---------------------------- 35 ----------------------------
Approximation Algorithms

Two approaches to NP-hard problems:

e.g. $O(2^n)$

(1) Exponential algorithms: (for small inputs)

- brute-force search
- branch-and-bound

(2) Near-optimal solutions: (polynomial time)

- approximation algorithms (with performance bounds)
- heuristic algorithms

Performance bounds (n is the input size)

e.g. $\rho(n)$ = 1.5 -> at most 1.5 times

ratio bound: $\begin{cases} C/C^* \le \rho(n) & \text{for minimization} \\ C^*/C \le \rho(n) & \text{for maximization} \end{cases}$

(Note that $\rho(n) \ge 1$.)

e.g. $\varepsilon(n)$ = 0.5 -> error within 50%

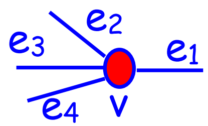
relative error bound: $\dfrac{|C - C^*|}{C^*} \le \varepsilon(n)$

(for both minimization & maximization)

$\varepsilon(n)$, $\rho(n)$ may be ① function of n (e.g., lg n, $n^{1/3}$) ② constant (e.g., 0.5, 2.6)

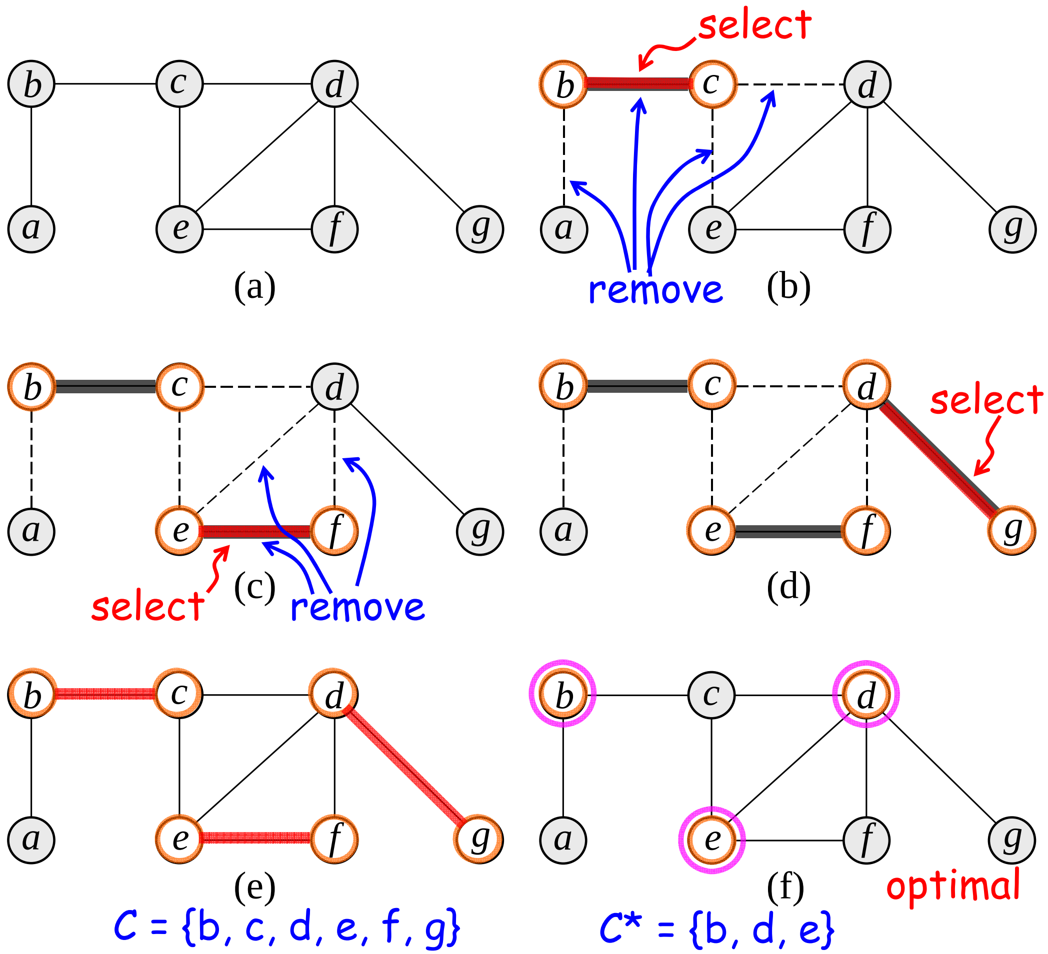
* For many problems, there are approximation algorithms with constant ratio bounds (relative error bounds), independent of n.

35.1 The vertex-cover problem



A ***vertex cover*** of an undirected graph $G=(V,E)$ is a subset C of V such that for each $(u, v) \in E$, either $u \in C$ or $v \in C$.

The ***vertex-cover problem*** is to find for G a vertex cover of minimum size. (an NP-hard problem)



(a) (b) (c) (d) (e) (f)

* (e): a vertex cover $C=\{b, c, d, e, f, g\}$
(f): the optimal vertex cover $C^*=\{b, d, e\}$

APPROX-VERTEX-COVER(G)

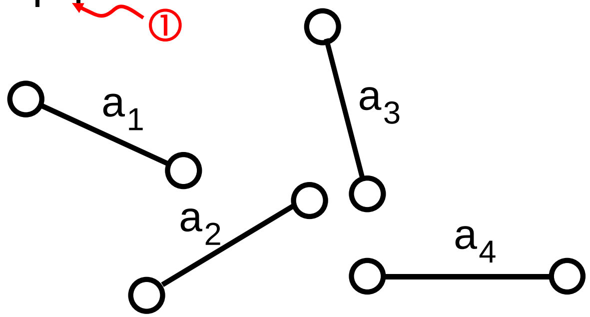
```
1  C ← Ø
2  E′ ← E[G]
3  while E′ ≠ Ø
4       do let (u, v) be an arbitrary edge of E′
5            C ← C ∪ {u, v}
6            remove from E′ every edge incident
                                 on either u or v
7  return C
```

uncovered edges

* Time: $O(E)$

$\varepsilon(n) = 1$
$C \le 2C^*$

Theorem 35.1: Approx-Vertex-Cover has $\rho(n) = 2$.
Proof: Let A be the set of edges picked in Line 4. Since no two edges in A share an endpoint, we have $|C|=2|A|$. ①



35-4xa

A needs at lest |A| vertices
C* covers A (A is a subgraph of G)

Let C^* be an optimal cover. C^* should cover A. That is, C^* should include at least one endpoint of each edge in A. Since no two edges in A share an endpoint, we have $|C^*|\geq|A|$ (=$|C|/2$) and thus $C/C^*\leq 2$. Q.E.D.

①+② ②

35-4b

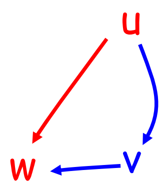
35.2 The traveling-salesman problem (NP-C)

35.2.1 The Euclidean TSP problem (NP-C)

①complete ② non-neg., triangle ineq.

The ***Euclidean traveling-salesman problem*** is to find in a ①*complete* weighted undirected graph $G=(V,E)$ a hamiltonian cycle (a tour) with minimum cost. The edges weights $c(u,v)$ are nonnegative integers. And, the weight function satisfies the following ②***triangle inequality:***

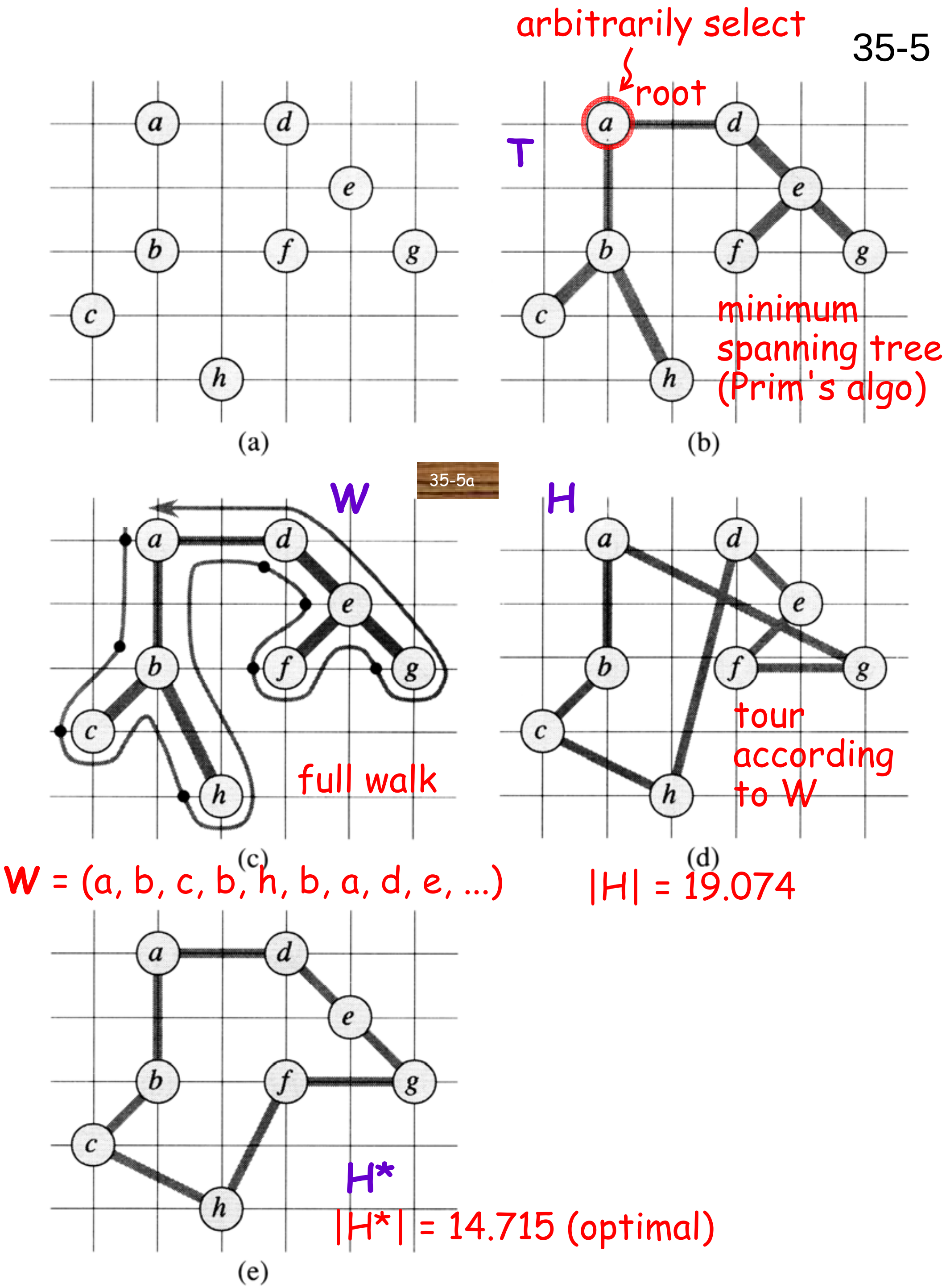
$$c(u,w) \leq c(u,v)+c(v,w).$$



APPROX-TSP-TOUR(G, c)

```
select a vertex r ∈ G.V to be a "root" vertex
compute a minimum spanning tree T for G from root r
     using MST-PRIM(G, c, r)
let H be a list of vertices, ordered according to when
     they are first visited in a preorder tree walk of T
return the hamiltonian cycle H
```

W



* (b): *T*: a minimum spanning tree *T*

O(V) { (c): *W*: a full walk of *T*

(d): *H*: a tour of length 19.074

(e): *H**: an optimal tour of length 14.715

Time: $O(E)=O(V^2)$ — complete graph

Prim's algo { $E + V^2$ (unsorted array); $E \lg V$ (binary heap); $E + V \lg V$ (Fib. heap)

Why not Kruskal's MST algo?

Theorem 35.2: Approx-TSP-Tour has $\rho(n) = 2$. $H \le 2H^*$

Proof: Let *T* be a minimum spanning tree. Deleting any edge from *H**, we can obtain a spanning tree. Thus, $|T| \le |H^*|$. ①

35-6x

A ***full walk***, denoted by *W*, of *T* lists the vertices when they are first visited and also whenever they are return to after a visit to a subtree. In our example,

$$W=(a, b, c, b, h, b, a, d, e, f, e, g, e, d, a).$$

1 2 3 4 5 6 7 8 1

② Clearly, $|W|=2|T|$. Thus, $|W| \le 2|H^*|$.

Note that *W* is not a tour. It visits a vertex more than once. However, by triangle inequality, we can delete unnecessary visits to a vertex without increasing the cost to obtain *H*. (In our example, *H*=(*a*, *b*, *c*, *h*, *d*, *e*, *f*, *g*).) Thus, $|H| \le |W| \le 2|H^*|$. ③ triangle ineq.

Q.E.D.

35.2.2 The general TSP problem

Current: 0.814 lg **n**

Without triangle inequality, an approximation algorithm with constant ratio bound does not exist unless P = NP.

$\in$ NP-C

35.5 The subset-sum problem

Approximation scheme: an approximation algorithm takes as input not only an instance of the problem, but also a constant relative error bound $\varepsilon > 0$.

part of input

* T(n) is a function of n & ε
(e.g., $(1/\varepsilon)2^{n/3}$, $n^{3/\varepsilon}$, $(1/\varepsilon)^2 n^3$, $2^{n/\varepsilon}$)

Polynomial-time approximation scheme: an approximation scheme runs in $O(n^k)$ time, where k is a constant. (e.g., $O(n^{3/\varepsilon})$.) polynomial for fixed ε
$O((1/\varepsilon)^2 n^3)$

和ε無關

Fully polynomial-time approximation scheme: an approximation scheme runs in $O((1/\varepsilon)^c n^k)$ time, c and k are constants. (e.g., $O((1/\varepsilon)^2 n^3)$ time)
polynomial in 1/ε, n

* ε 減小，只影響 constant factor

The subset-sum problem:

$S' \subseteq S$, sum(S') = t

Decision version: Given a set S of positive integers and an integer t, determine whether there is a subset of S that adds up exactly to the target t.

$S' \subseteq S$, sum(S') $\le$ t, sum(S') is max

☆ *Optimalization version:* find a subset of S whose sum is as large as possible but not larger than t.

* special case of the knapsack problem:
($C = t$, all $v_i = w_i$) $\Longrightarrow$ O(nC) by DP (Ex. 16.2-2, see 16-3ab)

35-8x

An exponential-time algorithm

35-8a

(A branch&bound algo - BFS)

L_i: all combinations of $\{x_1, x_2, \ldots x_i\}$ (sorted)

Cut 1: 拿掉相同的

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

35-8y

```
EXACT-SUBSET-SUM(S, t)
n ← |S|
L0 ← ⟨0⟩
for i ← 1 to n
      do Li ← MERGE-LISTS(L(i−1), L(i−1) + xi)
            remove from Li every element that is
                                    greater than t
return the largest element in Ln
```

Cut 2: 拿掉不合法的

Example: Let S=(2, 2, 14, 3) and t = 15.

$\Rightarrow L_0$=<0>

<0>$\cup$(<0>+2)=<0,2> $\Rightarrow L_1$=<0,2>

$L_0 \cup (L_0 + 2)$ (sorted)

拿掉相同的

L_1 $L_1 + 2$

<0,2>∪(<0,2>+2)=<0,2,~~2~~,4> ⇒ L_2=<0,2,4>

L_2∪(L_2+14)=<0,2,4,14,~~16~~,~~18~~> ⇒ L_3=<0,2,4,14> 拿掉不合法的

L_3∪(L_3+3)=<0,2,3,4,5,7,14,~~17~~>

⇒ L_4=<0,2,3,4,5,7,14>

optimal

35-9x
35-9a

Time: $\sum_{0\le i\le n-1} 2|L_i| = O(2^n)$. Note that $|L_i|=O(2^i)$. ①

e.g. $t = O(n^2)$, $L_i = O(t)$, $\sum L_i = O(nt) = O(n^3)$

②* In case *t* is polynomial in *n*, we have $|L_i|=O(t)$. Thus, the algorithm performs in polynomial time.

e.g. $m = \max(S) = O(n^2)$, $L_i = O(nm)$, $\sum L_i = O(n^2m)$

④* In case all integers in *S* are bounded by a polynomial in *n*, the algorithm also performs in polynomial time.

pseudo-polynomial 35-9b 35-9z

A fully polynomial-time approximation scheme

35-9c

/delta/

To ***trim** a list L by δ* is to remove as many elements from *L* as possible, in such a way that if *L'* is the result of trimming *L*, then for every element *y* that was removed from *L*, there is an element *z*≤*y* still in *L'* such that

z → ~~y~~ $y \le z(1+\delta)$ $(y-z \le \delta z)$ → error ≤ δz

why removing y, not z?

(We can think of "*z* representing *y*" in *L'*.)

Example: 11 ≤ 10 × 1.1   21,22 ≤ 20 × 1.1

Let L = (10, 11, 12, 15, 20, 21, 22, 23, 24, 29).

If δ = 0.1, we have

L' = (10, 12, 15, 20, 23, 29).

Let $L = (y_1, y_2, \ldots, y_m)$. The following procedure trims *L* in $O(m)$ time.

TRIM(L, δ)

```
m ← |L|
L' ← ⟨y_1⟩
last ← y_1
for i ← 2 to m
    do if y_i > last·(1 + δ)
        then append y_i onto the end of L'
             last ← y_i
return L'
```

L'=(……, last)

"y_i > (1+δ) × last"?

L=(………, y_i, …………)

An approximation scheme ($0 < \varepsilon < 1$) ★

APPROX-SUBSET-SUM(S, t, ϵ)

```
n ← |S|
L_0 ← ⟨0⟩
for i ← 1 to n
    do L_i ← MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
       L_i ← TRIM(L_i, ε/n)
       remove from L_i every element that is greater than t
let z* be the largest value in L_n
return z*
```

拿掉相同的

e.g., ε = 0.2, n = 4 ⇒ δ = ε/4 = 0.05

δ = ε/n

拿掉很接近的

拿掉不合法的！

Example: Let S=<104, 102, 201, 101>, t=308, and ε = 0.2. We have $\delta = \varepsilon/4$ =0.05 and

$\delta = \varepsilon/n$

line 2: $L_0 = \langle 0 \rangle$,

$i = 1$

line 4: $L_1 = \langle 0, 104 \rangle$,

line 5: $L_1 = \langle 0, 104 \rangle$,

line 6: $L_1 = \langle 0, 104 \rangle$,

$i = 2$

line 4: $L_2 = \langle 0, 102, \cancel{104}, 206 \rangle$,

line 5: $L_2 = \langle 0, 102, 206 \rangle$, trim

line 6: $L_2 = \langle 0, 102, 206 \rangle$,

$i = 3$

line 4: $L_3 = \langle 0, 102, 201, \cancel{206}, 303, 407 \rangle$,

line 5: $L_3 = \langle 0, 102, 201, 303, \cancel{407} \rangle$, trim

line 6: $L_3 = \langle 0, 102, 201, 303 \rangle$, remove > t (308)

$i = 4$

line 4: $L_4 = \langle 0, 101, \cancel{102}, 201, \cancel{203}, 302, \cancel{303}, 404 \rangle$,

line 5: $L_4 = \langle 0, 101, 201, 302, \cancel{404} \rangle$, trim

line 6: $L_4 = \langle 0, 101, 201, 302 \rangle$. remove > t (308)

The answer is z^*=302, which is well within ε = 20%. (The optimal answer is 307 (=104+102+101).)

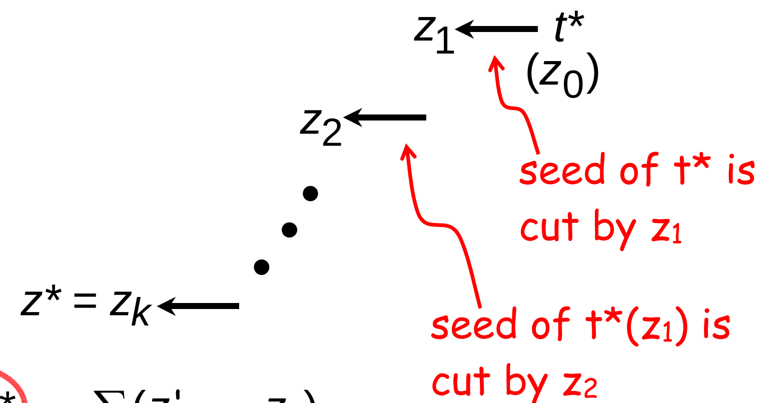
Theorem 35.8 Approx-Subset-Sum is a fully polynomial-time approximation scheme.

Proof:

(a) Clearly, the answer is legal. (not larger than t and being the sum of a subset).

(b) relative error bound is within ε:

35-12a



$$
\begin{aligned}
t^* - z^* &= \sum_{1 \le i \le k} (z'_{i-1} - z_i) \\
&\le \sum_{1 \le i \le k} \delta \times z_i \\
&\le k\delta t^* \quad \leftarrow z_i \le t^* \\
&\le n\delta t^* \quad \leftarrow k \le n \\
&\le \varepsilon t^* \quad \leftarrow \delta = \varepsilon/n
\end{aligned}
$$

(c) **fully polynomial-time:** Time = $\sum 2|L_i| = O(\sum |L_i|)$

35-12x

L_i: $y_1 = 0, \overset{\ge 1}{y_2}, y_3, \ldots, \overset{\le t}{y_k}$ (distinct integers)

Since $y_2 \ge 1$ and $y_i > y_{i-1} \times (1+\delta)$, we have

$y_k > (1+\delta)^{k-2}$. ①

$\begin{cases} y_k > (1+\delta)^{k-2} & ① \\ y_k \le t & ② \end{cases}$

Since $y_k \le t$, we have

$k\text{-}2 \le \log_{1+\delta} t$. Thus,

$$k = |L_i| \le \log_{1+\delta} t + 2$$

$$= \frac{\ln t}{\ln(1+\delta)} + 2 \qquad (\lg_a b = \frac{\lg_c b}{\lg_c a})$$

$$\le \frac{(1+\delta)\ln t}{\delta} + 2$$

(by (3.17), $\frac{x}{1+x} \le \ln(1+x)$ for $x > -1$)

$$\le \frac{n(1+\frac{\varepsilon}{n})\ln t}{\varepsilon} + 2 \qquad \delta = \frac{\varepsilon}{n}$$

$$\le \frac{2n\ln t}{\varepsilon} + 2 \quad (\text{by } \frac{\varepsilon}{n} < 1)$$

$\varepsilon < 1$ (See 35-10)

$$\text{Time} = O(\sum_{0 \le i \le n-1} |L_i|)$$

$$= O(n(\frac{2n\ln t}{\varepsilon} + 2))$$

$$= O(\frac{1}{\varepsilon} n^2 \log t)$$

polynomial or pseudo-polynomial?

fully polynomial

Q.E.D.

Homework: Ex. 35.1-4, 35.5-4.

Differences in the 3rd Edition

Approximation scheme: (1st) ε
(defined by relative error bound)

Given a parameter: α (0.6)
Goal: $\varepsilon = \alpha$ (0.6)
(or simply "Given ε")
(set $\delta = \varepsilon / n$) (0.06 for n = 10)

Approximation scheme: (2nd, 3rd)
Approximation Scheme: ρ
(defined by ratio bound)

Given a parameter: α (0.6)
Goal: $\rho = 1 + \alpha$ (1.6)
(for MAX, set $\delta = \alpha / 2n$) (0.03 for n = 10)
(In the textbook, ε is used to denote α)

```
APPROX-SUBSET-SUM(S, t, ε)
1  n ← |S|
2  L0 ← ⟨0⟩
3  for i ← 1 to n
4       do Li ← MERGE-LISTS(Li−1, Li−1 + xi)
5          Li ← TRIM(Li, ε/2n) ☆
6          remove from Li every element that is greater than t
7  let z* be the largest value in Ln
8  return z*
```