Kruskal's Algorithm for MST
Kruskal's Algorithm for MST

Start w/ vertex set
Kruskal’s Algorithm for MST

SORT edges
KRUSKAL'S ALGORITHM for MST
Kruskal's Algorithm for MST

SCAN sorted list

Are endpoints in different components?
Kruskal's Algorithm for MST

Yes.
so add
dge to
MST
Kruskal’s Algorithm for MST

Yes. So add edge to MST.

Why?
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. 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.

Higher weight
Kruskal’s Algorithm for MST

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

Endpoints in different components

Continue scanning
KRUSKAL'S ALGORITHM for MST

add to MST

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

add to MST & merge
KRUSKAL'S ALGORITHM for MST
KRUSKAL'S ALGORITHM for MST

add to MST & merge

Maintaining several components → forest of trees
Kruskal's Algorithm for MST
Kruskal's Algorithm for MST

Finally, example of endpoints in same component.

\[\text{IGNORE edge}\]
KRUSKAL’S ALGORITHM for MST
KRUSKAL'S ALGORITHM for MST

...DONE
why?
Kruskal's Algorithm for MST
KRUSKAL'S ALGORITHM for MST
Kruskal's Algorithm for MST
Kruskal's Algorithm for MST
Kruskal’s Algorithm for MST

1) Make forest of vertices

---

Diagram showing a graph with weighted edges and a minimum spanning tree.
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?)

   1. No? Add edge.
   2. Merge.
KRUSKAL'S ALGORITHM for MST

1) make forest of vertices \( O(V) \)
2) sort edges \( O(E \log E) \)
3) for each edge \( \cdots \) \( O(E^2) \)

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

Diagram of a graph with various edges and nodes, illustrating the algorithm's steps.
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?)
  - **MERGE**
  - **NO? Add edge.**

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

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

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 ENDPONITS
(same component?)
▷ No? Add edge.
▷ MERGE

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

Kruskal’s algo

$O(E) : O(V^2)$
Store each component as a linked list

$L_1$  

$L_2$  

$L_3$
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.

Query "same component": check representatives: $O(1)$
Store each component as a linked list

Every node points to one representative.

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

Merge: looks like \(O(n)\)

\(n = |V|\)
Store each component as a linked list

Every node points to one representative.

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.

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.

$L_1 \rightarrow R_1$

$L_2 \rightarrow (\text{better})$

$L_3 \rightarrow R_3$

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?)

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

How many times can an item merge?

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

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

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: \( O(n \log n) \)
Given an edge (~ two vertices) & a forest

\[\text{CHECK ENDPOINTS} \quad \text{(same component?)} \quad \rightarrow \quad O(1)\]

\[\text{No? Add edge, MERGE} \rightarrow O(1)\]

\[O(V) \text{ worst case} \quad O(V \log V) \text{ total} \quad \uparrow \quad \uparrow \quad \text{many times}\]

\[O(E) \text{ checks} \quad O(V) \text{ merges}\]

... for an application that will do this

\[\text{Kruskal's algo}\]
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?)
   \( \begin{cases} 
   \text{No? Add edge,} \\
   \text{MERGE} 
   \end{cases} \)  \( O(E) \) total
   \( O(V \log V) \) total
Kruskal's Algorithm for MST

1) make forest of vertices \( O(v) \)
2) sort edges \( O(E \log E) = O(E \log V) \) \( \text{why?} \)
3) for each edge
   
   \[
   \begin{align*}
   \text{CHECK ENDPOINTS} \\
   \text{(same component?)} \\
   \begin{cases}
   \text{Yes} & \text{Add edge} \\
   \text{No} & \text{Reject edge}
   \end{cases}
   \end{align*}
   \]

\( O(E) \) total

\( O(V \log V) \) 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
   \[
   \text{CHECK ENDPOINTS (same component?)} \begin{cases}
     \text{Yes} & \rightarrow O(E) \text{ total} \\
     \text{No? Add edge, MERGE} & \rightarrow O(V \log V) \text{ total}
   \end{cases}
   \]
\[\log E = O(\log V^2)\]
**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?)
     - No? Add edge, MERGE
       - $O(E)$ total
     - Yes? Merge components
       - $O(V \log V)$ total $= O(E \log V)$

$\log E = O(\log V^2)$
**Kruskal's Algorithm** for MST

1. Make forest of vertices \( O(V) \)
2. Sort edges \( O(E \log E) = O(E \log V) \) \( \log E = O(\log V^2) \)
3. For each edge
   - Check endpoints
     - Same component? \( \text{CHECK ENDPOINTS} \)
     - No? Add edge, Merge
   \( \rightarrow O(E) \) total

\( \rightarrow O(V \log V) \) 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?) $O(E)$
   - No? Add edge. MERGE $O(V \log V)$ total $= O(E \log V)$

$E \geq V-1$

assuming connected $G$