Kruskal’s Algorithm for MST
KRUSKAL’S ALGORITHM for MST

Start w/ vertex set
Kruskal's Algorithm for MST

SORT edges

Diagram of graph with edges labeled: 10, 11, 13, 15, 31, 33, 38, 42, 47, 50.
KRUSKAL’S ALGORITHM for MST

SCAN sorted list
Kruskal’s Algorithm for MST

are endpoints in different components?

SCAN sorted list
Kruskal’s Algorithm for MST

Yes, so add edge to MST
Kruskal's Algorithm for MST
Kruskal's Algorithm for MST

Suppose this edge is not in MST
Kruskal's Algorithm for MST

Suppose this edge is not in MST

Eventually we must link the endpoints with some path
Kruskal's Algorithm for MST

Suppose this edge is not in MST

Hypothesis: endpoints in different components

Eventually we must link the endpoints with some path

Some edge will be scanned later. 

\[ \text{higher weight} \]
Kruskal's Algorithm for MST

Suppose this edge is not in MST

Hypothesis: endpoints in different components

Eventually we must link the endpoints with some path

Some edge will be scanned later. 4 higher weight
Kruskal's Algorithm for MST

Resume example
just added
edge & merged
2 components
Kruskal’s Algorithm for MST

continue scanning

endpoints in different components
Kruskal's Algorithm for MST

Add to MST

10
11
13
15
31
38
42
47
50
33
Kruskal's Algorithm for MST

Add to MST & merge

10 11 13 15 47
50 33 38

10 11 13 15 31 33 38 42 47 50
Kruskal’s Algorithm for MST
Kruskal's Algorithm for MST

Add to MST & merge

Maintaining several components (a forest of trees)
Kruskal’s Algorithm for MST

must add & merge
Kruskal's Algorithm for MST
Kruskal's Algorithm for MST

Finally, example of endpoints in same component. Ignore edge...
Kruskal’s Algorithm for MST
Kruskal's Algorithm for MST
Kruskal’s Algorithm for MST
Kruskal’s Algorithm for MST

Diagram of a graph with edges and weights, showing the process of constructing a minimum spanning tree (MST).
Kruskal's Algorithm for MST
Kruskal's Algorithm for MST

1) make forest of vertices
KRUSKAL'S ALGORITHM for MST

1) make forest of vertices
2) sort edges

time?
Kruskal’s Algorithm for MST

1) make forest of vertices $O(v)$
2) sort edges $O(E \log E)$
KRUSKAL’S ALGORITHM for MST

1) make forest of vertices \(O(v)\)
2) sort edges \(O(E \log E)\)
3) for each edge
   - CHECK ENDPOINTS (same component?)
   - No? Add edge.
   - MERGE.
**Kruskal's Algorithm for MST**

1) Make forest of vertices  \( O(V) \)
2) Sort edges  \( O(E \log E) \)
3) For each edge  \( O(E) \)

**Check endpoints**

(same component?)

\( \triangleright \) No? Add edge.  \( O(1) \)
\( \triangleright \) Merge.
Given an edge (~ two vertices) & a forest

CHECK ENDPOINTS
(same component?)

→ No? Add edge.
MERGE (entire components)

brute force?
Given an edge (~ two vertices) & a forest

**CHECK ENDPOINTS**
(same component?)

- No? Add edge. **MERGE**

→ use a variable for each vertex \( O(1) \)

→ change variable for all vertices in component \( O(v) \)
Given an edge (~ two vertices) & a forest

1. **CHECK ENDPOINTS**
   (same component?)
   - No? Add edge.
   - **MERGE**

... for an application that will do this many times

- Kruskal's algo

- \( O(E) \) : \( O(V^2) \)
**basic UNION-FIND**

Given an edge (~ two vertices) & a forest

CHECK ENDPOINTS
(same component?)

\( \Downarrow \) No? Add edge.

MERGE

... for an application that will do this many times

\[ O(E) : O(V^2) \]

Kruskal's algo
Store each component as a linked list
Store each component as a linked list

Every node points to one representative.
Store each component as a linked list

Every node points to one representative.

"Find/add/delete: O(1)

we don't need these

NOT search.
Assume pointer to node
Store each component as a linked list

Every node points to one representative.

Find/add/delete: O(1)
Query "same component": check representatives: O(1)
Store each component as a linked list

Every node points to one representative.

Find/add/delete: $O(1)$

Query "same component": check representatives: $O(1)$

Merge: looks like $O(n)$

WHY?
Store each component as a linked list

Every node points to one representative.

Find/add/delete : $O(1)$

Query "same component" : check representatives : $O(1)$

Merge : looks like $O(n)$

WHY?
Store each component as a linked list

Every node points to one representative.

Find/add/delete: $O(1)$
Query "same component": check representatives: $O(1)$
Merge: looks like $O(n)$

Why?
Store each component as a linked list

Every node points to one representative.

Find/add/delete: $O(1)$
Query "same component": check representatives: $O(1)$
Merge: looks like $O(n)$
$\sum n_i = n$
$\sum n_i = n$

$\text{merge}(j,k) = O(\min(n_j, n_k))$
$\sum n_i = n$ 

How many times can an item merge? 

(= how many times can an item change representatives?)
How many times can an item merge? (= how many times can an item change representatives?)

\[ \sum n_i = n \]

Component size doubles each time
\[ \sum n_i = n \]

How many times can an item merge?

(= how many times can an item change representatives?)

\[ \text{merge}(j,k) = O(\min(n_j, n_k)) \]

Component size doubles each time

\[ O(\log n) \]
\( \sum n_i = n \)  

How many times can an item merge?  
(= how many times can an item change representatives?)

\[ \text{merge}(j,k) = O(\min(n_j, n_k)) \]

\( O(\log n) \)  

Component size doubles each time  

Total cost of \( n \) merges: \( O(n \log n) \)
basic UNION-FIND

Given an edge (~ two vertices) & a forest

CHECK ENDPOINTS (same component?)
  \[ \rightarrow O(1) \]

\( \downarrow \) No? Add edge,

MERGE

\[ \rightarrow O(1) \]

\[ \left\{ \begin{array}{l}
O(V) \text{ worst case } \\
O(V \log V) \text{ total }
\end{array} \right. \]

... for an application that will do this many times

\[ \downarrow \]

Kruskal's algo

\[ \rightarrow O(V) \text{ checks } O(E) \text{ merges} \]
KRUSKAL'S ALGORITHM for MST

1) make forest of vertices \(O(v)\)
2) sort edges \(O(E \log E)\)
3) for each edge
   CHECK ENDPOINTS (same component?)
   \(\rightarrow\) No? Add edge.
   MERGE

\(\rightarrow\) \(O(V \log V)\) total
Kruskal’s Algorithm for MST

1) make forest of vertices $O(v)$
2) sort edges $O(E\log E) = O(E\log V)$
3) for each edge
   - check endpoints (same component?) $O(E)$ total
     - No? Add edge. $O(V\log V)$ total
     - Merge

- 10
- 11
- 13
- 15
- 31
- 33
- 38
- 42
- 47
- 50
Kruskal's Algorithm for MST

1) Make forest of vertices \( O(v) \)
2) Sort edges \( O(E \log E) = O(E \log V) \)
3) For each edge
   - Check endpoints (same component?) \( O(E) \) total
   - No? Add edge, merge \( O(V \log V) \) total
KRUSKAL'S ALGORITHM for MST

1) make forest of vertices \( O(V) \)
2) sort edges \( O(E \log E) = O(E \log V) \) \( \log E = O(\log^2 V) \)
3) for each edge
   - CHECK ENDPOINTS (same component?) \( O(E) \) total
     - No? Add edge. MERGE
     - Yes? \( O(V \log V) \) total \( = O(E \log V) \) why?
Kruskal's Algorithm for MST

1) Make forest of vertices $O(V)$
2) Sort edges $O(E \log E) = O(E \log V)$ \( \log E = O(\log^2 V) \)
3) For each edge
   - Check endpoints (same component?)
     - No? Add edge, Merge

$O(E)$ total

$O(V \log V)$ total $= O(E \log V)$

E $\geq V+1$ assuming connected G
Kruskal's Algorithm for MST $\rightarrow O(E \log V)$

1) make forest of vertices $O(V)$
2) sort edges $O(E \log E) = O(E \log V)$  $\log E = O(\log V^2)$
3) for each edge
   - check endpoints (same component?)
     - if no? Add edge, merge
       $O(E)$ total
     - $O(V \log V)$ total = $O(E \log V)$

$E \geq V+1$ assuming connected $G$