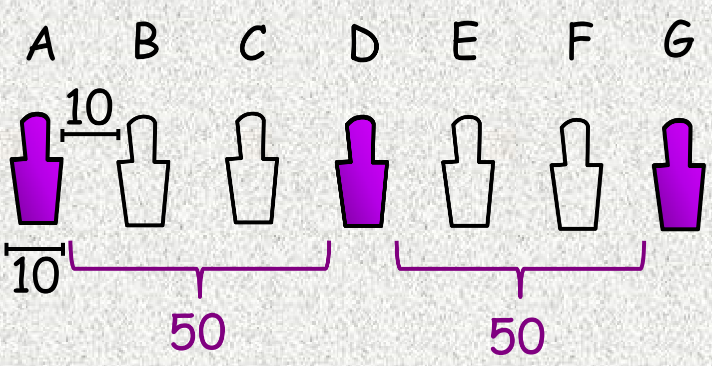
	$C^* = 100$	$\lvert C - C^* \rvert$	ρ	ε
min	$C = 120$	20	?	?

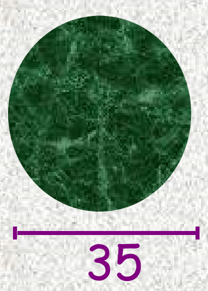
	$C^* = 100$	$\lvert C - C^* \rvert$	ρ	ε
max	$C = 80$	20	?	?

Note: $\rho \geq 1$ and $\varepsilon \geq 0$
(larger value means larger inaccuracy)

Establish a lower bound on an optimal solution

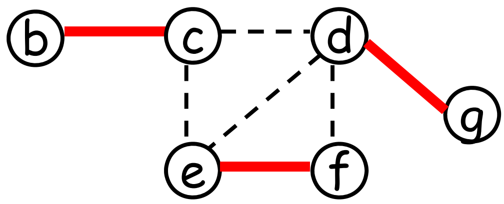
Idea: an independent set implies a lower bound





should be 21, I know

$G \supseteq$ A: three disjoint edges

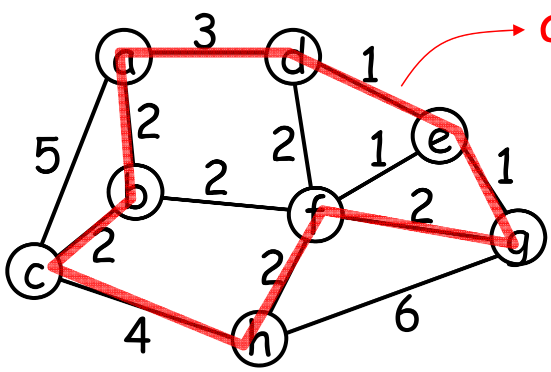


A needs at least $|A|$ = 3 vertices

$\Rightarrow$ G needs at least $|A|$ = 3 vertices

$\Rightarrow$ $|C^*| \geq |A|$ — ② (a lower bound on C^*)

The TSP Problem

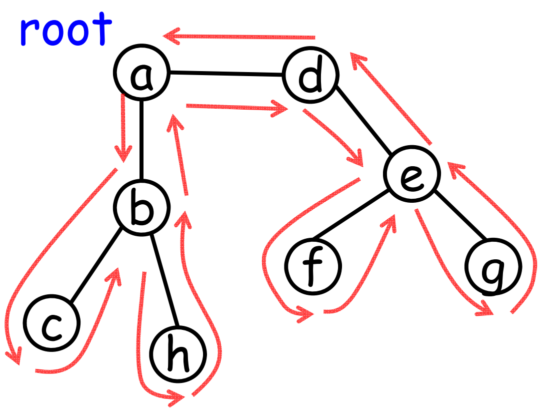


a TSP tour

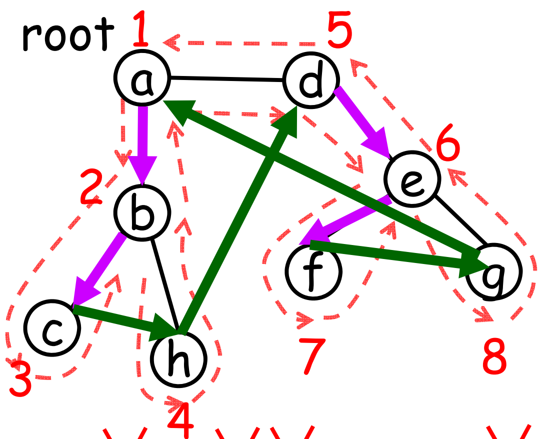
hamiltonian cycle

① visit each vertex exactly once

② minimum total length



full walk W=(a, b, c, b, h, b, a, d, e, f, e, g, e, d, a)



full walk W=(a, b, c, ~~b~~, h, ~~b~~, ~~a~~, d, e, f, ~~e~~, g, ~~e~~, ~~d~~, a)

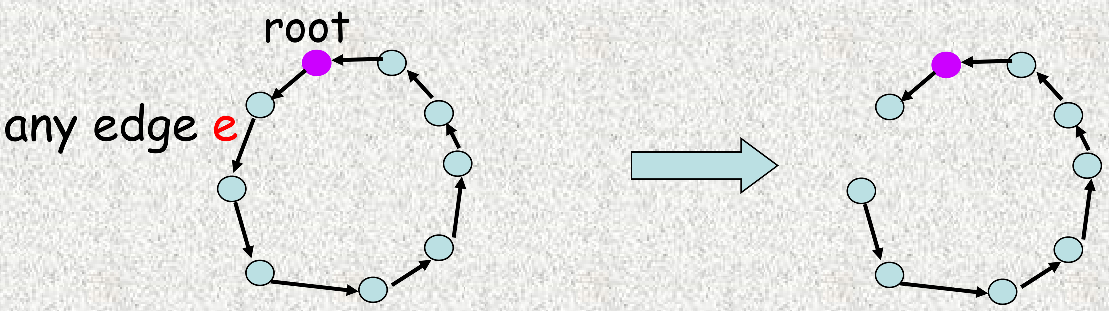
1 2 3 4 5 6 7 8 1

H=(a, b, c, h, d, e, f, g, a)

pre-order traversal on T

Establish a lower bound on an optimal TSP tour H^*

Idea: an MST implies a lower bound



an optimal TSP tour H^*
(n edges)

a path $P = H^* - \{e\}$
(n-1 edges)

$|H^*| \geq |P|$

$|P| \geq |T|$ (since P is a tree and T is MST)

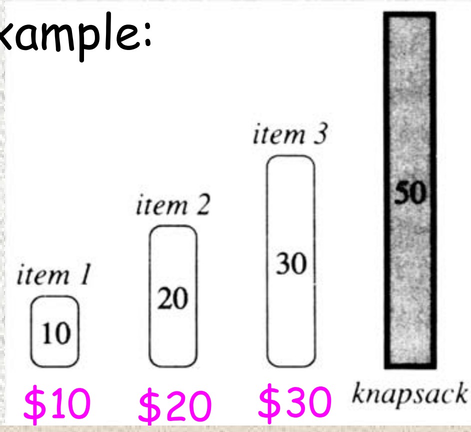
⇒ ① $|H^*| \geq |T|$ (a lower bound on H^*)

0-1 knapsack problem (integer):

Input: n items with weight w_i and value v_i
capacity C

Output: a subset of items with
weight $\leq C$ and maximum value

Example:



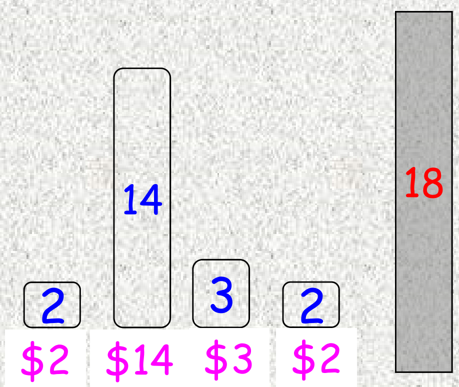
special case, in which $v_i = w_i$

Subset-Sum problem (integer):

Input: a set of n integers x_i
target t

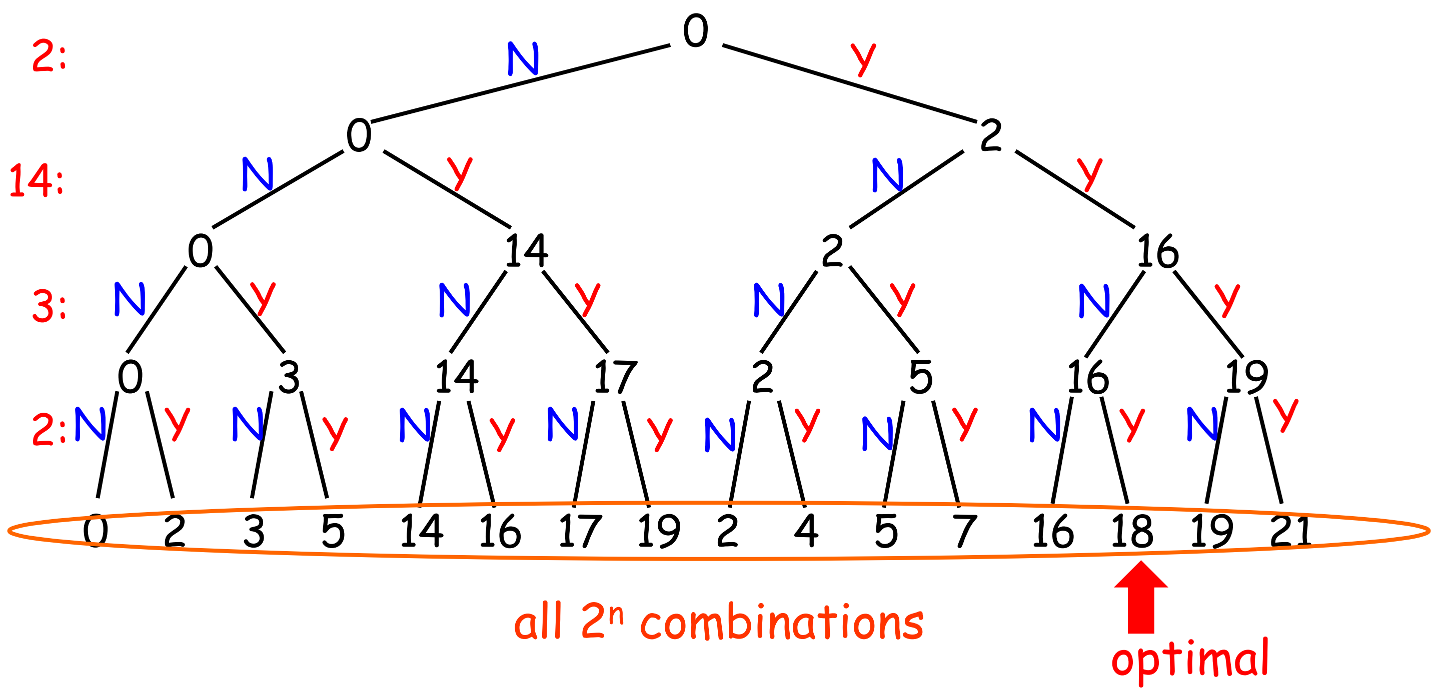
Output: a subset of integers whose
sum $\leq t$ and is maximum

Example: $S = \{2, 14, 3, 2\}$, $t = 18$ ⇒



t = 18

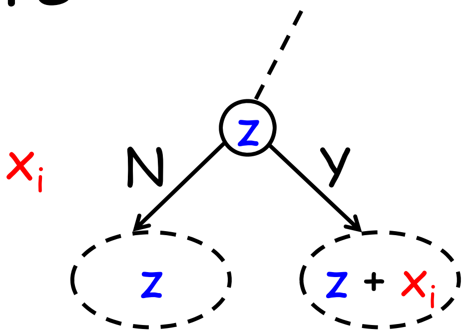
S = (2, 14, 3, 2)



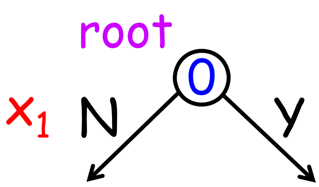
Brute-Force-DFS

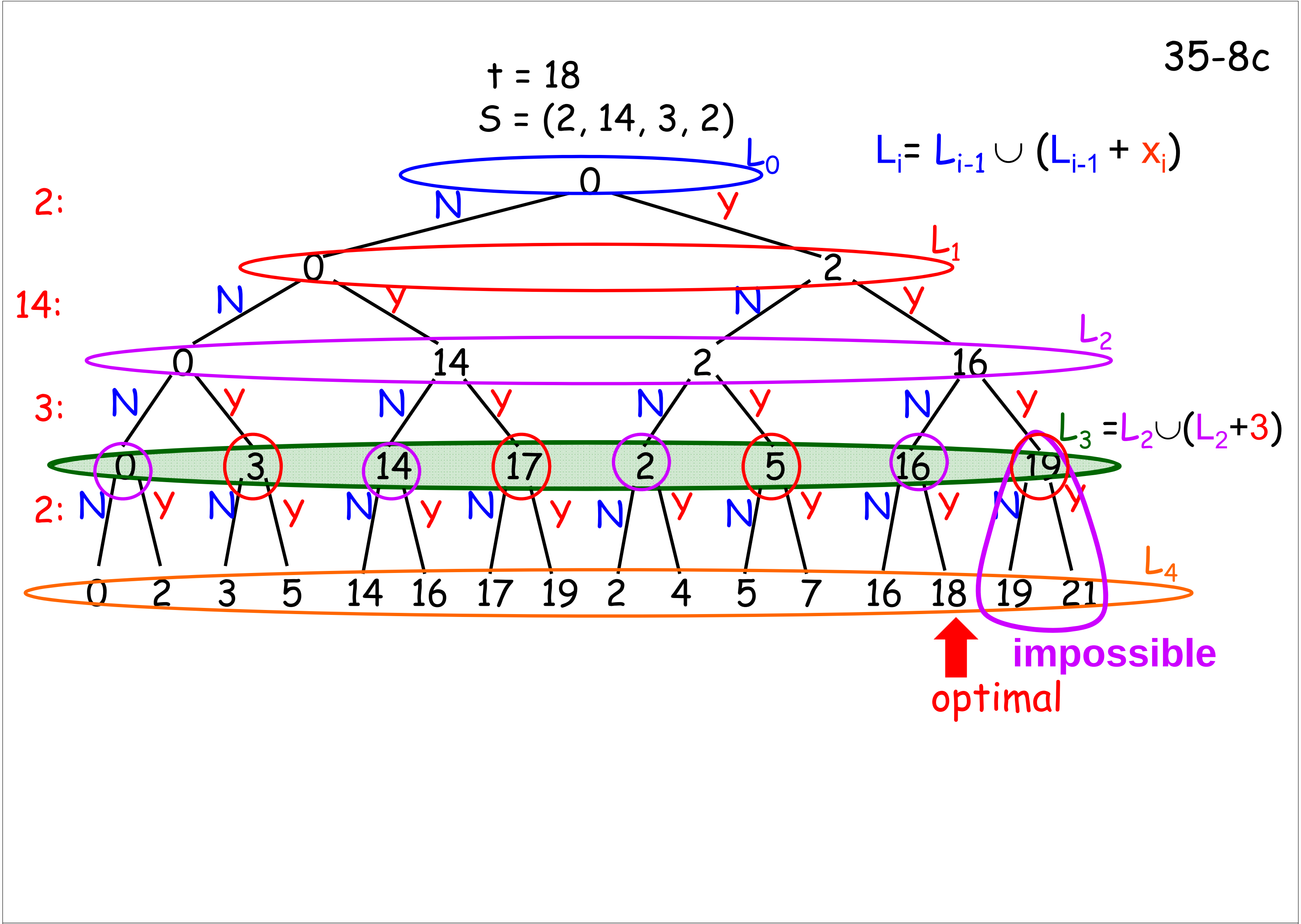
```
DFS(i, z)
begin
    if i ≤ n then
        DFS(i + 1, z)
        DFS(i + 1, z + x_i)
    else /* z is a leaf-candidate
        z* = better(z*, z)
end
```

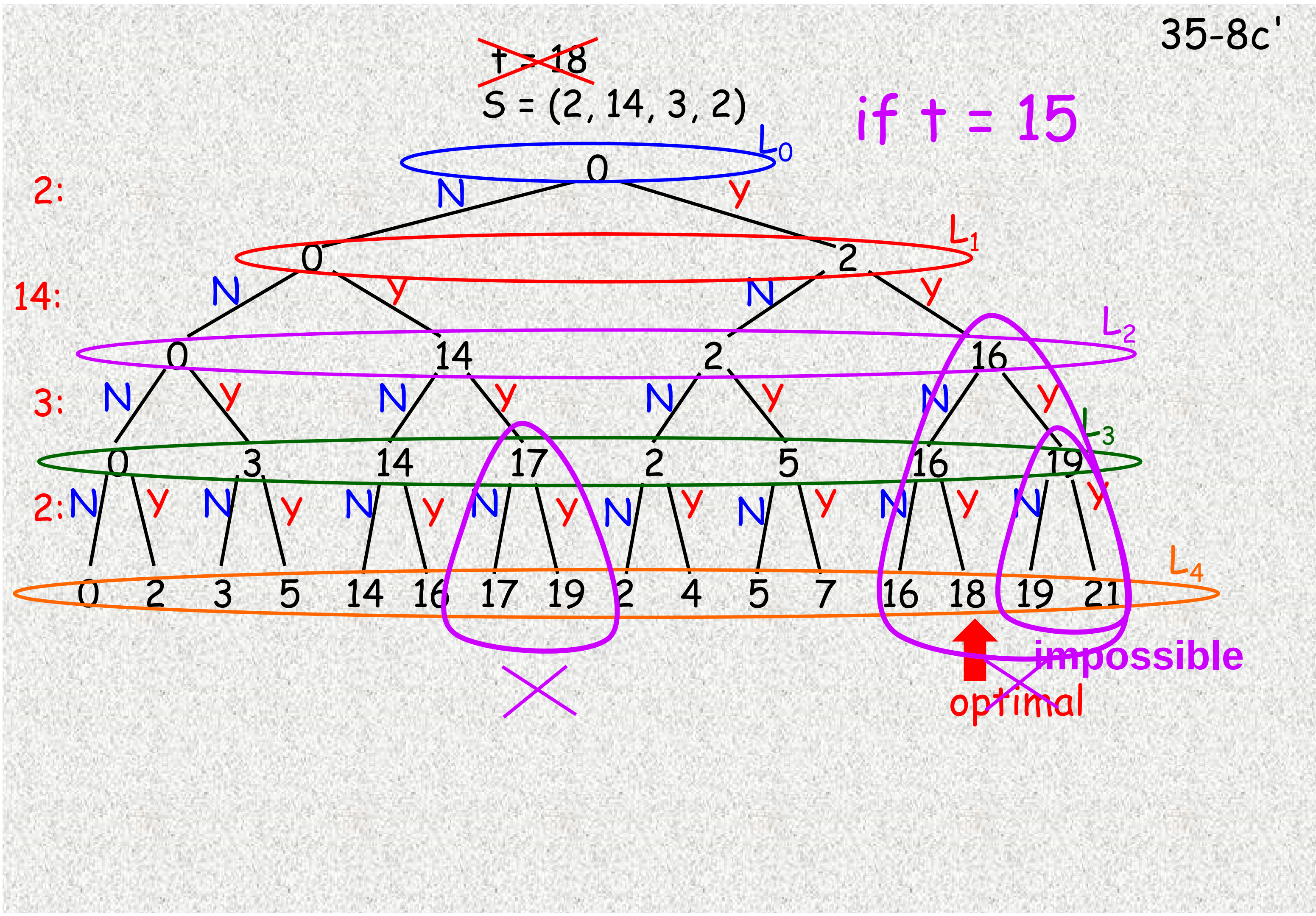
also check valid or not



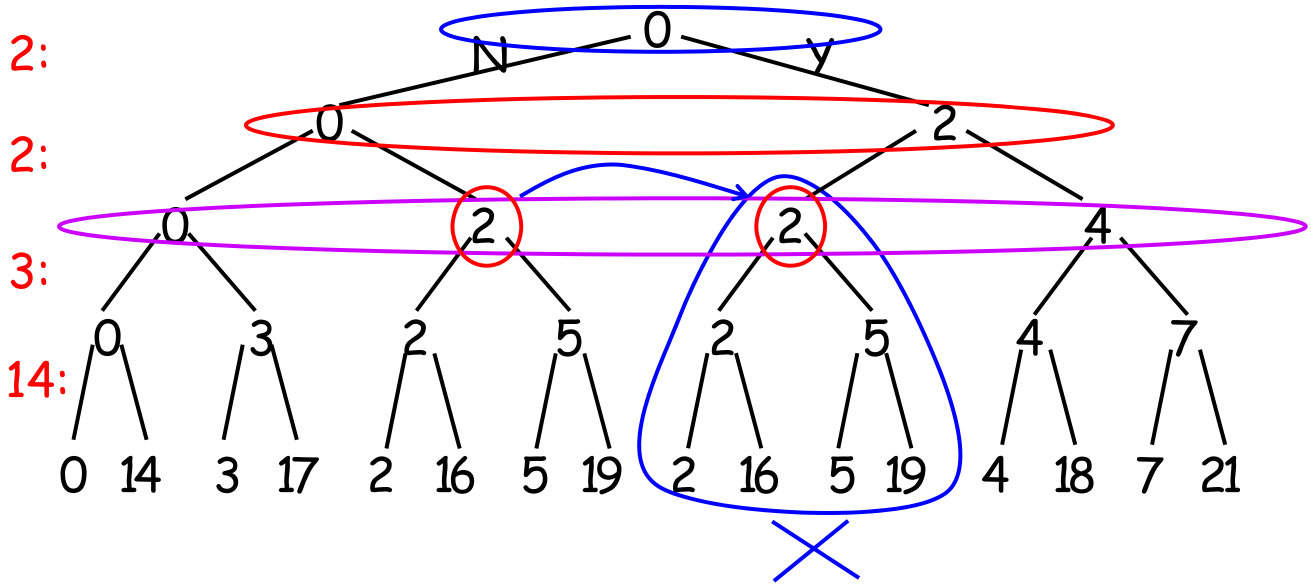

```
SubsetSum(S = (x_1, x_2, ..., x_n), t)
begin
    z* = 0
    DFS(1, 0)
    Output z*
end
```




35-8c
t = 18
S = (2, 14, 3, 2)
$L_i = L_{i-1} \cup (L_{i-1} + x_i)$
L_0
L_1
L_2
$L_3 = L_2 \cup (L_2 + 3)$
L_4
2:
14:
3:
2:
0
0 2
0 14 2 16
0 3 14 17 2 5 16 19
0 2 3 5 14 16 17 19 2 4 5 7 16 18 19 21
impossible
optimal


35-8c'
~~t = 18~~
S = (2, 14, 3, 2)
if t = 15
L_0
L_1
L_2
L_3
L_4
2:
14:
3:
2:
0
0 2
0 14 2 16
0 3 14 17 2 5 16 19
0 2 3 5 14 16 17 19 2 4 5 7 16 18 19 21
impossible
optimal

$S = (2, 2, 3, 14)$



L_i = Merge-List(L_{i-1}, x_i)   Time: $O(2|L_{i-1}|)$

Cut 1: 拿掉相同的

Example: $L_{i-1} = (0, 2, 3, 5, 14, 16, 17)$, $x_i = 2$
(sorted)

$L_i = L_{i-1} \cup (L_{i-1} + 2)$

$(L_{i-1} + 2)$

$O(2|L_{i-1}|)$

$= (0, 2, 3, 5, 14, 16, 17) \cup (2, 4, 5, 7, 16, 18, 19)$

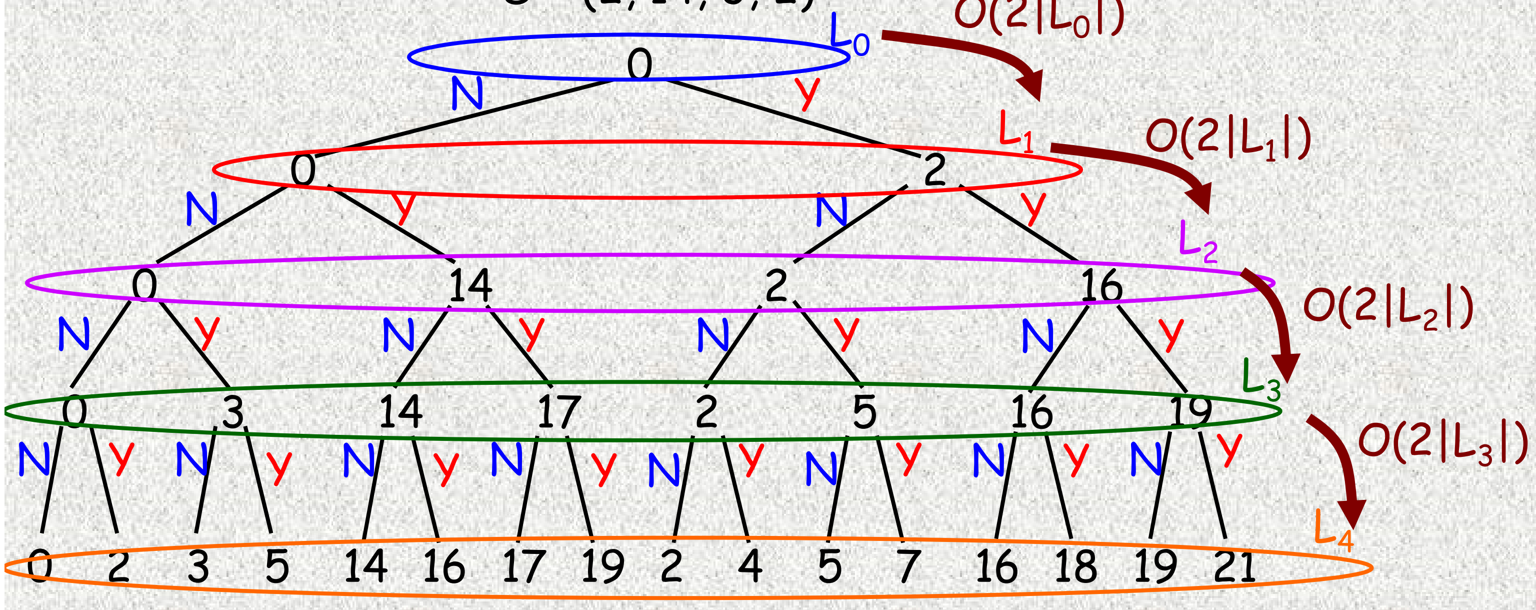
merge

$= (0, 2, \cancel{2}, 3, 4, 5, \cancel{5}, 7, 14, 16, \cancel{16}, 17, 18, 19)$

$= (0, 2, 3, 4, 5, 7, 14, 16, 17, 18, 19, 21)$
(sorted)

t = 18

S = (2, 14, 3, 2)



$$T(n) = 2|L_0| + 2|L_1| + 2|L_2| + 2|L_3| = \sum_{i=0}^{n-1} 2|L_i|$$

35-9x

35-9a

$$T(n) = \sum_{i=0}^{n-1} 2|L_i| \qquad L_i = (l_1, l_2, \ldots, \underset{\underset{100}{\wedge|}}{l_k})$$

all l_j are distinct and $l_k \le t = 100$ $\Rightarrow |L_i| \le \underset{\underset{101}{\wedge|}}{l_k + 1}$

① $|L_i| \le 2^i$

$T(n) = 2^0 + 2^1 + 2^2 + \ldots + 2^{n-1}$
$= O(2^n)$

② $l_k \le t \Rightarrow |L_i| \le t+1$

$T(n) = O(nt)$

③ $l_k \le \underbrace{sum(S)}_{} \overset{}{=} W \Rightarrow |L_i| \le W+1$

$T(n) = O(nW)$

④ let $m = \max(S) \Rightarrow W \le nm$
$\Rightarrow |L_i| \le nm+1$

$T(n) = O(n \times nm) = O(n^2 m)$

T(n) is polynomial if one of t, W, m is polynomial !

⇨ T(n) is pseudo-polynomial! (t, W, m may be ∞)

Pseudo-Polynomial:

If time is in the numeric value of an integer x,
we consider x as a $\lg_2 x$-bit integer (input size)

e.g. x = 60000, s = lg x = 16 bits

Example
input: N
output: IsPrime(N)

input siz : $s = \lg_2 N$

Algorithm 1:
$O(N) = O(2^s)$
exponential in s
pseudo-polynomial

Algorithm 2:
$O(N^{1/2}) = O(2^{s/2})$
pseudo-polynomial

Example
input: a, X
output: X^a

input siz : $s = \lg_2 a$

Algorithm 1:
$O(a) = O(2^s)$
exponential in s
pseudo-polynomial

Algorithm 2:
$O(\lg a) = O(s)$
polynomial

Pseudo-Polynomial:

polynomial in the numeric value of an integer
(exponential in the length (# of bits) of the integer)

The subset sum problem ($S = \{x_1, ..., x_n\}, t$)

* Consider $s = \lg t$ as the "input size" of t. (t is an s-bit integer)

e.g. t = 60000, s = lg t = 16 bits

* $T(n) = O(nt) = O(n2^s)$ is exponential in s (pseudo-polynomial)
* $A(n) = O(n^2 \log t) = O(n^2 s)$ is polynomial (in n and s)

Examples:

pseudo-polynomial

Counting sort - $O(n + k)$
Knapsack - $O(nC)$
GCD- $O(b)$
X^a- $O(a)$

polynomial

GCD- $O(\lg b)$
X^a- $O(\lg a)$

$$T(n) = \sum_{i=0}^{n-1} 2|L_i|$$

① $T(n) = O(2^n)$　　③ $T(n) = O(nW)$

② $T(n) = O(nt)$　　④ $T(n) = O(n^2m)$

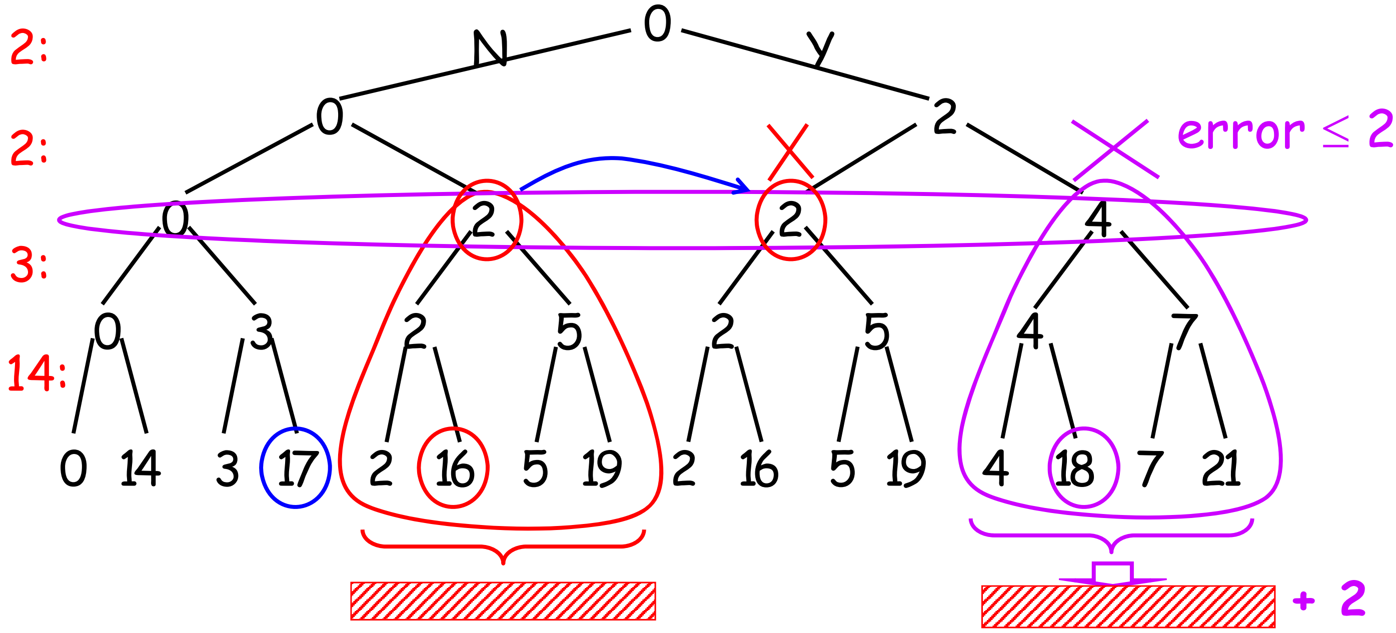
T(n) is polynomial if one of t, W, m is polynomial !

⇨ T(n) is pseudo-polynomial! (t, W, m may be ∞)

$$T(n) = O(\min\{2^n, nt, nW, n^2m\})$$

Note: 2^n may be the best.
(e.g., $n = 10$, $t = 10^{100}$, $W = 3 \times 10^{100}$, $m = 10^{99}$)

$S = (2, 2, 3, 14)$

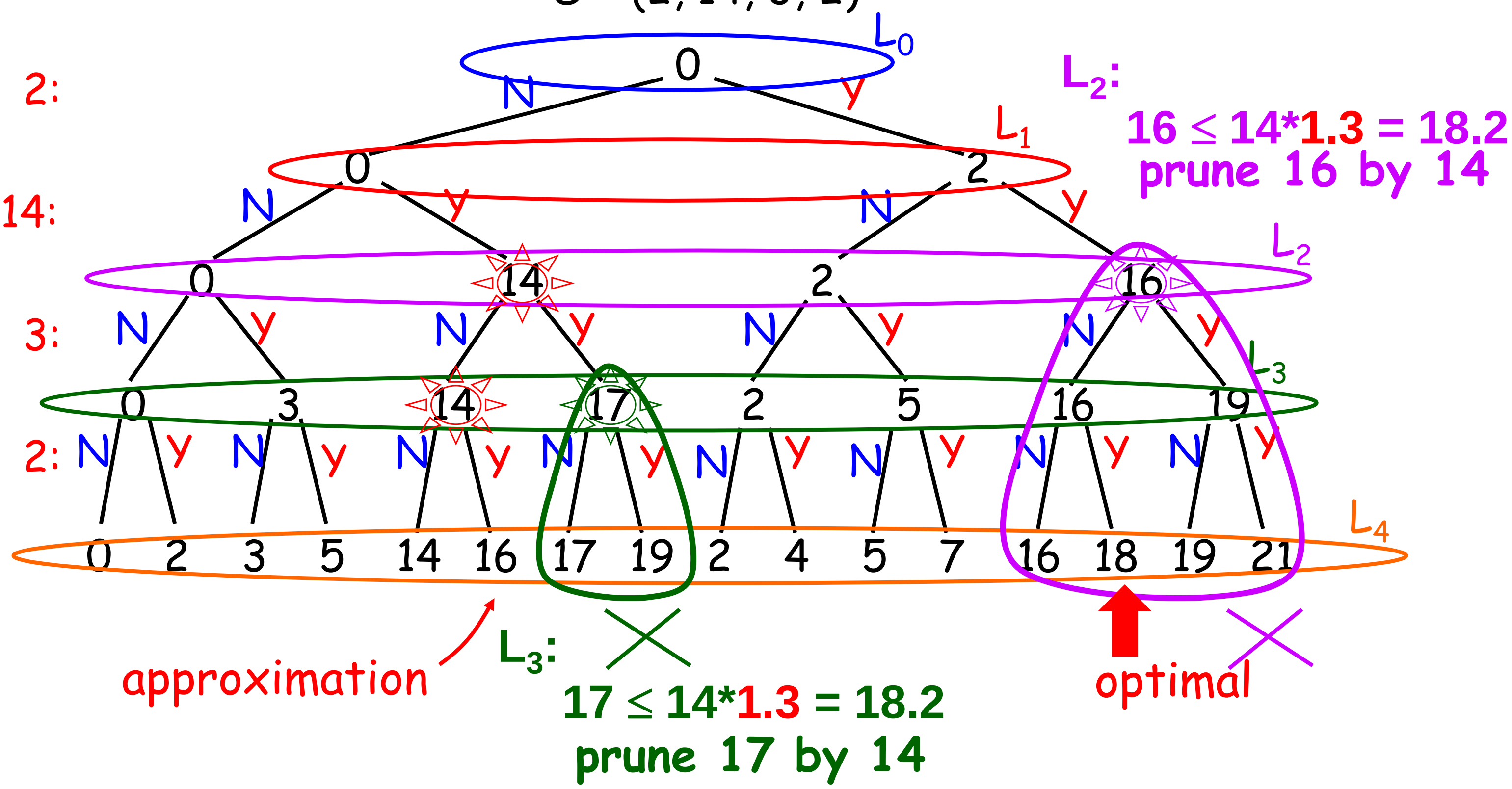


e.g., $t = 18 \Rightarrow$ error ≤ 2　$\Rightarrow$ error = 1

$\Rightarrow$ ~~error~~ = 2

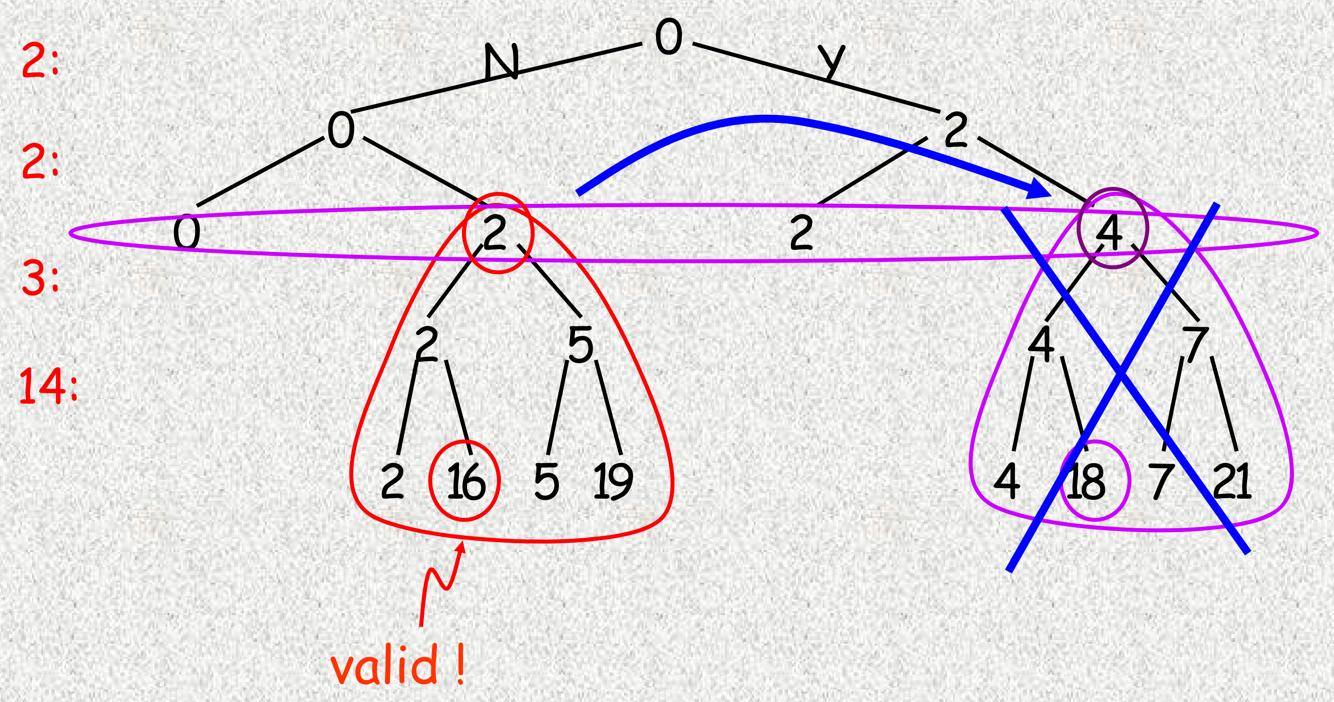
$t = 18$
$S = (2, 14, 3, 2)$

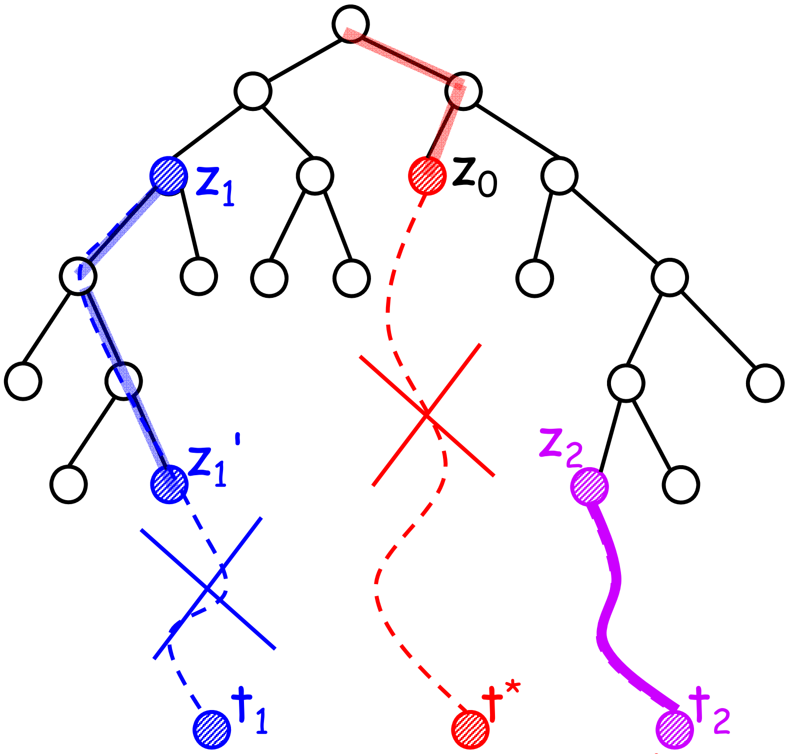
$\delta = 0.3$



Question: Can we prune 14 by 16 at L_2?

Assume $t = 18$ and "4" is cut.





$z_0 \le z_1(1+\delta)$

error $\le \delta \times z_1$

$z_1' \le z_2(1+\delta)$

error $\le \delta \times z_2$

total error $\le \delta \times (z_1 + z_2)$

Note: z_1 and z_2 are valid ($\le t$)

$z_k \leftarrow \cdots \leftarrow z_3 \leftarrow z_2 \leftarrow z_1 \leftarrow z_0$

Step i:

seed for t^*

$L_i = \{ \_, \_, \ldots, \square, \square, \ldots \}$

trim

$L_i = \{ \_, \_, \ldots, \square, \ldots \}$

error $\le \delta \times \square$

Step j:

$L_j = \{ \_, \_, \ldots, \square, \blacksquare, \ldots \}$

trim

$L_j = \{ \_, \_, \ldots, \square, \ldots \}$

error $\le \delta \times \square$

Note: all z_i are valid ($\le t$)

total error

$$\le \sum z'_{i-1} - z_i$$

$$\le \sum_{i=1}^{k} \delta \times z_i$$

$$\le \sum_{i=1}^{k} \delta \times t^* \qquad (z_i \le t^*)$$

$$\le k\delta t^*$$

$$\le n\delta t^* \qquad (k \le n)$$

$$\le t^* \varepsilon \qquad (\delta = \varepsilon / n)$$

(reason for this setting)

$X = (x_1, \quad x_2, \quad x_3, \quad x_4, \quad \ldots, \quad x_k)$

1. 從 1 開始，越來越大，但不得超過 1024

⇨ $|X| \leq$?

("=" when X= (1, 2, 3, ..., 1024))

2. 從 1 開始，每次至少成長 2 倍，但不得超過 1024

⇨ $|X| \leq$?

("=" when L_i = (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024)

$= (2^0, 2^1, 2^2, 2^3, \quad \ldots, \quad , 2^{\log_2 1024}))$

3. 從 1 開始，每次至少成長 1.5 倍，但不得超過 1024

⇨ $|X| \leq$?

("=" when $L_i = (1.5^0, 1.5^1, 1.5^2, 1.5^3, \quad \ldots, \quad , 1.5^{\log_{1.5} 1024}))$

$L_i = (y_1 = 0, \; y_2 \geq 1, y_3, \quad y_4, \quad \ldots \; , \quad y_k)$

Effect of two cuts: all y_j are distinct integers and $y_k \leq t$

(從 0 開始，越來越大，但不得超過 t)

⇨ $|L_i| \leq t + 1$

Example: t = 1024

⇨ $|L_i| \leq 1024 + 1$

("=" when L_i = (0, 1, 2, 3, ..., 1024))

$L_i = (y_1 = 0,\ y_2 \ge 1, y_3,\ \ y_4,\ \ \ldots\ ,\ \ \ y_k)$

Effect of trimming: y_j is at least $(1 + \delta) \times y_{j-1}$ for $j \ge 3$

(每次至少成長 $(1 + \delta)$ 倍，但不得超過 t)

$\Rightarrow\ |L_i| = k \le \lg_{(1+\delta)} t + 2$

Example: $t = 1024$, $(1 + \delta) = 2$

$\Rightarrow\ |L_i| \le \lg_2 1024 + 2 = 10 + 2$

("=" when $L_i = (0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024)$)

Example: $t = 1024$, $(1 + \delta) = 3$

$\Rightarrow\ |L_i| \le \lg_3 1024 + 2 = 6 + 2$

Example: $t = 1024$, $(1 + \delta) = 1.1$

$\Rightarrow\ |L_i| \le \lg_{1.1} 1024 + 2$

Q: Why O(nt)?

$L_i = (l_1 = 0, l_2 \geq 1\ , l_3,\ \ l_4,\ \ \ldots\ ,\ \ \ l_k)$

Effect of two cuts: all l_j are distinct integers and $t_k \leq t$

(從 0 開始，越來越大，但不得超過 t)

$\Rightarrow |L_i| \leq t + 1$

Example: $t = 1024 \Rightarrow |L_i| \leq 1024 + 1$

("=" when $L_i = (0, 1, 2, 3, \ldots, 1024)$)

$$\Rightarrow T(n) = \sum_{i=0}^{n-1} 2|L_i| = O(nt)$$

Q: P1: the knapsack problem;
P2: The subset sum problem

P2 is a special case of P1. (see 35-8)

In CH16, we solved P1 by DP. (see 16-3)

Can P2 be solved by DP?

If yes, why P2 is NP-H?

Q: P1: the knapsack problem;
P2: The subset sum problem

P2 is a special case of P1. (see 35-8)
- yes! Which is harder?

In CH16, we solved P1 by DP. (see 16-3)
- in O(nC), which is pseudo-polynomial.
- not solved!

Can P2 be solved by DP?
- yes, in O(nt), as the B&B algo

If yes, why P2 is NP-H?
- O(nt) is pseudo-polynomial

Q: Is P1, the knapsack problem, NP-H?

Optimization Problems

P
- * trivial
- * greedy
- * DP

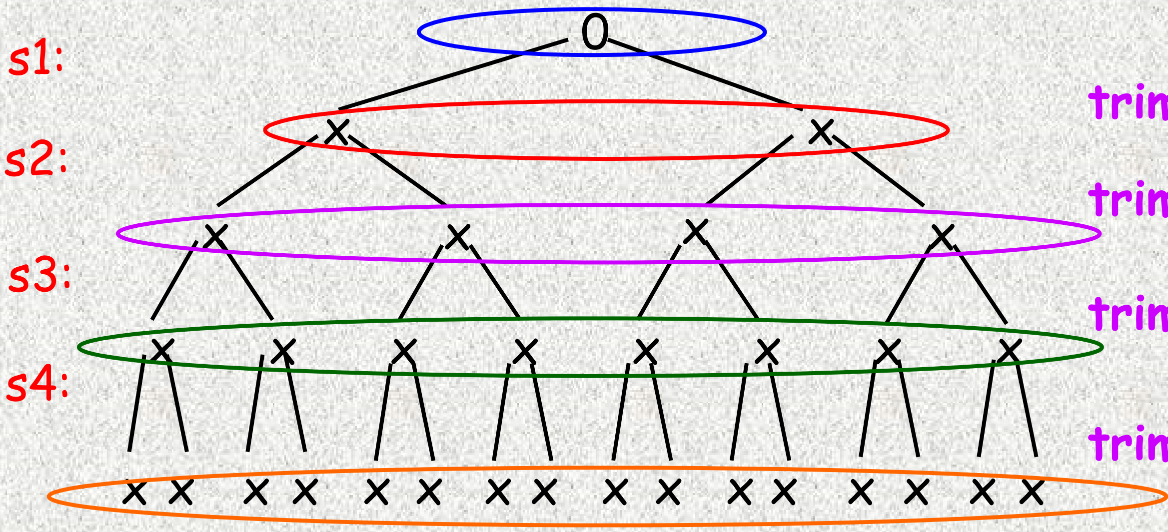
NP-hard
- * brute-force ($n \le 20$) ⇨ exact solution
- * B & B ($n \le 30$~100)
- * $n > 100$???
- * pseudo-polynomial (inputs are small integers)
- * approximation ⇨ near-optimal solution
- * heuristic

Remark: Pseudo-polynomial algorithms (DP, usually) are possible for NP-H problems (when inputs are small integers)

Q: Why $\delta = \varepsilon/n$?

Example: take δ = 5% for n = 4 and ε = 20%

S = (s1, s2, s3, s4)



trim: error ≤ 5%

trim: error ≤ 5%

trim: error ≤ 5%

trim: error ≤ 5%

total error ≤ 20%