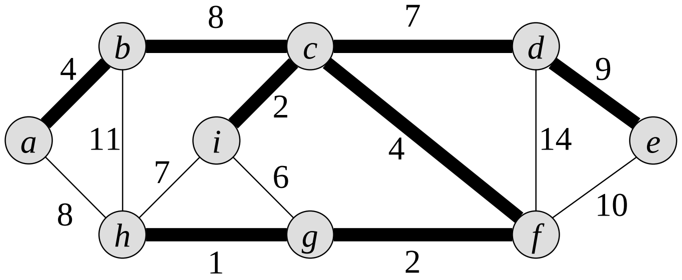
------------------------------ 23 ------------------------------

Minimum Spanning Trees

Input: A connected undirected graph *G*=(*V*, *E*)

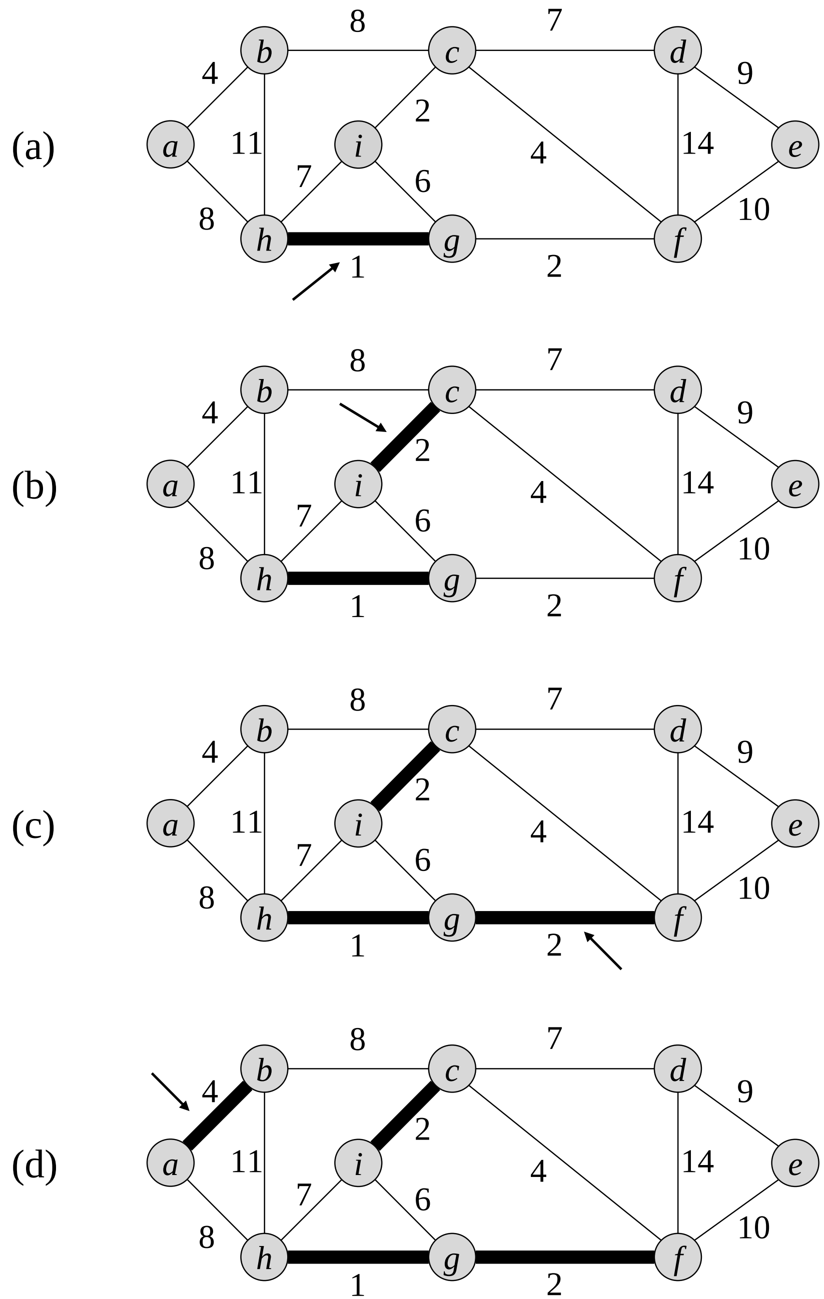
Output: A minimum spanning tree of *G*



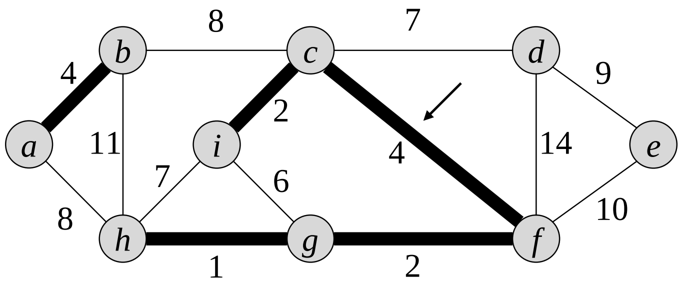
Two greedy algorithms: Managing a set *A* that is always a subset of some minimum spanning tree.

23.2 Kruskal's algorithm: smallest weighted first

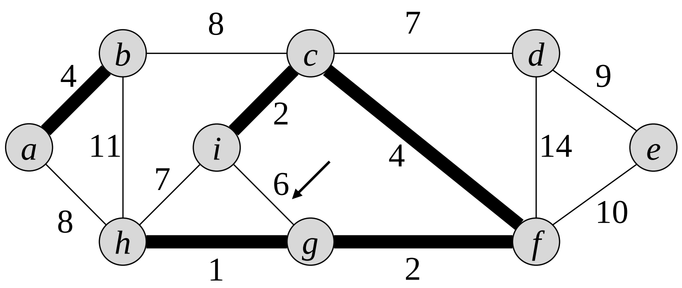
```
MST-KRUSKAL(G, w)
A ← Ø
for each vertex v ∈ V[G]
      do MAKE-SET(v)
sort the edges of E into nondecreasing order by weight w
for each edge (u, v) ∈ E, taken in nondecreasing order by weight
      do if FIND-SET(u) ≠ FIND-SET(v)
            then A ← A ∪ {(u, v)}
                  UNION(u, v)
return A
```

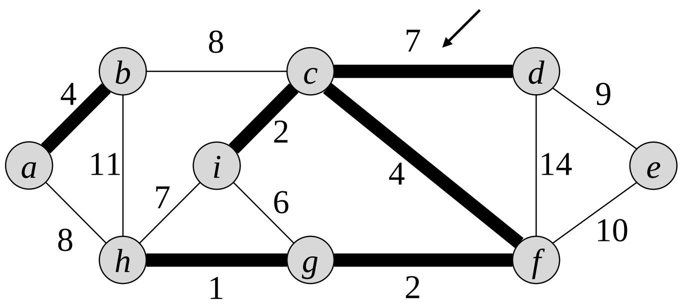
(e)



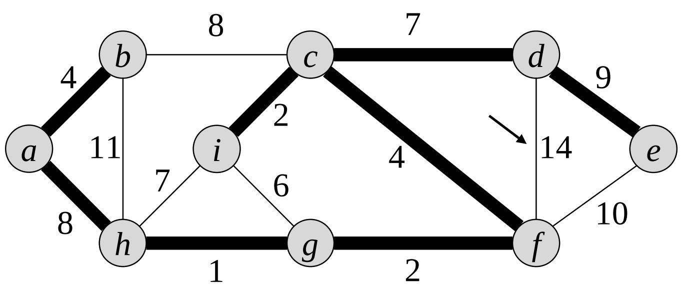
(f)



(g)



(n)



Time complexity:

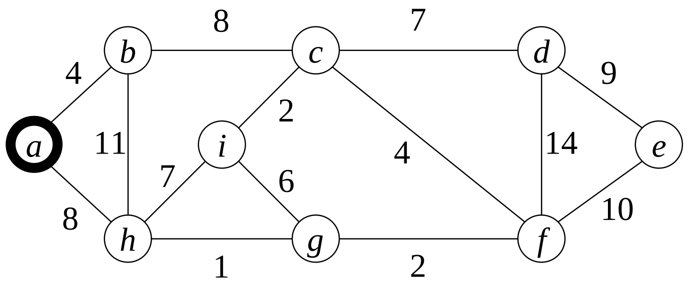
Steps 1~3:	$O(V)$
Step 4:	$O(E\lg E)$ (sorting)
Steps 5~8:	$O(E\alpha(V))=O(E\lg E)$ (disjoint-set-forest in 21.3)

* α is the inverse Ackermann's function
* $\alpha(n) \leq 4$ for for all practical cases
* $T(n) = O(E\lg E)$
* If all weights are bounded integers, $T(n) = O(E\alpha(V))$

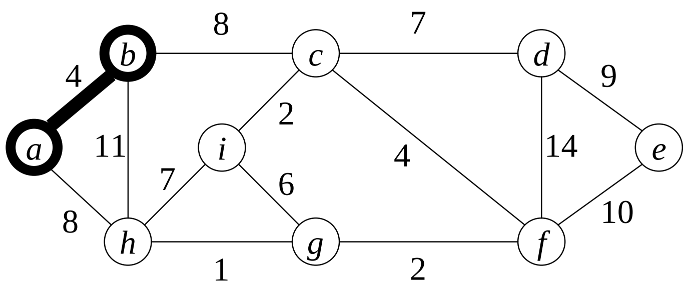
Prim's algorithm: vertices in A always form a single tree.

```
MST-PRIM(G, w, r)
for each u ∈ V[G]
    do key[u] ← ∞
       π[u] ← NIL
key[r] ← 0
Q ← V[G]
while Q ≠ Ø
    do u ← EXTRACT-MIN(Q)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
                 then π[v] ← u
                      key[v] ← w(u, v)
```

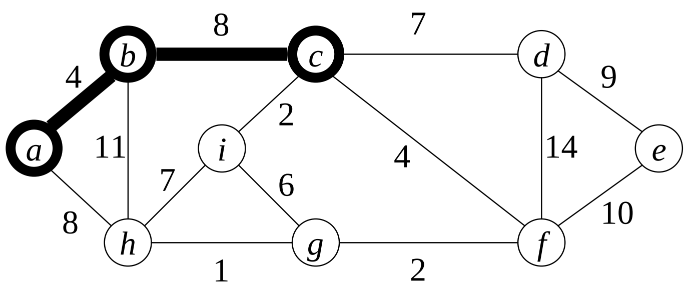
(a)


8 7
4 b c d 9
2
a 11 i 4 14 e
7 6
8 10
h g f
1 2

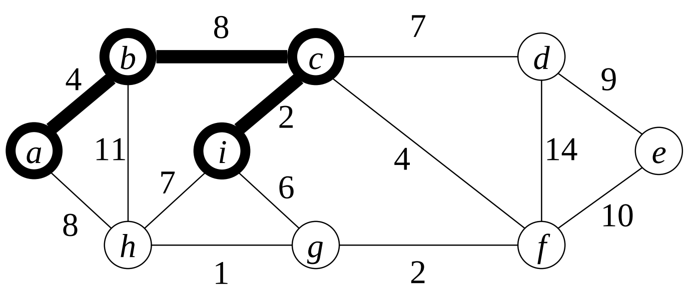
(b)


8 7
4 b c d 9
2
a 11 i 4 14 e
7 6
8 10
h g f
1 2

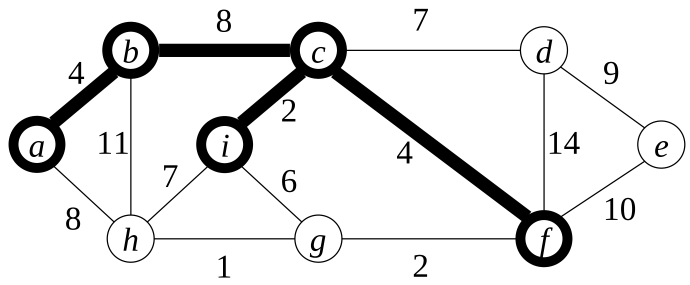
(c)


8 7
4 b c d 9
2
a 11 i 4 14 e
7 6
8 10
h g f
1 2

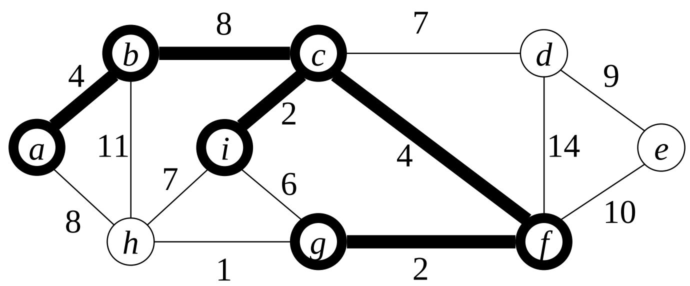
(d)


8 7
4 b c d 9
2
a 11 i 4 14 e
7 6
8 10
h g f
1 2

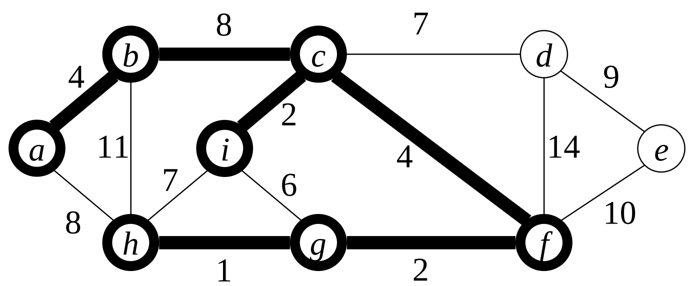
(e)


8 7
4 b c d 9
2
a 11 i 4 14 e
7 6
8 10
h g f
1 2

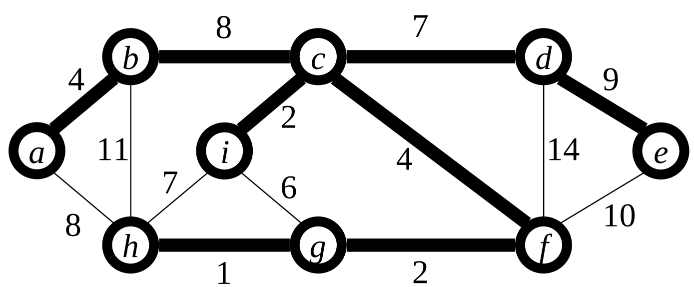
(f)


8 7
4 b c d 9
2
a 11 i 4 14 e
7 6
8 10
h g f
1 2

(g)


8 7
4 b c d 9
2
a 11 i 4 14 e
7 6
8 10
h g f
1 2

(i)


8 7
4 b c d 9
2
a 11 i 4 14 e
7 6
8 10
h g f
1 2

Time complexity:

(a) Implement priority queue Q as an array

Steps 1~5: $O(V)$ (Build Q)
Step 7: $O(V^2)$ (V times Extract-Min)
Steps 8~11: $O(E)$ ($2E$ times Decrease-key)
Total: $O(V^2 + E) = O(V^2)$ (for dense G)

(b) Implement priority queue Q as a binary heap

Steps 1~5: $O(V)$ (Build Q)
Step 7: $O(V \lg V)$ (V times *Extract-Min*)
Steps 8~11: $O(E \lg V)$ ($2E$ times *Decrease-Key*)
Total: $O(E \lg V)$ (for sparse G)

(c) Implement Q as a Fibonacci heap

Steps 1~5: $O(V)$ (Build Q)
Step 7: $O(V \lg V)$ (V times *Extract-Min*)
Steps 8~11: $O(E)$ ($2E$ times *Decrease-Key*)
Total: $O(E + V \lg V)$ (for sparse G)

Homework: Ex. 23.2-2, 23.2-4, 23.2-5, Prob. 23-1, 23-3.