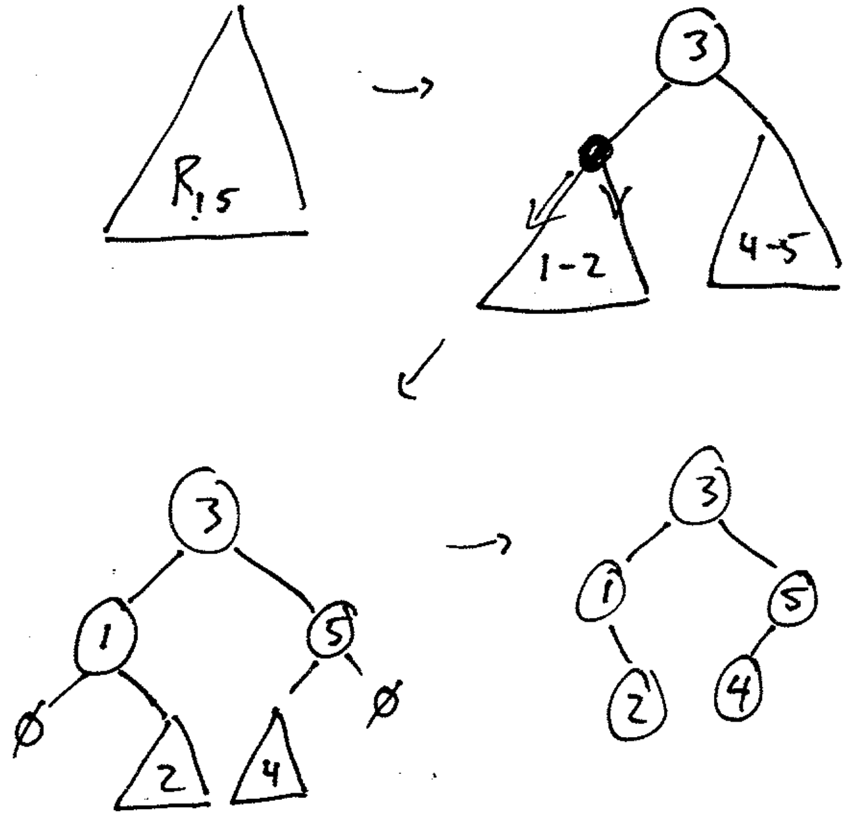
→ Constructing the OBST

→ Root Table:

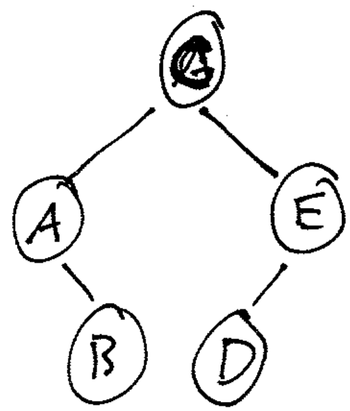
- Keeps track of the key that gave the best solution
- R_{ij} = root of the tree with keys $K_i \dots K_j$

i \ j	1	2	3	4	5
1	1	1	3	3	3
2		2	3	3	3
3			3	3	3
4				4	5
5					5

Root Table



	key	prob		# comps		
1	A	.213	·	2	=	.426
2	B	.020	·	3	=	.060
3	C	.547	·	1	=	.547
4	D	.100	·	3	=	.300
5	E	.120	·	2	=	.240
						1.573



Expected key comparisons: 1.573

OBST_Construct

Input: keys $K_1 K_2 \cdots K_n$, Root table R

Output: the root of the OBST

(u, i, j)

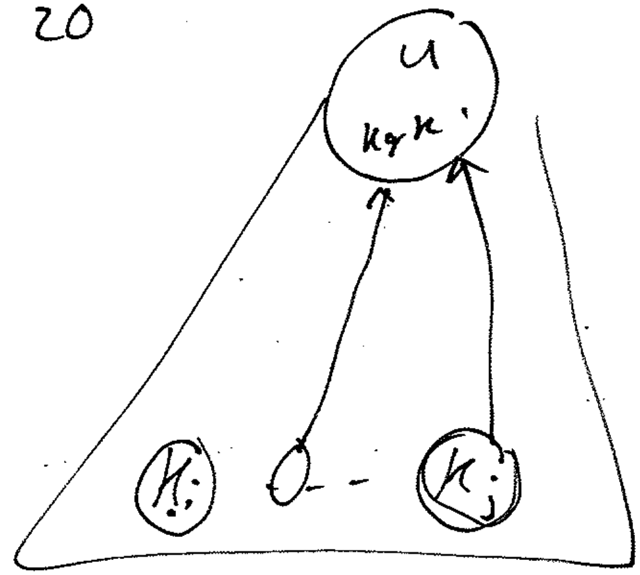
root of OBST containing $K_i \cdots K_j$

```
root.key ← R_{1,n}
S ← init stack
S.push(root, 1, n)  // triple: node, left, right index
while (S is not empty)
    (u, i, j) ← S.pop()
    k ← R_{ij}  // Root table lookup, key k of the node u
    if (i < k)
        // build the left sub tree
        v ← new node
        v.key ← R_{i,k-1}
        u.leftChild ← v
        S.push(v, i, k-1)
    if (k < j)
        // build the right tree
        v ← new node
        v.key ← R_{k+1,j}
        u.rightChild ← v
        S.push(v, k+1, j)
output root
```

$a, b = 10, 20$

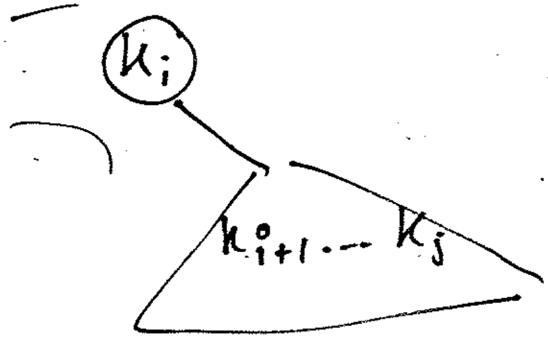
$a = 10$

$b = 20$



$k = k_j \Rightarrow$ no right tree

$k < k_j \Rightarrow$ is a right tree



$k_i = k \Rightarrow$ no left tree to build

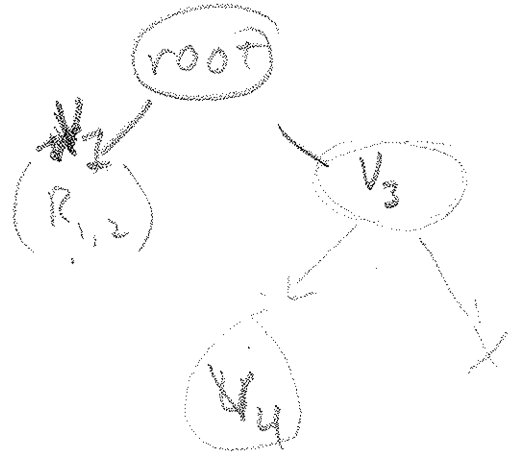
$k_i < k \Rightarrow$ exist a left tree to build

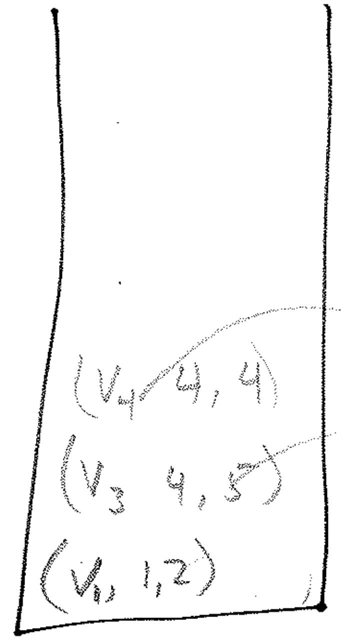
$k_i \ldots \cancel{k}-1$

$root.key \leftarrow R_{1,n}$

$= 3$

push(root, 1, n)





$u, i, j = root, 1, 5$

$k = R_{15} = 3$

$1 < 3$ true $\therefore$ $3 < 5$ true

$v.key = R_{1,2}$ $v.key = R_{4,5}$

push(v, 1, 2)

items	wt	val
a_1	5	10
a_2	2	5
a_3	4	8
a_4	2	7
a_5	3	7

$K = 7$

took $a_5 \longrightarrow 7 \longrightarrow 4$

taken $a_4 \longrightarrow 4 \longrightarrow 2$

a_3? No, leave it

$V_{1,3}$ = best solution consider elements $\{a_1\}$, $wt = 3$ $= 0$

$V_{1,4} = 0$

$V_{1,5} = 10$ vs 0

took a_2 $2 \to 0$

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10
2	0	0	5	5	5	10	10	15
3	0	0	5	5	8	10	13	15
4	0	0	7	7	12	12	15	17
5	0	0	7	7	12	14	15	19

$V_{1,5}$:

$V_{i-1,j}$ $\quad V_{0,5} = 0$

$V_{i-1,j-w_i}$ $\quad V_{0,5-5}$

$V_{00} = 0 + V_1$

$$V_{2,5} = \max \begin{cases} V_{i-1,j} = V_{1,5} = 10 \\ V_{i-1,j-w_i} + V_2 = V_{1,5-2} \end{cases}$$

$= V_{1,3} + 5$

$= 0 + 5 = 5$

$j - w_i = 4 - 4 = 0$

take a_3 or not

$V_{3,4} =$

$V_{i-1,j} = V_{2,4} = 5$

vs

$V_{i-1,j-w_i} = V_{2,0} + 8$

$= 0 + 8 = 8$

Construct a 0-1 Knapsack Solution:

Input: 0-1 Knapsack tableau V

Output: the optimal solution to 0-1 knapsack.

$S \leftarrow \emptyset$

$i \leftarrow n$

$j \leftarrow K$

while $i \geq 1$ and $j \geq 1$

row above = curr row
= you did not take item a_i

  while $i \geq 1$ and $V_{ij} = V_{i-1,j}$

    i --

  $S \leftarrow S \cup \{a_i\}$

  $j \leftarrow j - w_i$

  i --

output S

- The 0-1 Knapsack problem is NP-Complete
- <u>no</u> polynomial time algorithm is known to solve it
- it is <u>highly</u> unlikely that there is a polytime algo for it.

Dynamic Prog. Solution:

$O(n \cdot K)$ is k constant?

$\shortparallel$

$O(n \cdot 2^n)$

k could be constant

$O(n)$

k could be $O(n^k)$

$O(n \cdot n^k) = O(n^{k+1})$

K could be $O(2^n)$

$O(nk)$ = pseudopolynomial time

Radix Sort

8734
9472
1873
4217
9573
1247
1317

1873	4217	8734	9472
1247			9573
1317			

1247
1317
1873

$O(nk)$ pseudolinear

k = max. of digits in any of the numbers

k <u>may</u> be constant

$K = n \Rightarrow O(n^2)$

Complexity:

~~is~~ can a computer solve any problem?

~~Can any problem be decide.~~

can any language be decided by a Turing Machine

NO

0-1 Knapsack Problem $O(2^n)$

Given: a set of items $A = \{a_1 \ldots a_n\}$

with values $v_1 \ldots v_n$

and weights $w_1 \ldots w_n$

and a capacity K

Find a subset $S \subseteq A$ that

maximizes your total value: $\sum_{a \in S} val(a)$

subject to $\sum_{a \in S} wt(a) \leq K$

V_{ij} = value of the best solution to 0-1 knapsack considering elements $a_1 \dots a_i$ with a weight capacity of j

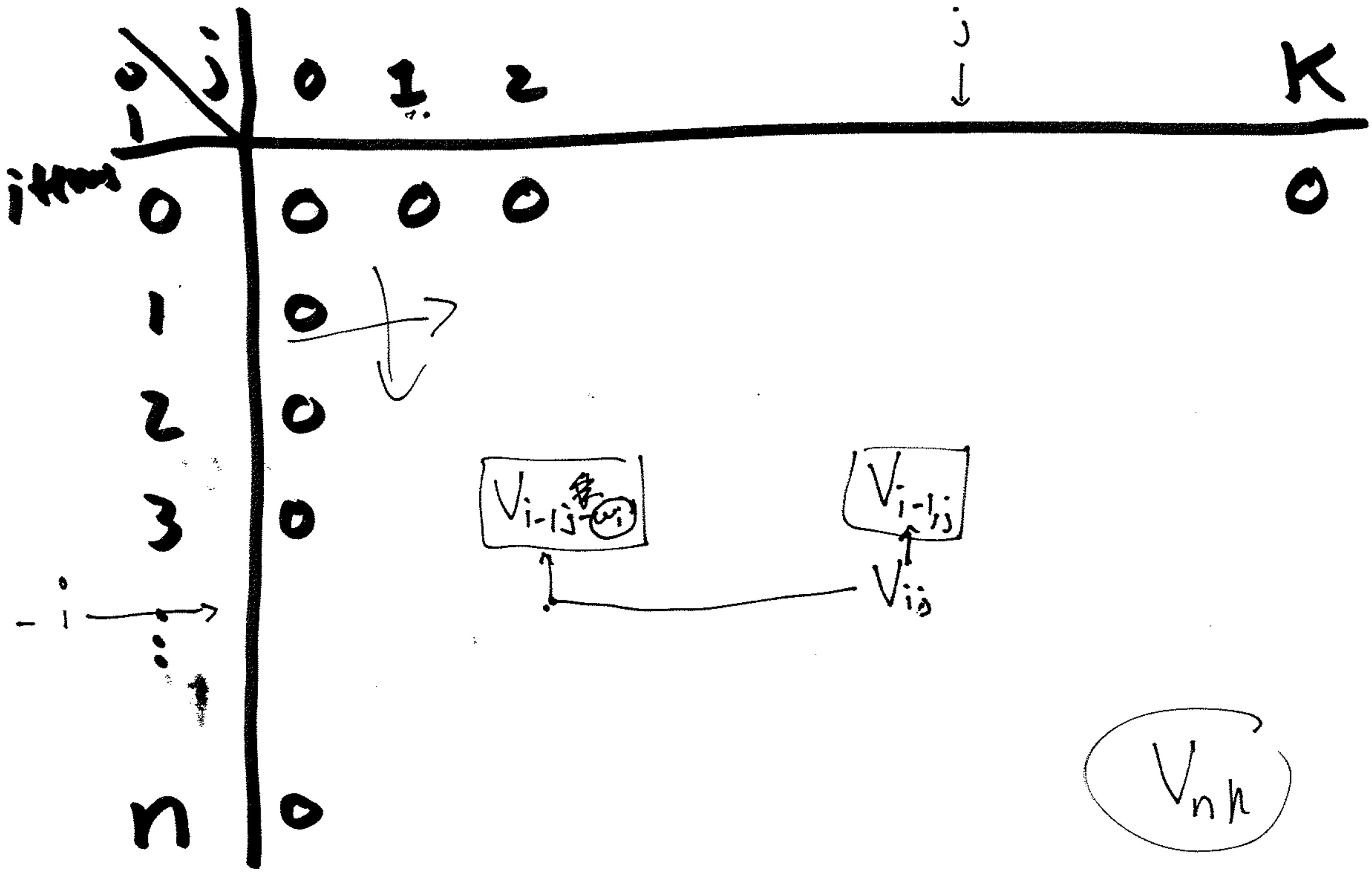
$V_{2,5}$ = best solution of subsets of $\{a_1, a_2\}$, $K \leq 5$

$V_{n,K}$ = find solution

$V_{0,j}$ = solutions of no elements = 0

$V_{i,0}$ = solutions with zero capacity

= 0



i \ j
items
0
1
2
j
K
0
1
2
3
i
n
V i-1,j-wi
V i-1,j
V ij
V nk

$$V_{ij} = \begin{cases} \max\overbrace{\left\{ \underbrace{V_{i-1,j}}_{\text{leave } a_i},\ \underbrace{V_{i-1,j-w_i}}_{\text{add } a_i} + v_i \right\}}^{\text{choice}} & \text{if } j-w_i \geq 0 \\ \underbrace{V_{i-1,j}}_{\text{no choice, you cannot take } a_i} & \text{if } j-w_i < 0 \end{cases}$$

Consider:

a) is taking element a_i possible?

$V_{i,j} =$

j = current weight capacity

w_i = weight of a_i & weight w_i

$\boxed{j - w_i \geq 0}$ => yes you can take it → value increases v_i

$\boxed{j - w_i < 0}$ => NO value does not increase

b) if it is feasible: $j - w_i \geq 0$

Choice:

take a_i: $V_{ij} = v_i + V_{i-1,\,j-w_i}$

leave a_i: $V_{ij} = V_{i-1,\,j}$

// base cases:

for $j = 0 \ldots K$

$\quad V_{0,j} = 0$

for $i = 0 \ldots n$

$\quad V_{i,0} = 0$

for $i = 1 \ldots n$

$\quad$ for $j = 1 \ldots K$ ← capacity

$\quad\quad$ if $(j - w_i) \geq 0$

$\quad\quad\quad V_{ij} = \max\{V_{i-1,j}\ \ V_{i-1,j-w_i}\}$

$\quad\quad$ else

$\quad\quad\quad V_{ij} = V_{i-1,j}$