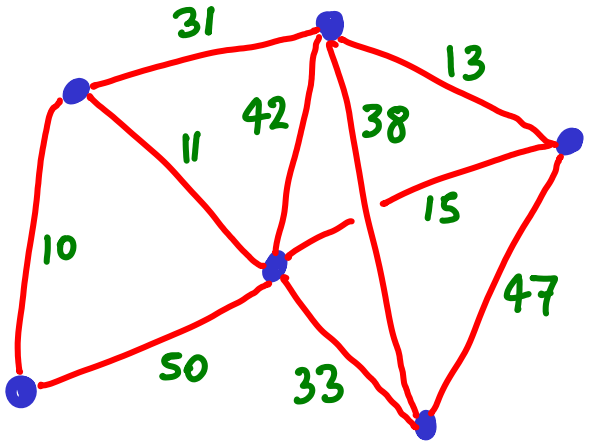
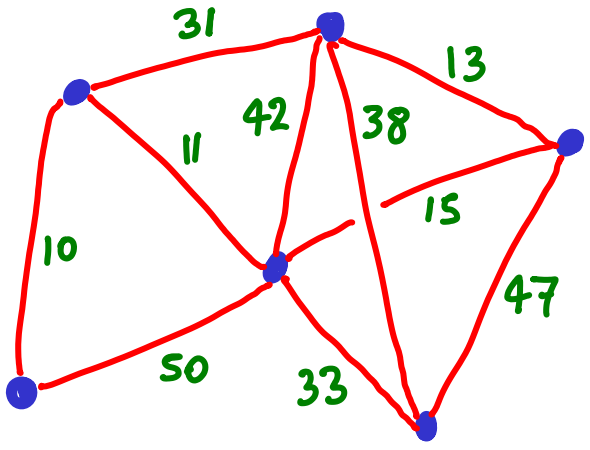
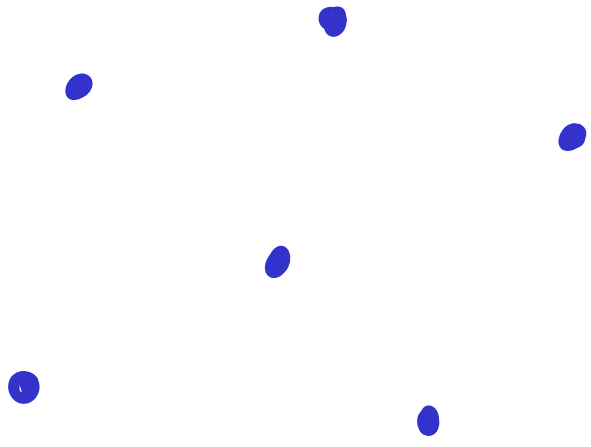
KRUSKAL'S ALGORITHM for MST



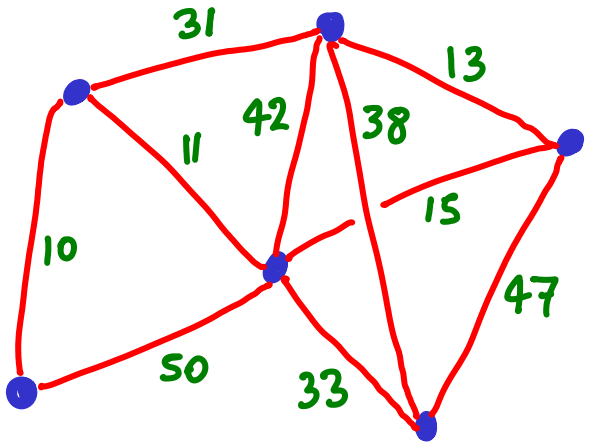
KRUSKAL'S ALGORITHM for MST

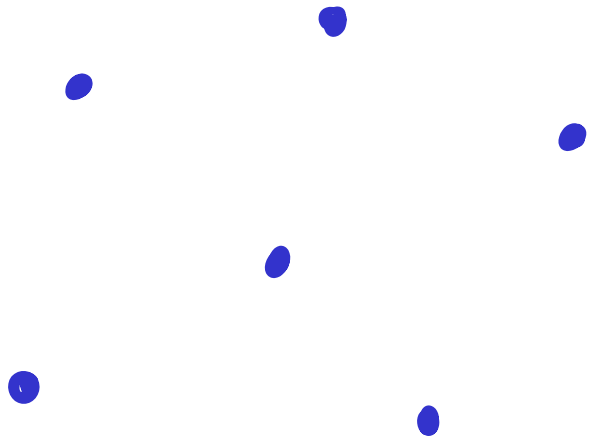




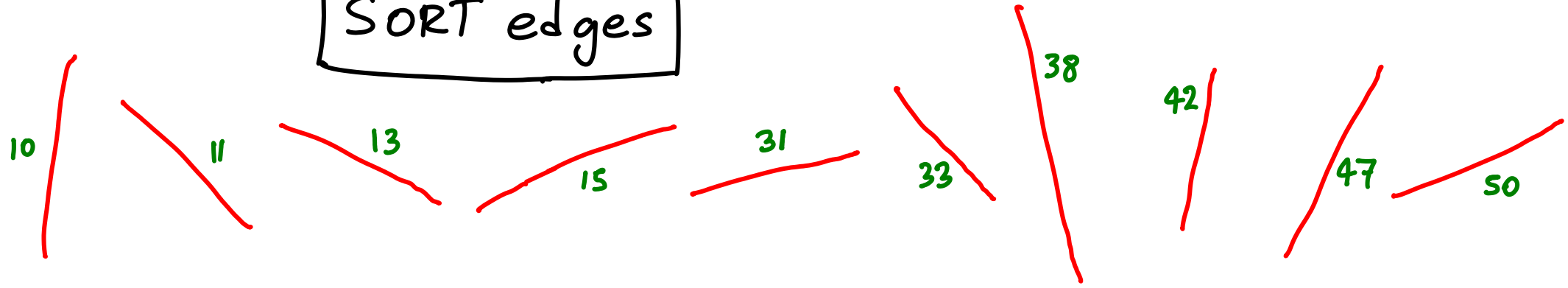
Start w/ vertex set

KRUSKAL'S ALGORITHM for MST

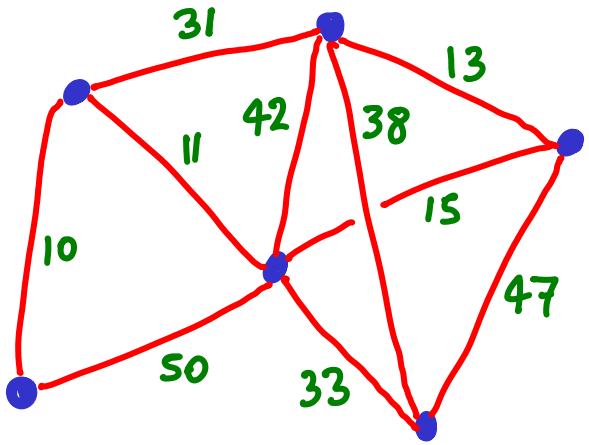




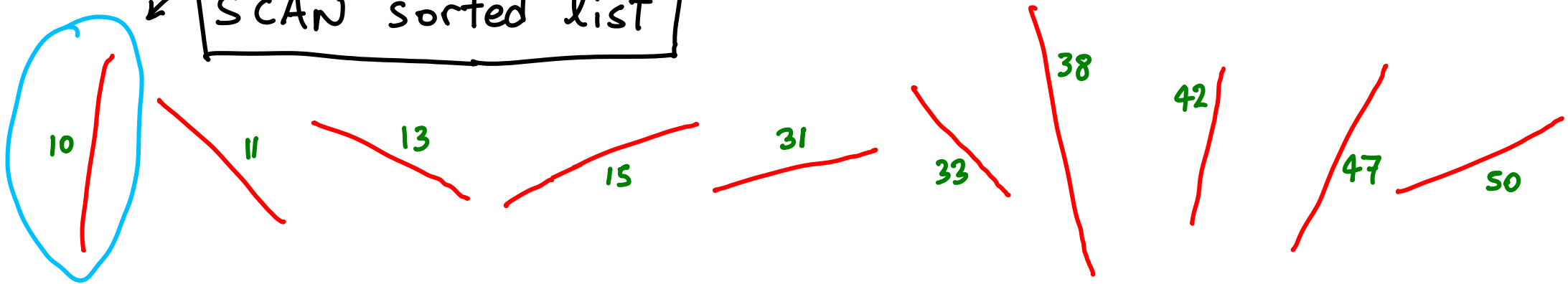
SORT edges



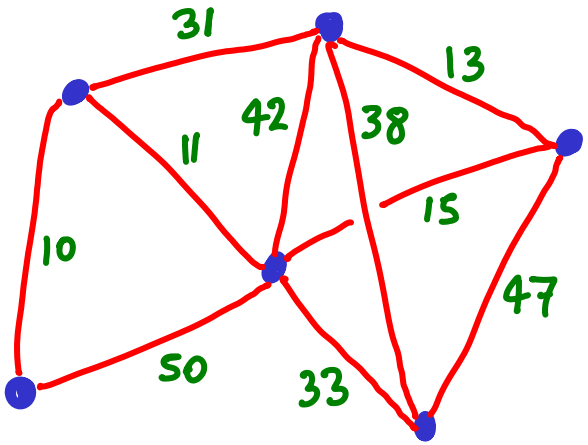
KRUSKAL'S ALGORITHM for MST



SCAN sorted list

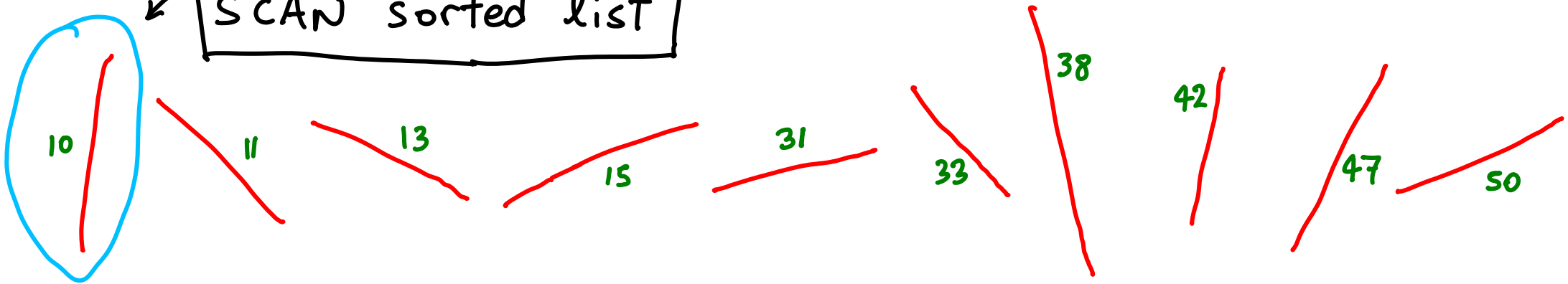


KRUSKAL'S ALGORITHM for MST

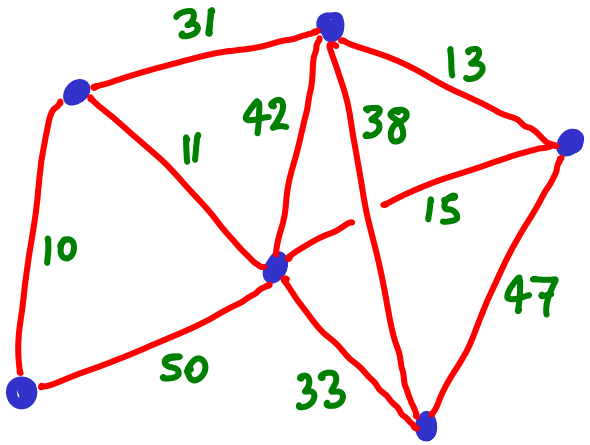


are endpoints in different components?

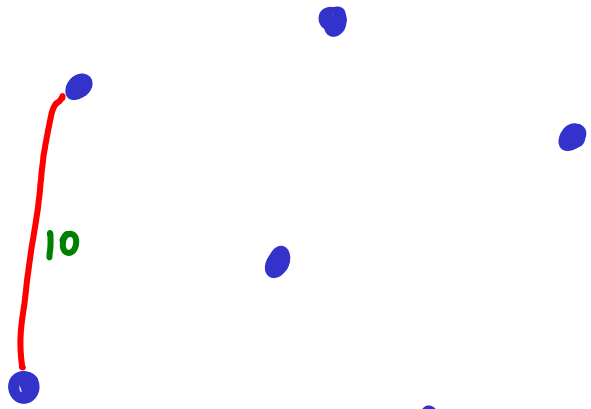
SCAN sorted list

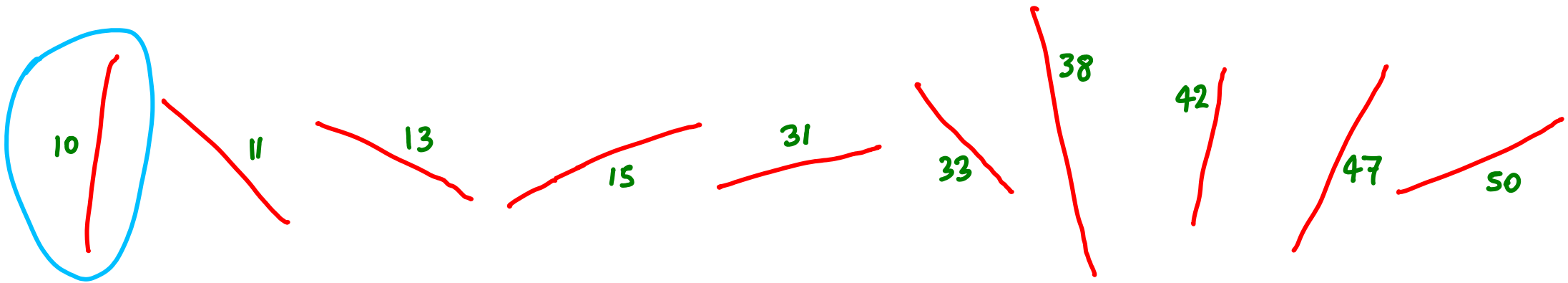


KRUSKAL'S ALGORITHM for MST

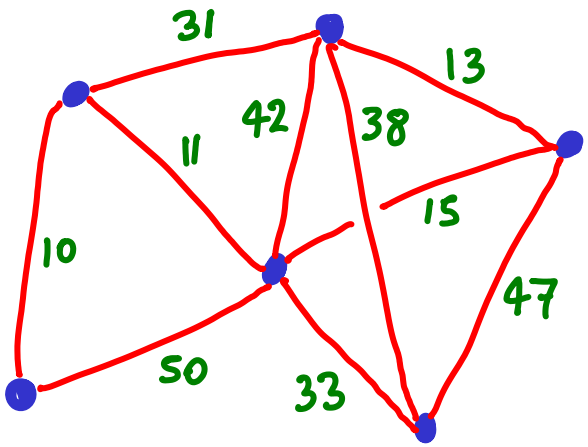


YES.
so add edge to MST

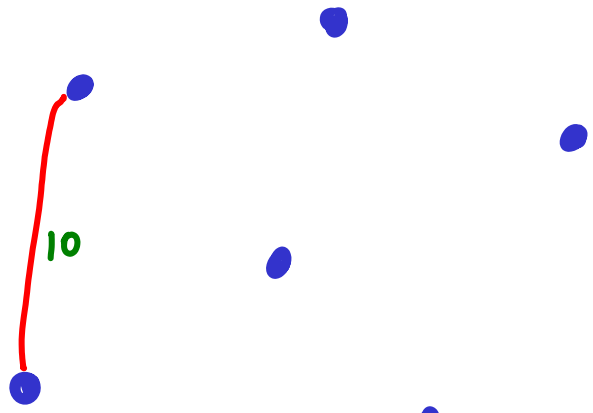




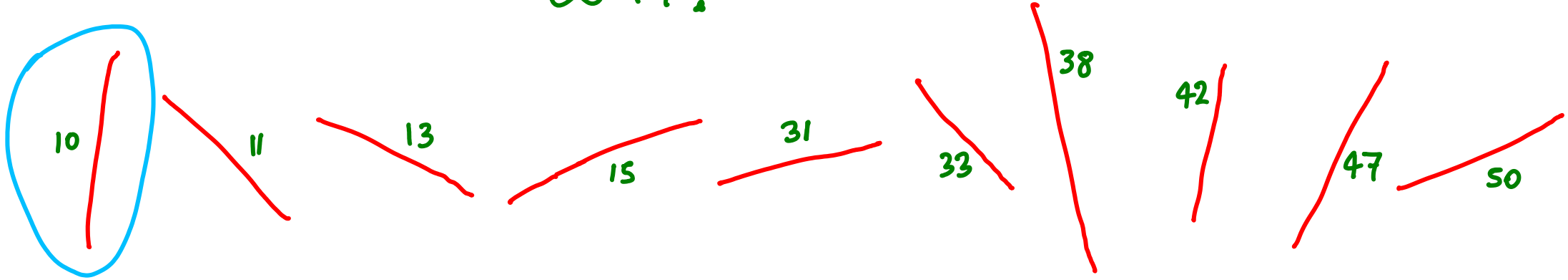
KRUSKAL'S ALGORITHM for MST



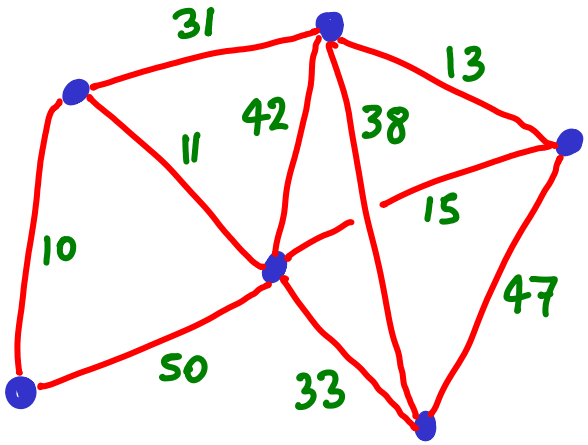
YES.
so add edge to MST



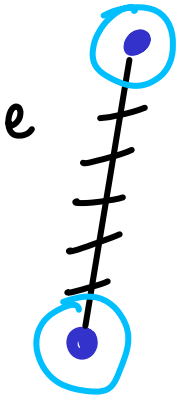
WHY?

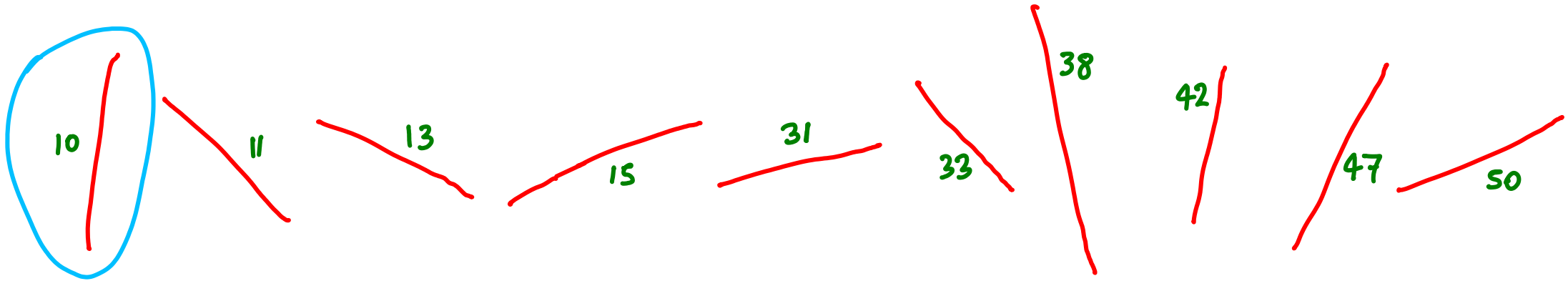


KRUSKAL'S ALGORITHM for MST

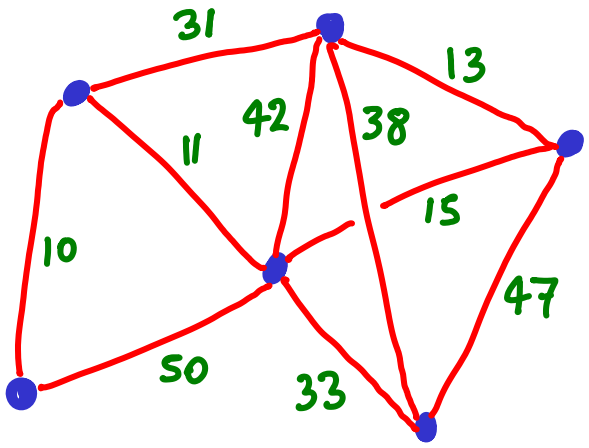


suppose this edge is not in MST

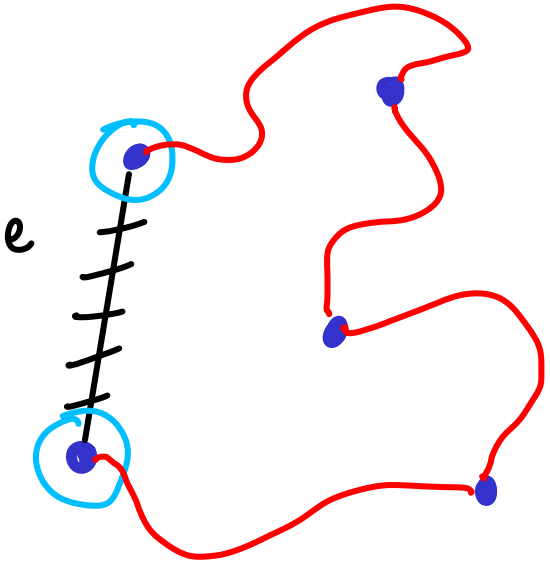




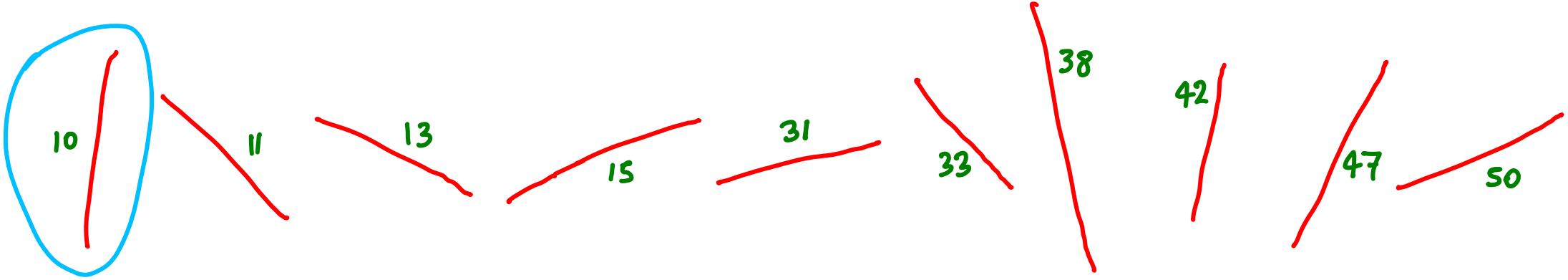
KRUSKAL'S ALGORITHM for MST



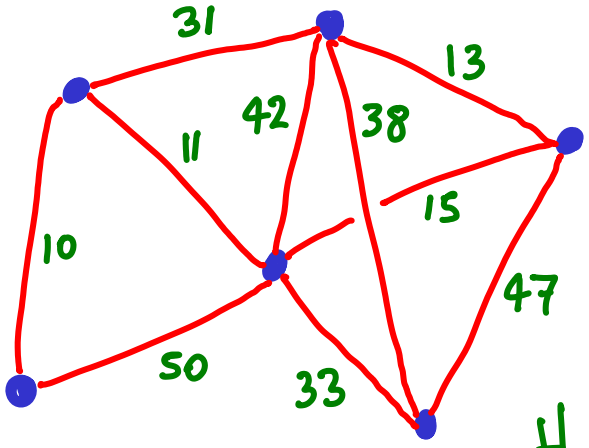
suppose this edge is not in MST



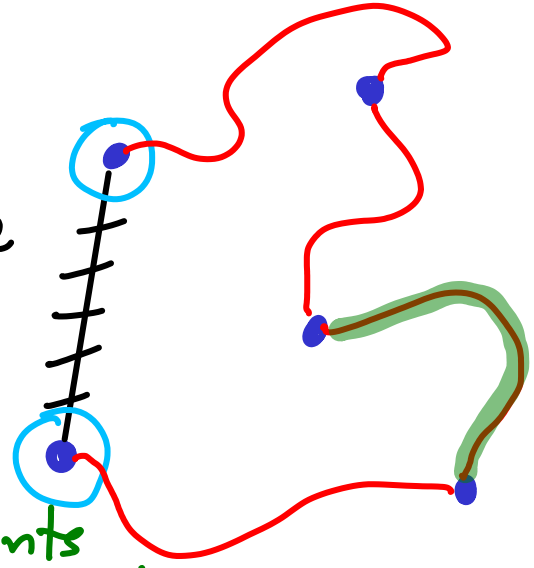
- eventually we must link the endpoints with some path



KRUSKAL'S ALGORITHM for MST



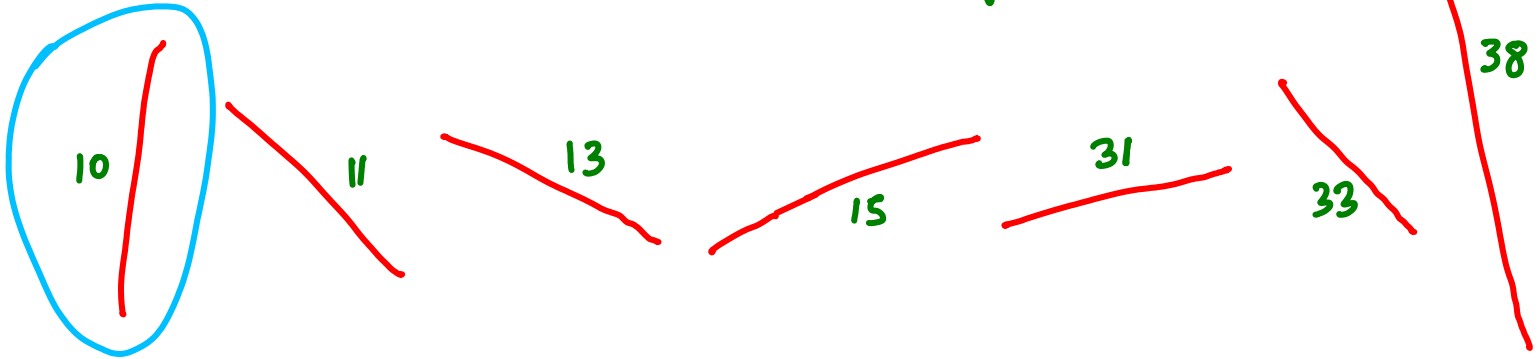
suppose this edge is not in MST

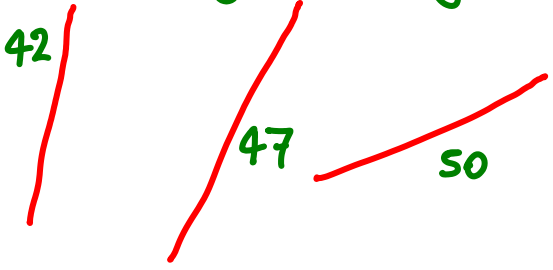


Hypothesis: endpoints in different components

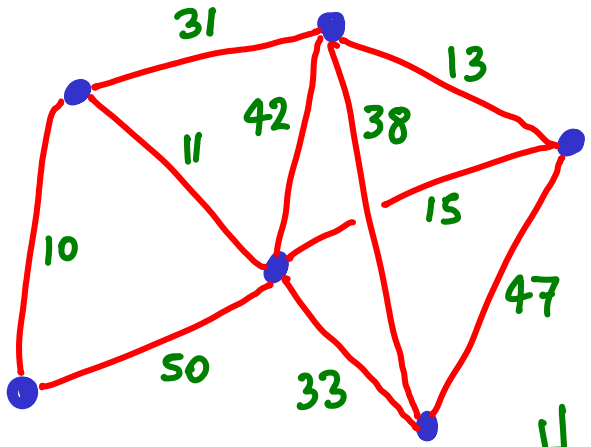
- eventually we must link the endpoints with some path

some edge will be scanned later.
↳ higher weight

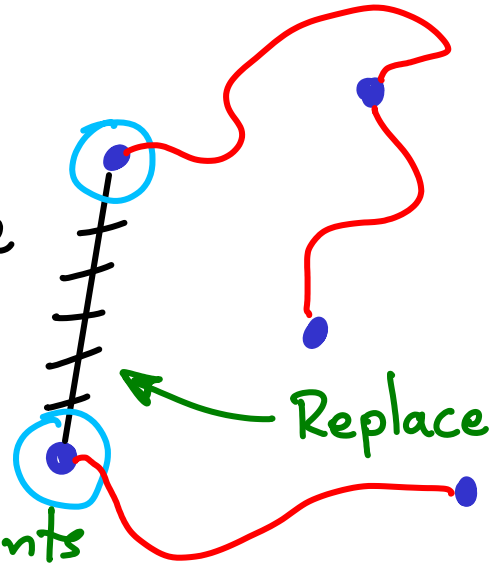




KRUSKAL'S ALGORITHM for MST



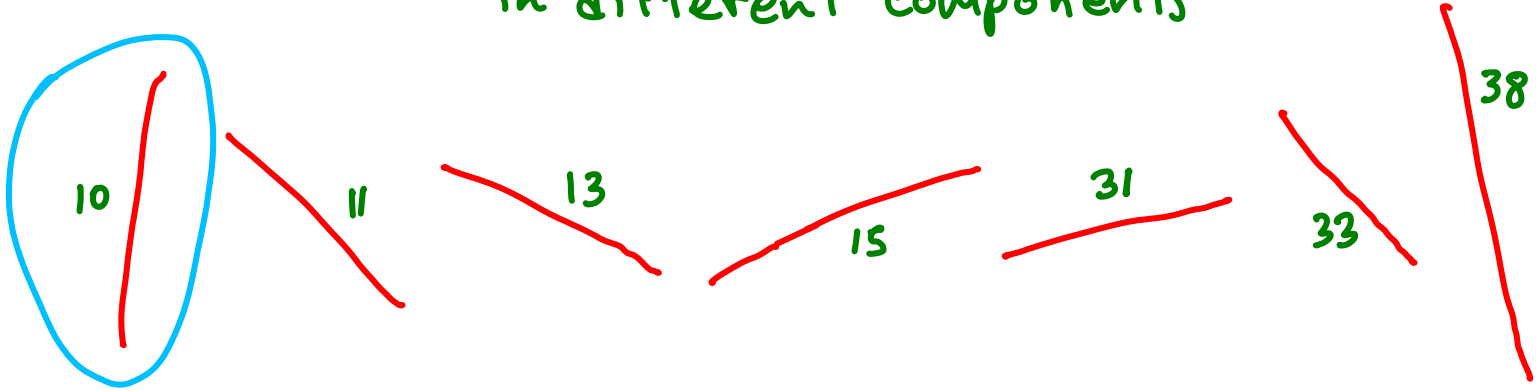
suppose this edge is not in MST



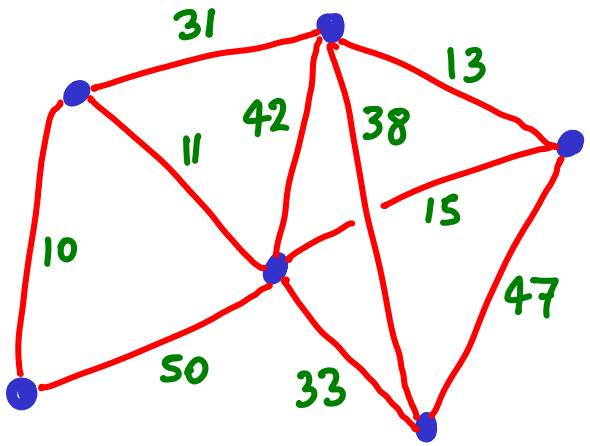
Hypothesis: endpoints in different components

- eventually we must link the endpoints with some path

some edge will be scanned later.
↳ higher weight

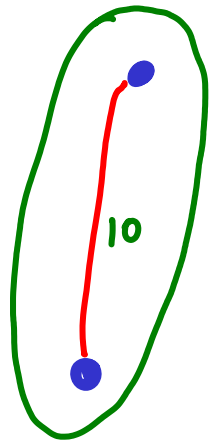


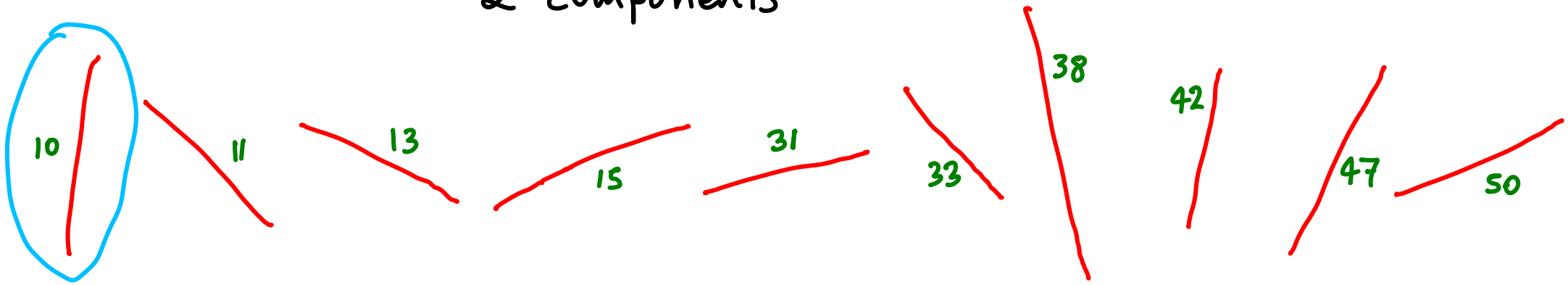
KRUSKAL'S ALGORITHM for MST



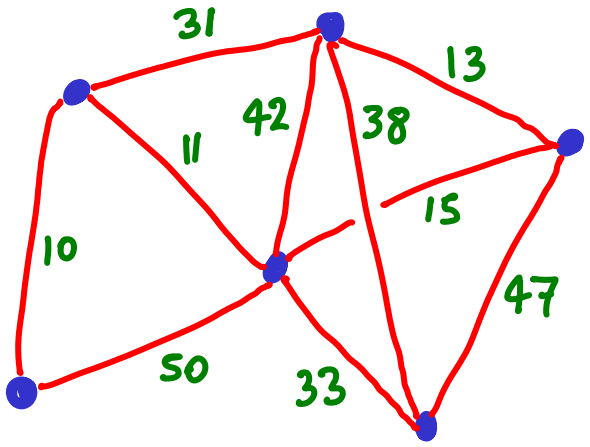
Resume example

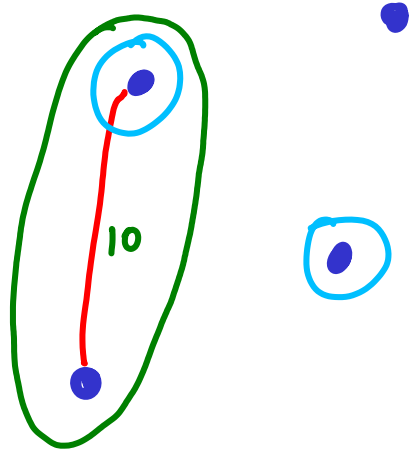
just added edge & merged 2 components





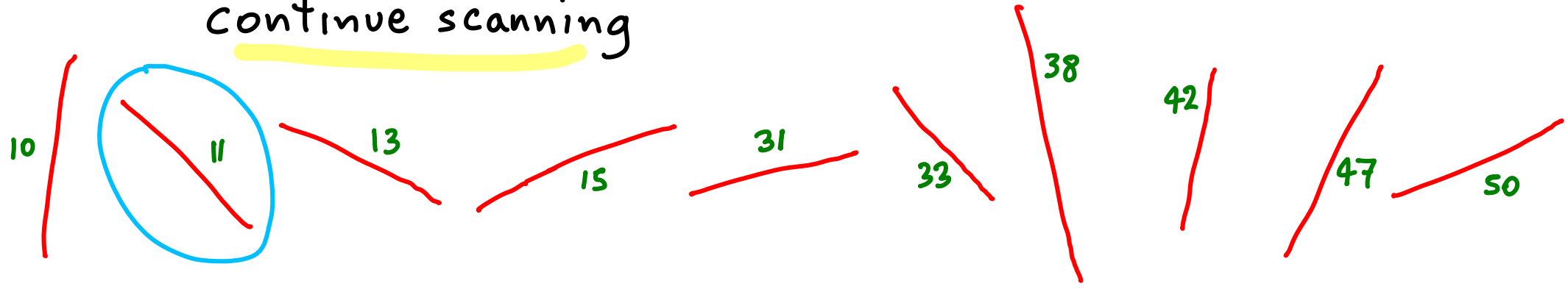
KRUSKAL'S ALGORITHM for MST



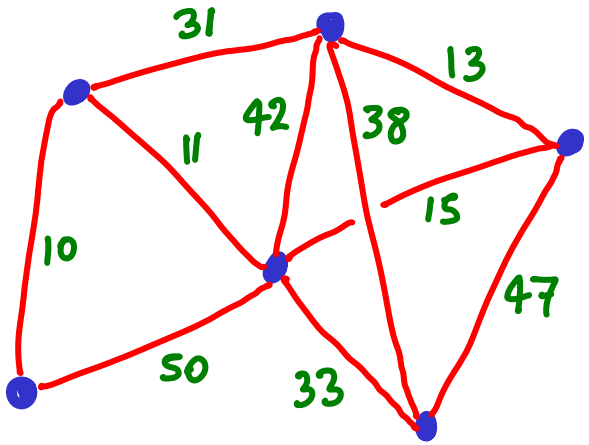


endpoints in different components

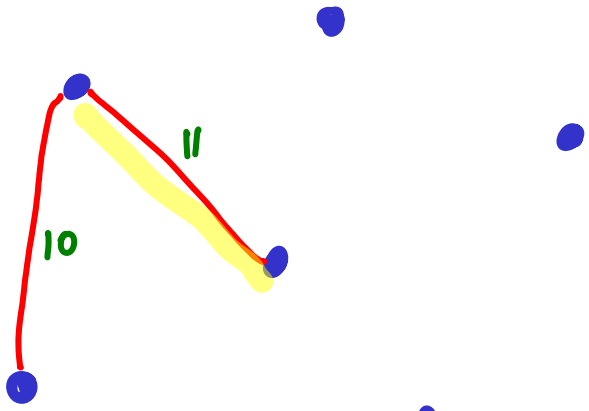
continue scanning

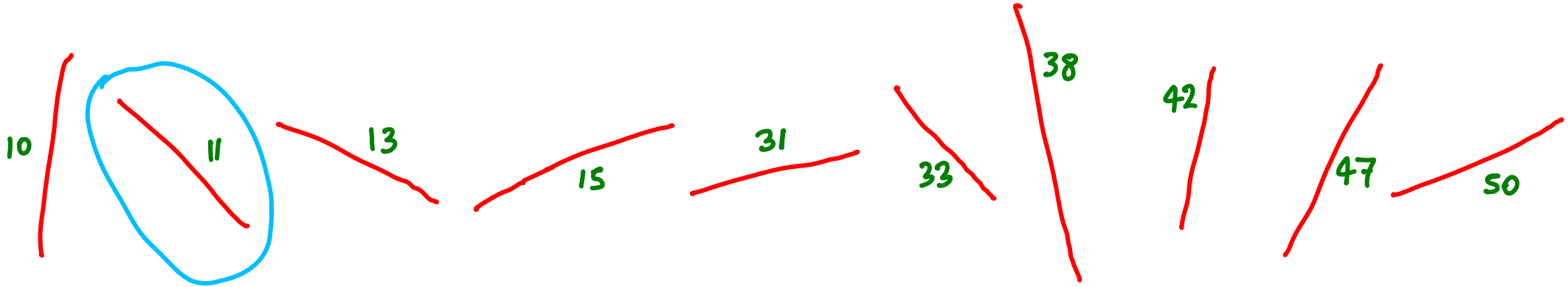


KRUSKAL'S ALGORITHM for MST

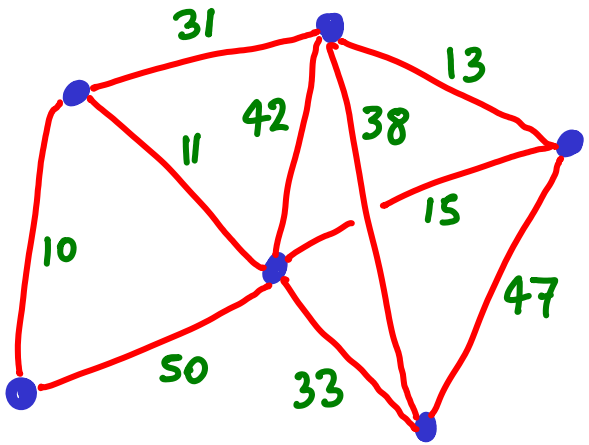


add to MST

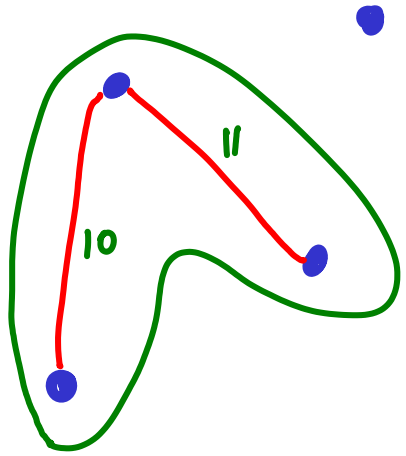


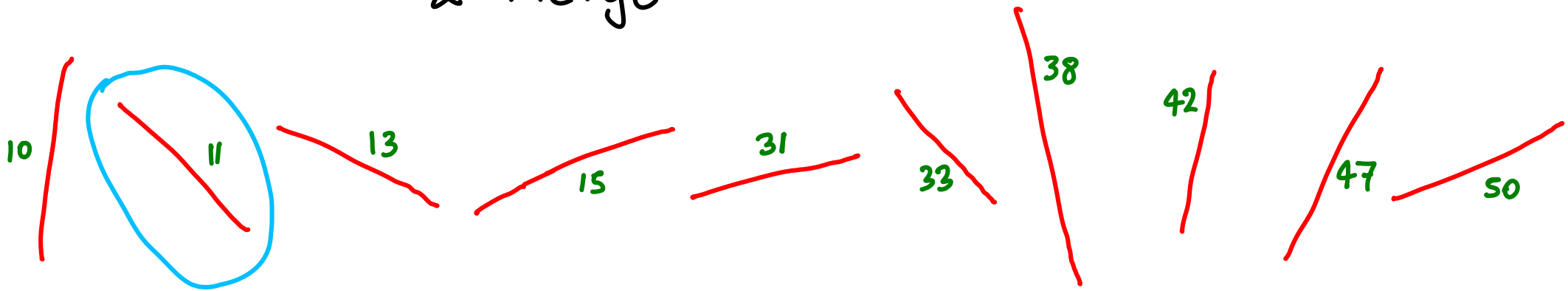


KRUSKAL'S ALGORITHM for MST

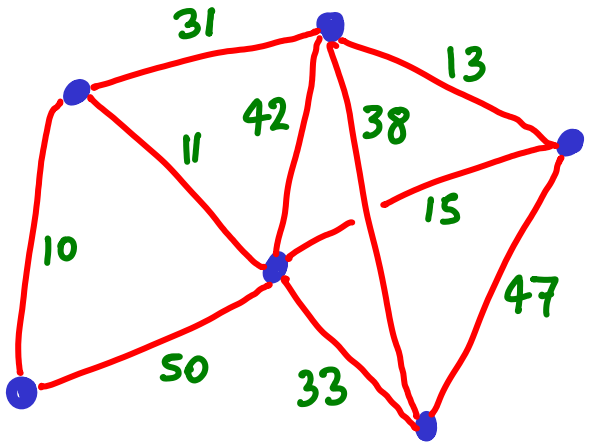


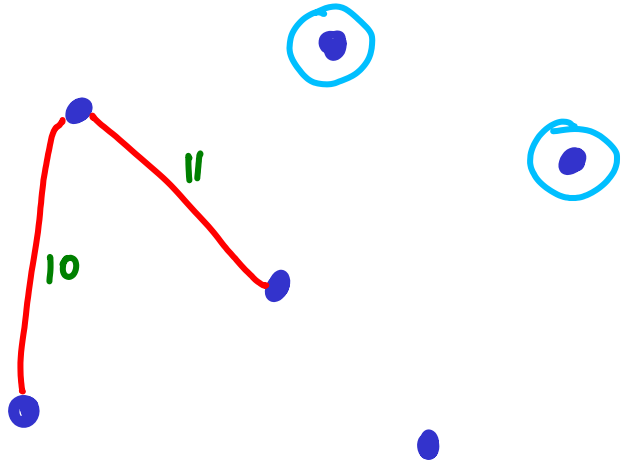
add to MST & merge

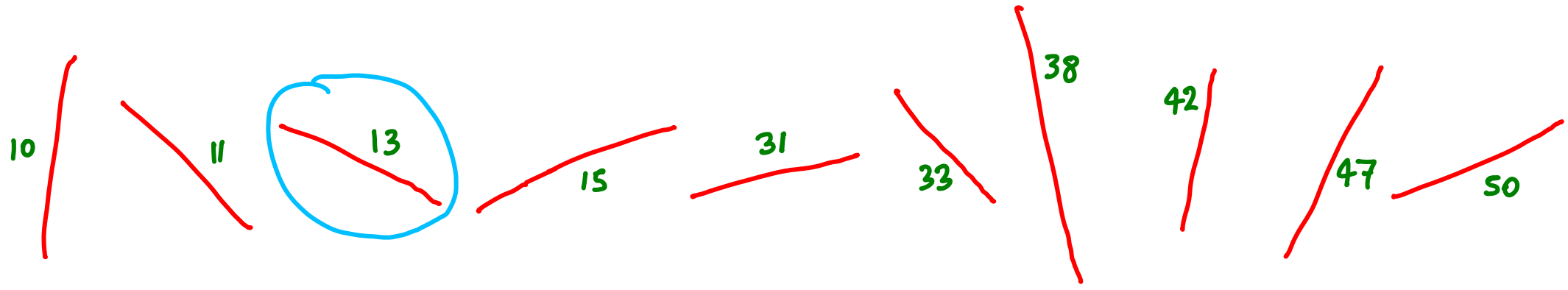




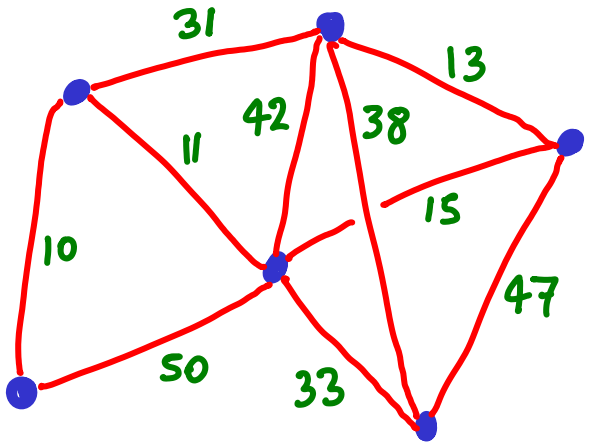
KRUSKAL'S ALGORITHM for MST

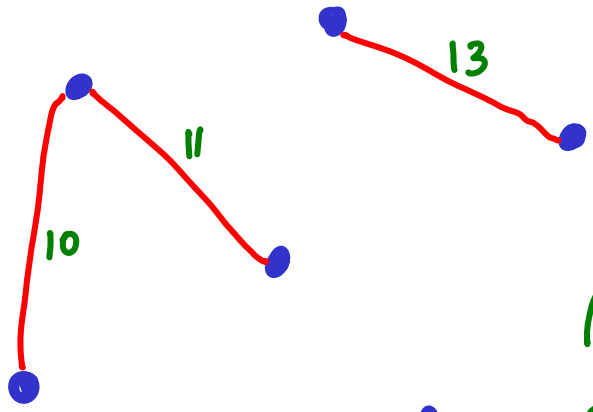






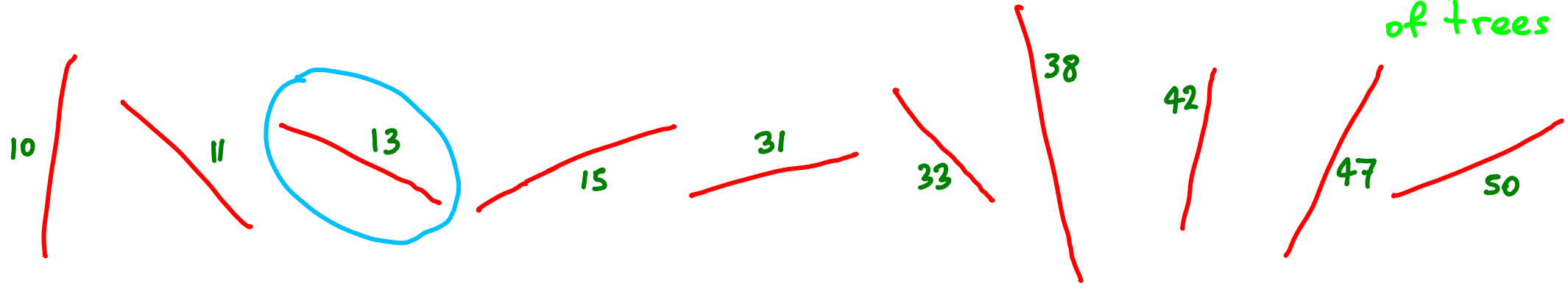
KRUSKAL'S ALGORITHM for MST



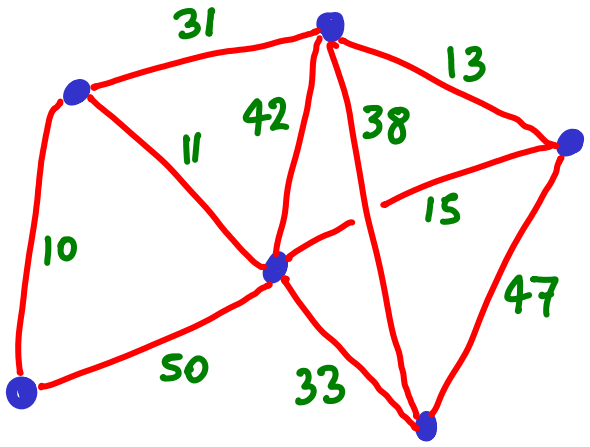


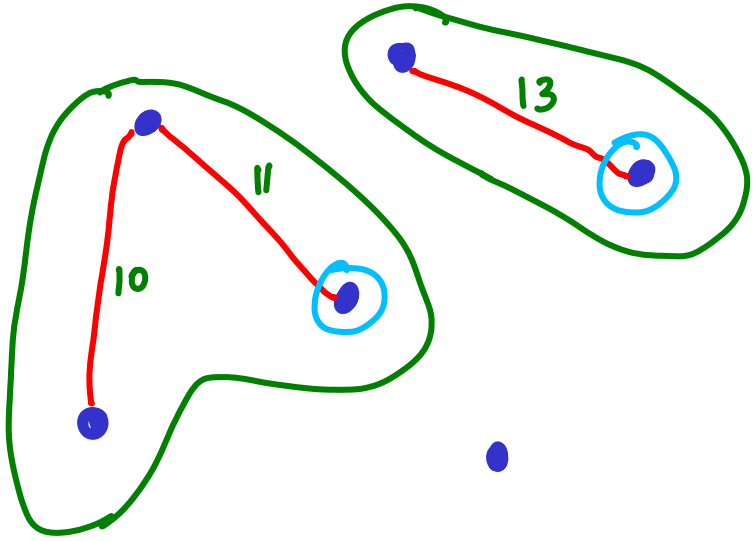
add to MST
& merge

Maintaining several components
↳ forest of trees

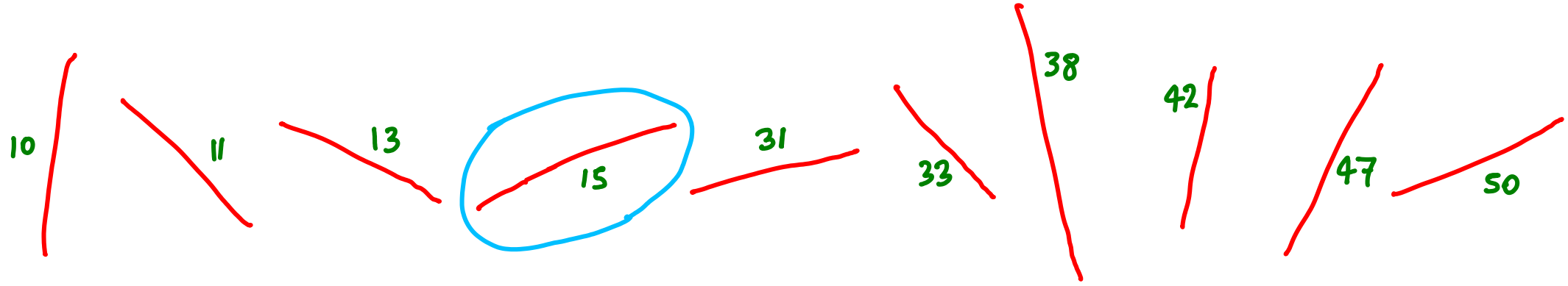


KRUSKAL'S ALGORITHM for MST

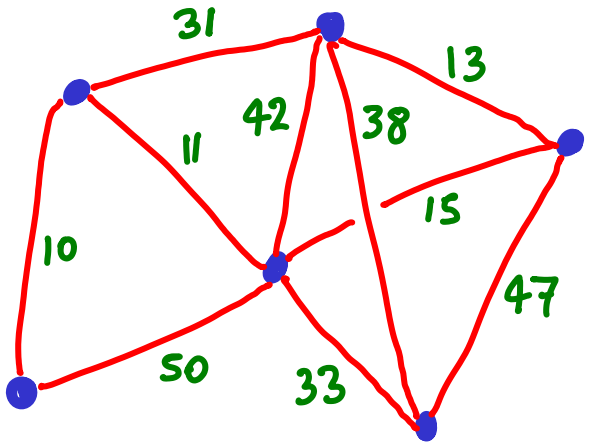


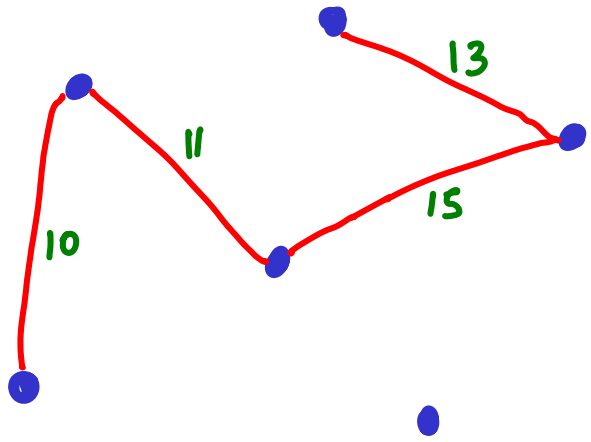


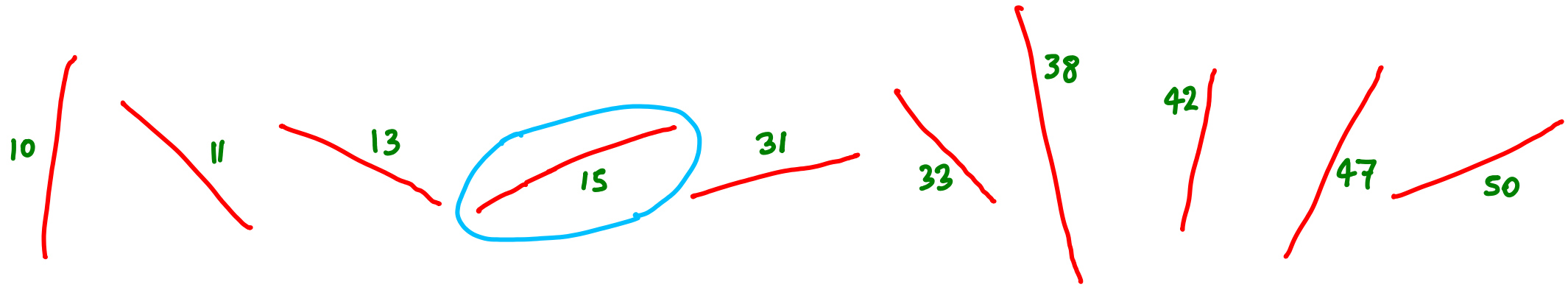
must add & merge



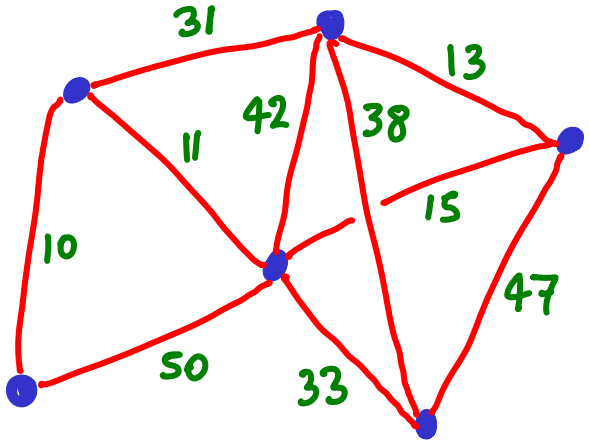
KRUSKAL'S ALGORITHM for MST

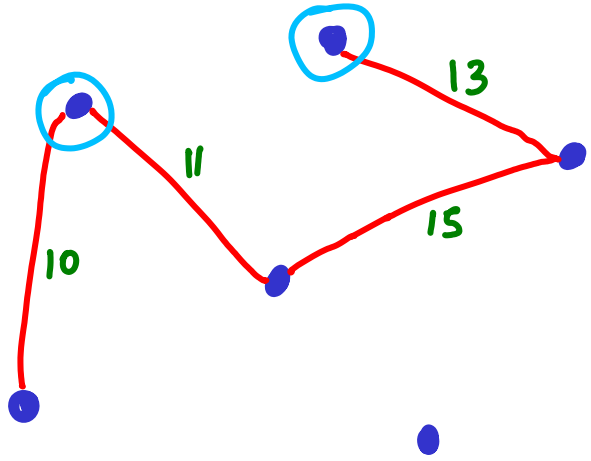






KRUSKAL'S ALGORITHM for MST

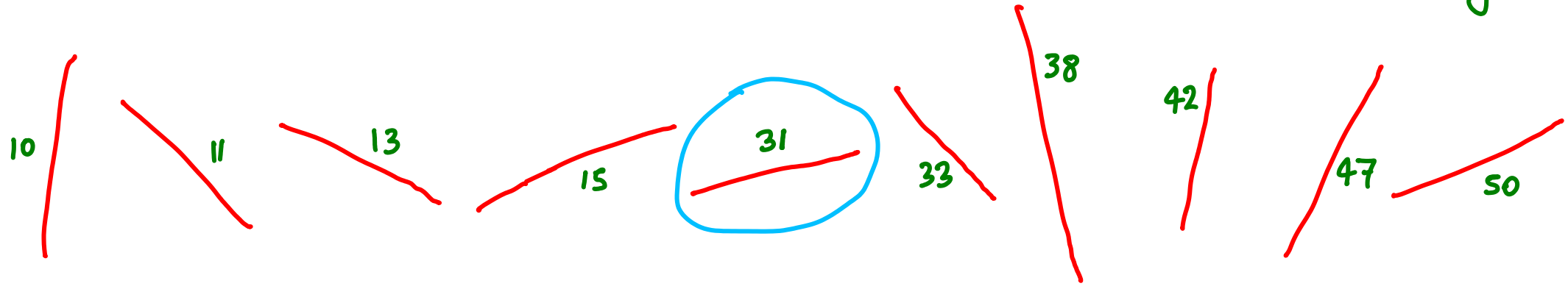




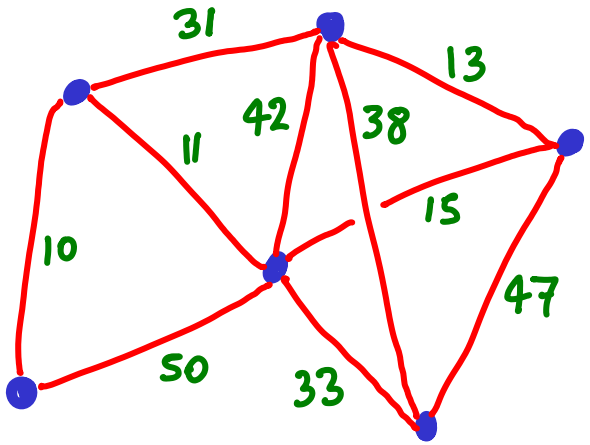
Finally, example of endpoints in same component.

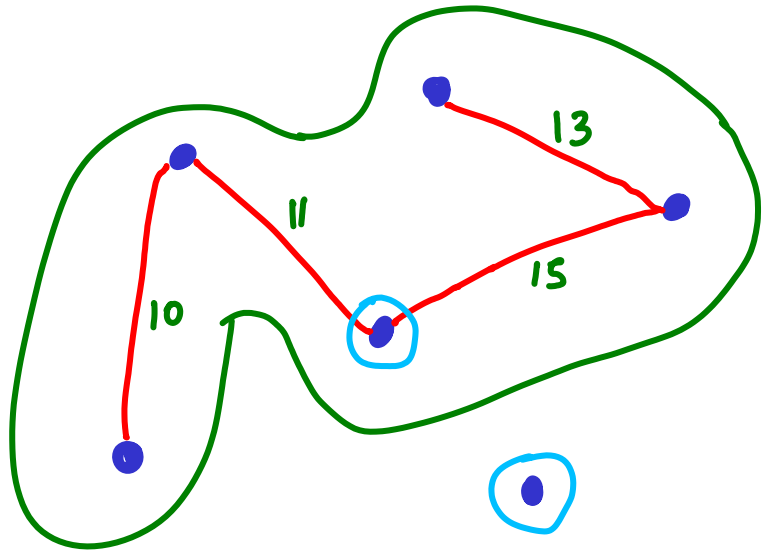
↓

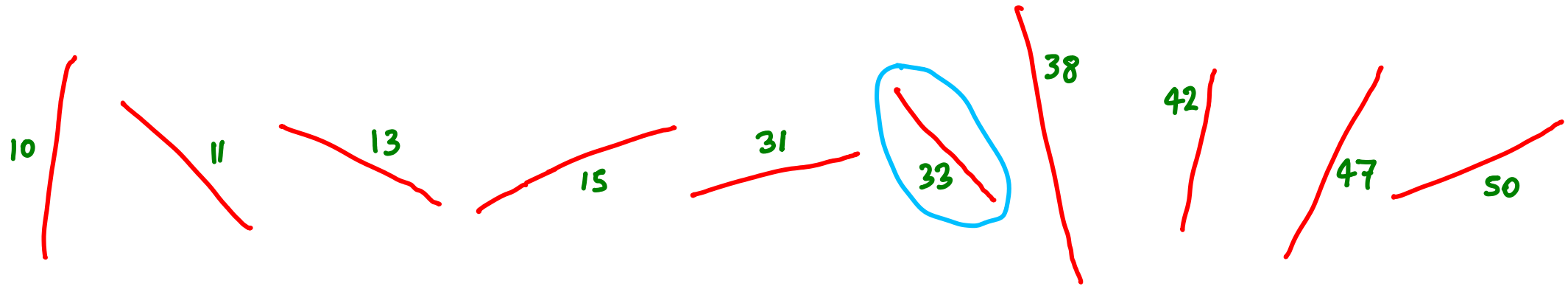
IGNORE edge



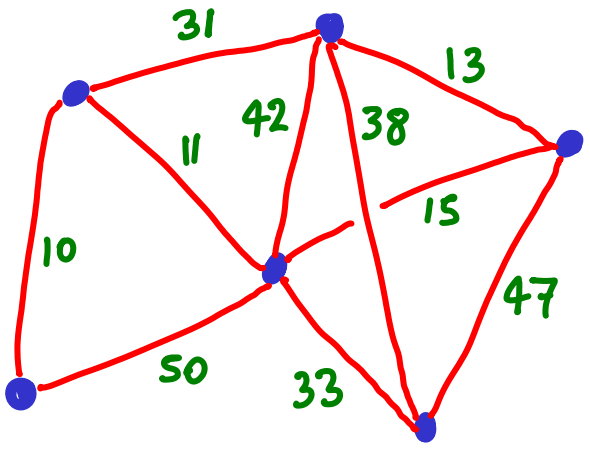
KRUSKAL'S ALGORITHM for MST

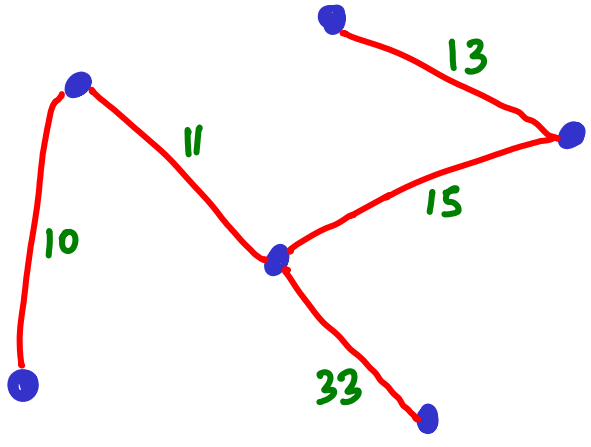






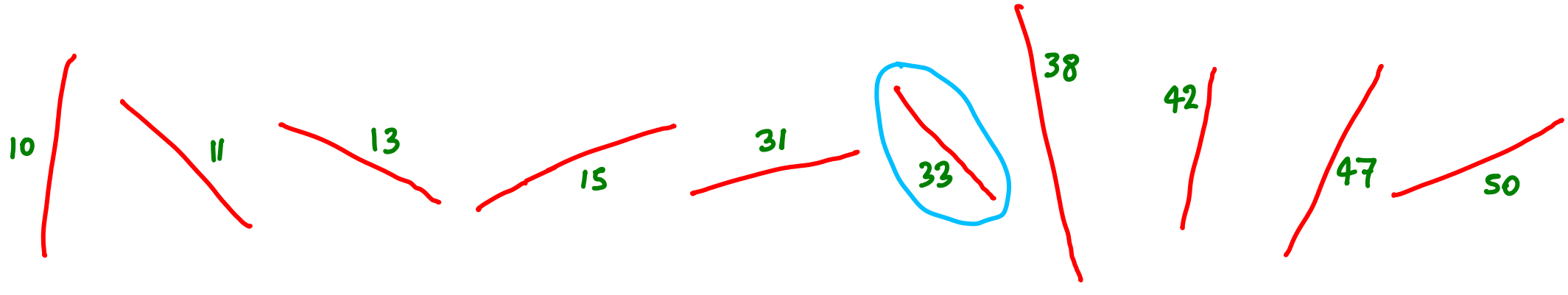
KRUSKAL'S ALGORITHM for MST



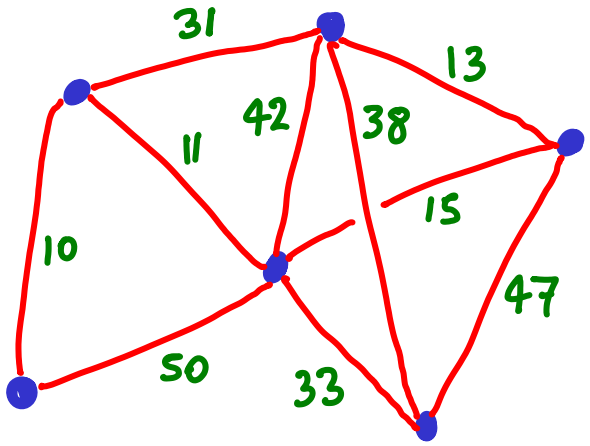


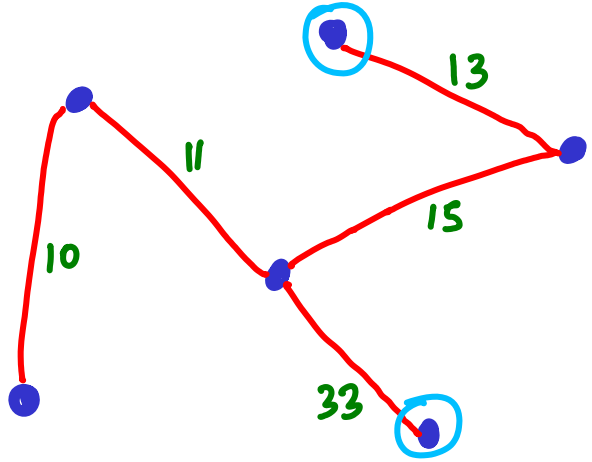
...DONE

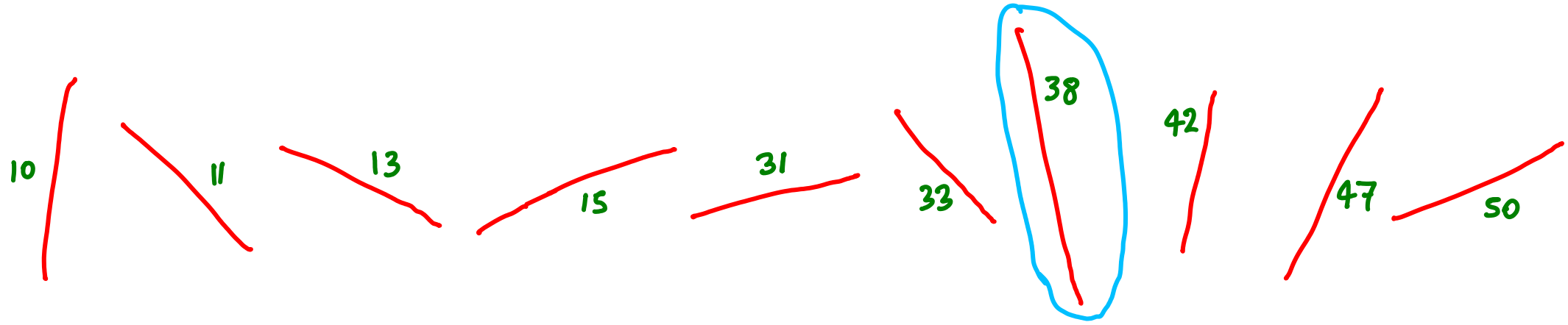
why?



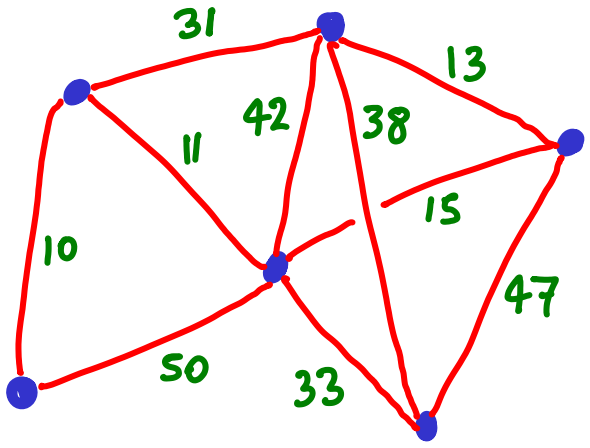
KRUSKAL'S ALGORITHM for MST

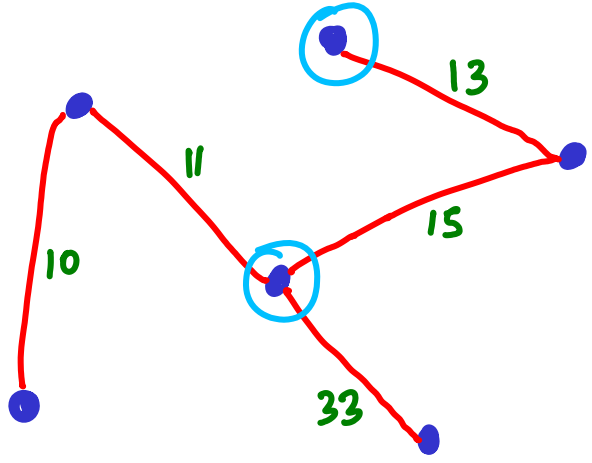


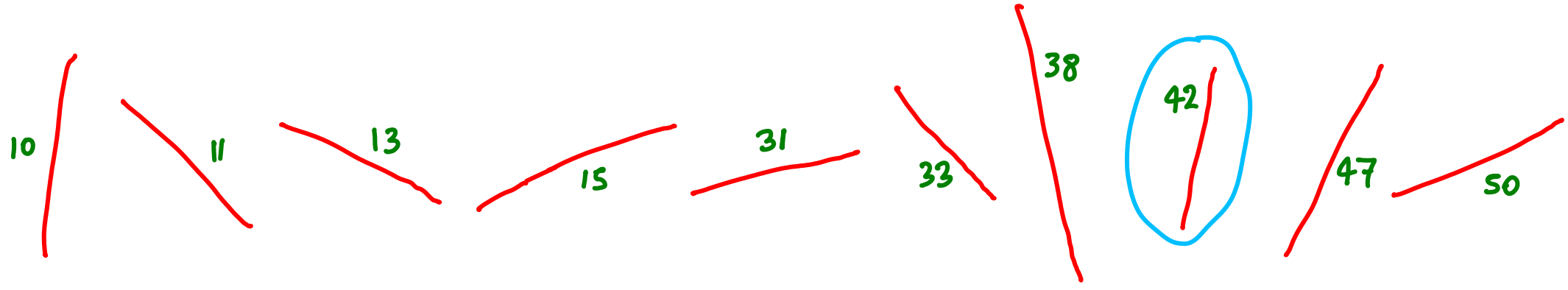




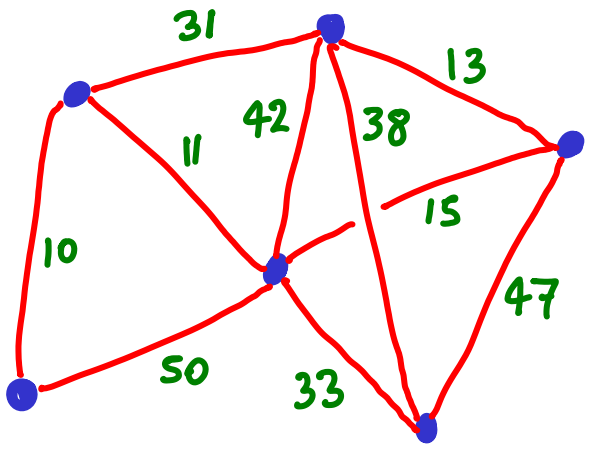
KRUSKAL'S ALGORITHM for MST

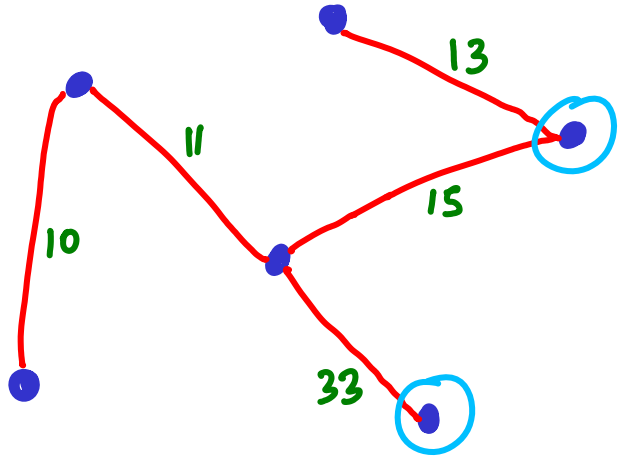


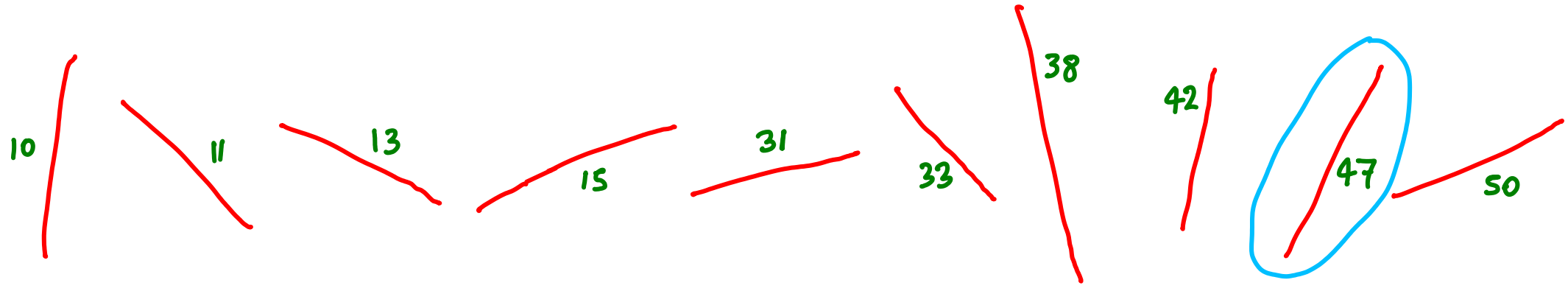




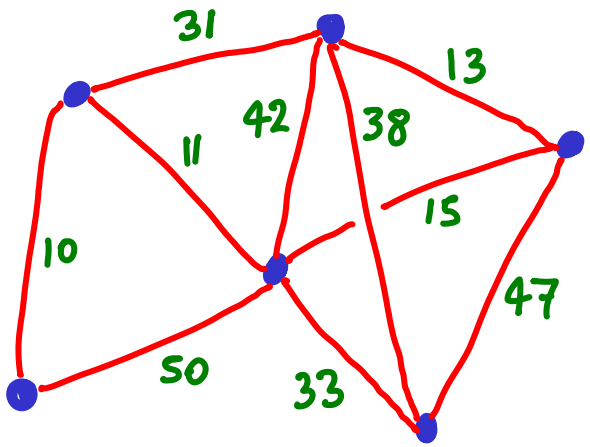
KRUSKAL'S ALGORITHM for MST

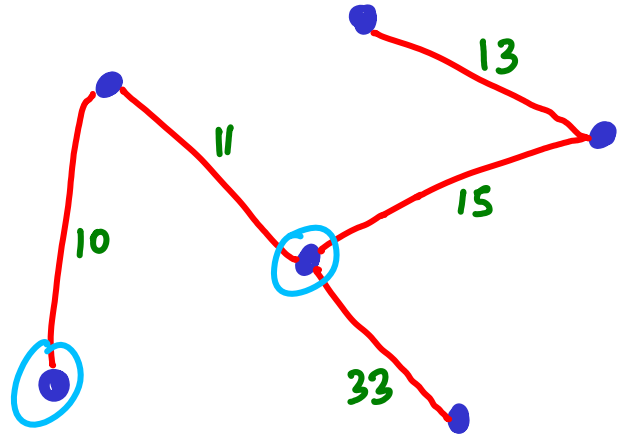




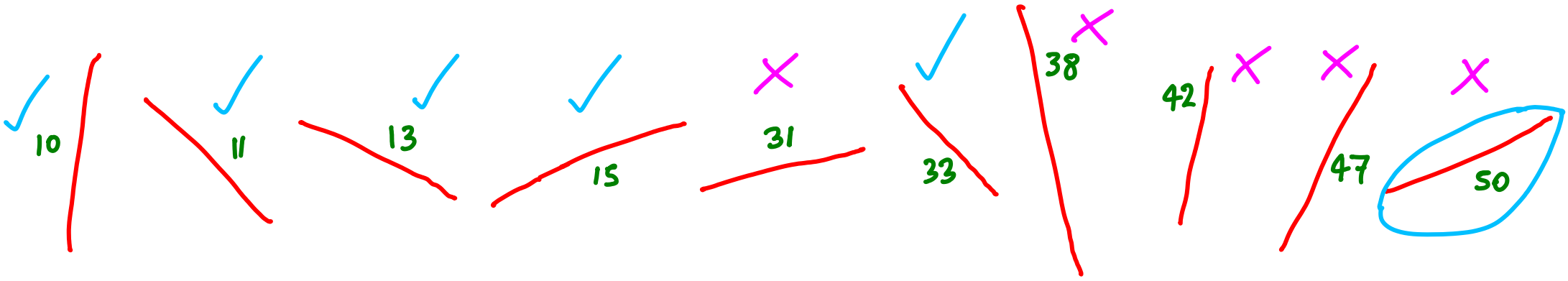


KRUSKAL'S ALGORITHM for MST



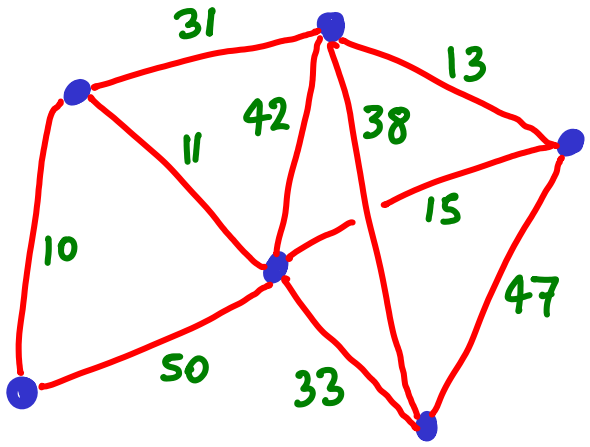


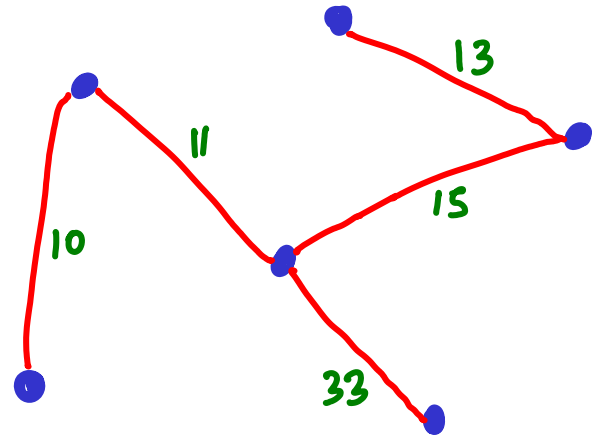
DONE

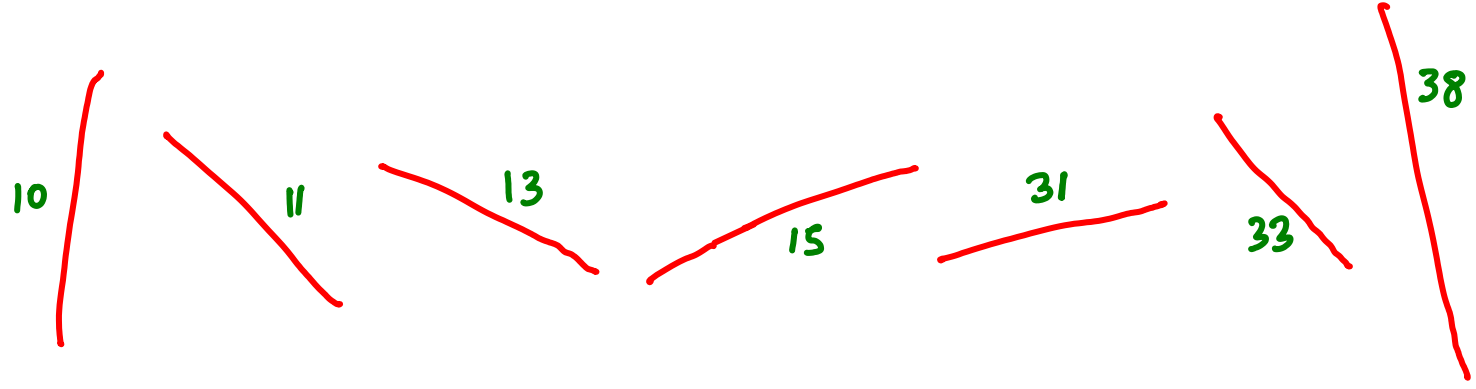


KRUSKAL'S ALGORITHM for MST

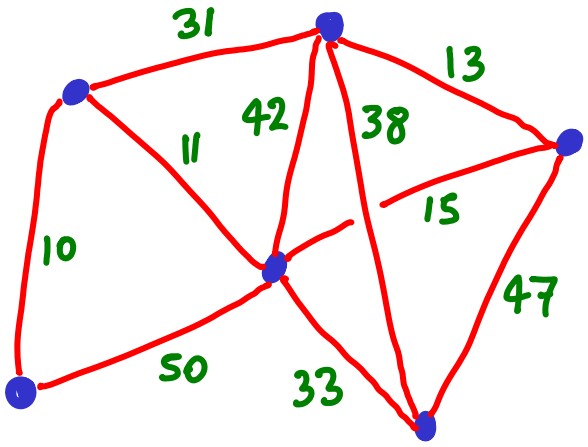
1) make forest of vertices





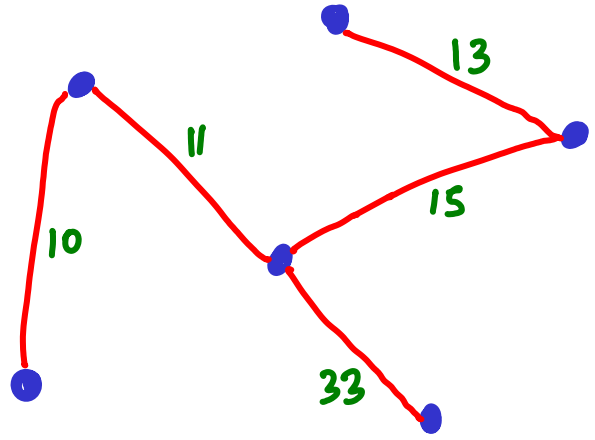


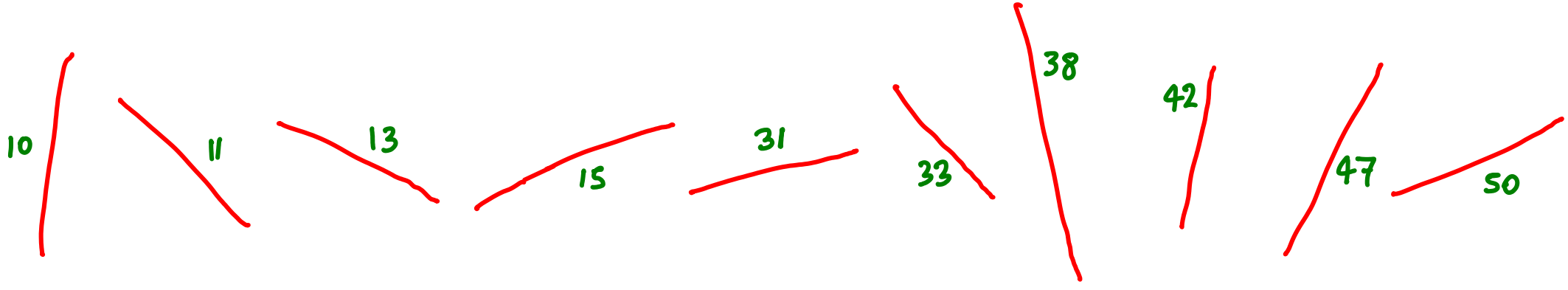
KRUSKAL'S ALGORITHM for MST



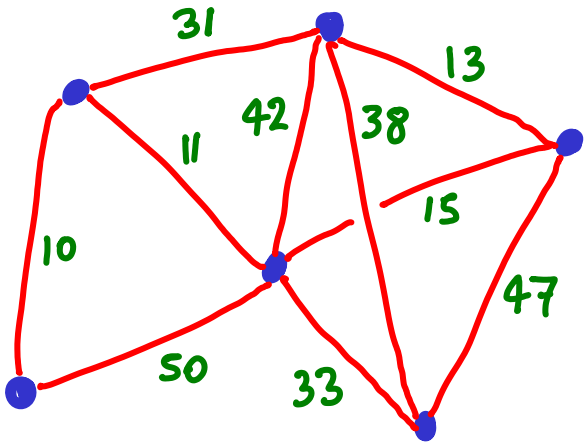
1) make forest of vertices
2) sort edges

time?

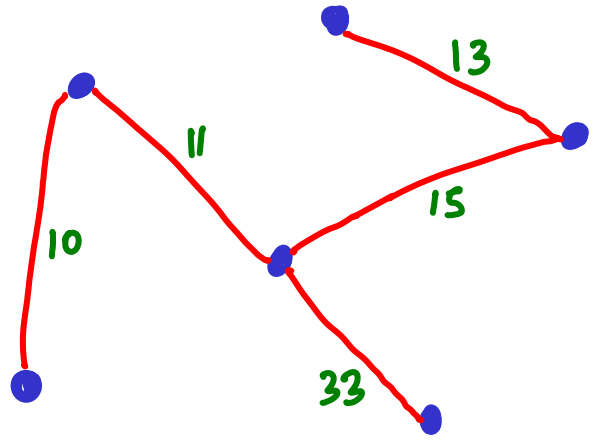


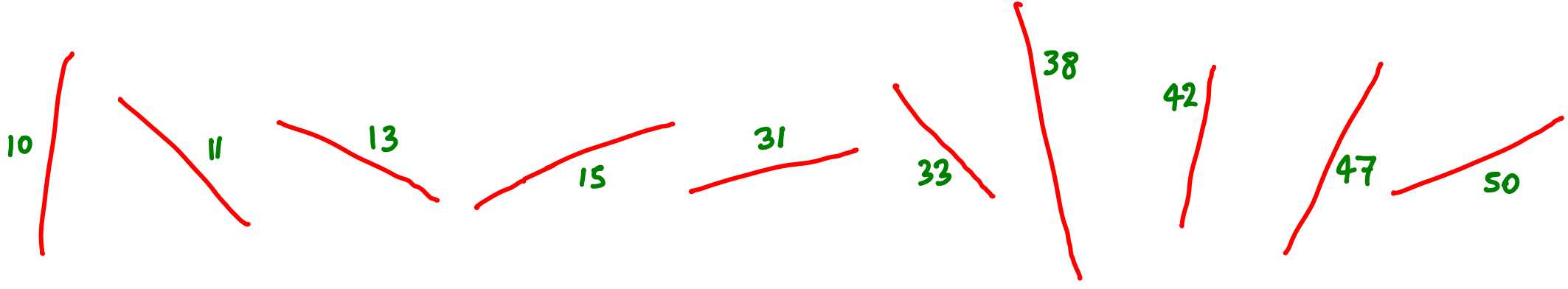


KRUSKAL'S ALGORITHM for MST

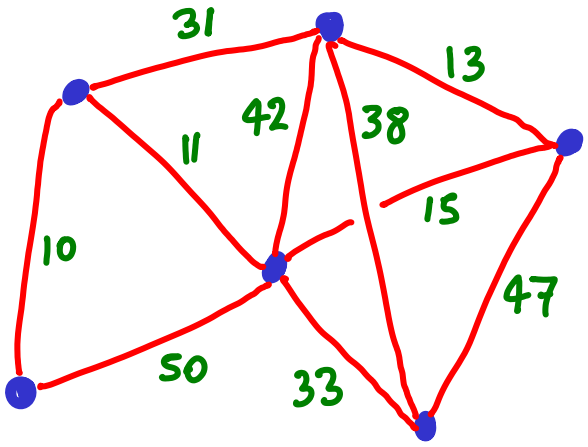


1) make forest of vertices $O(V)$
2) sort edges $O(E \log E)$



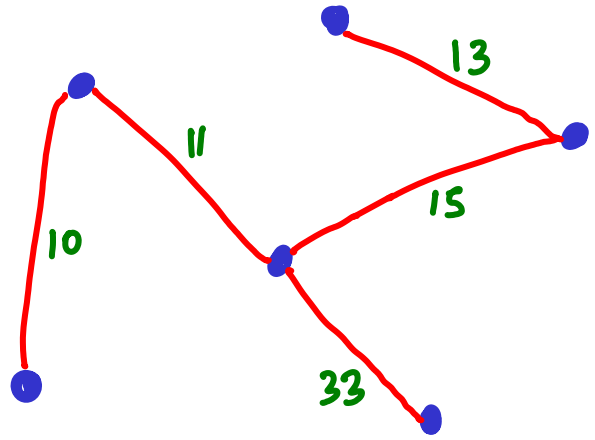


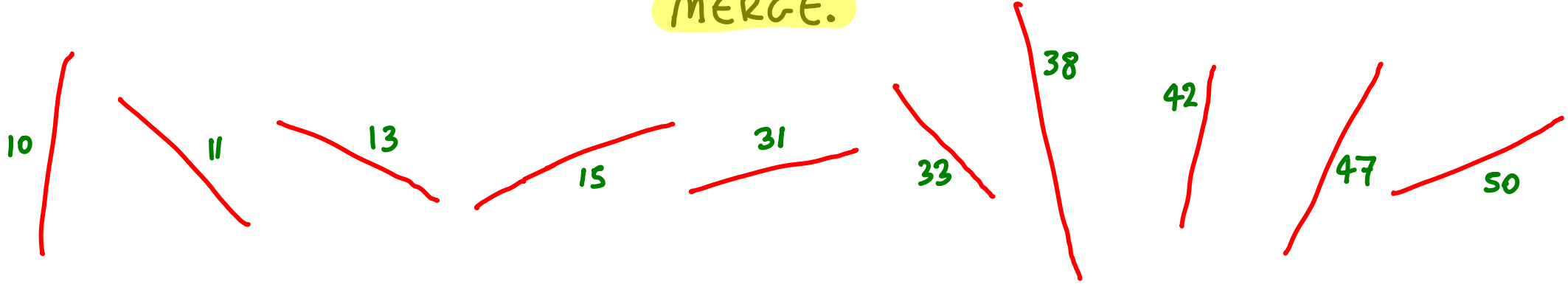
KRUSKAL'S ALGORITHM for MST



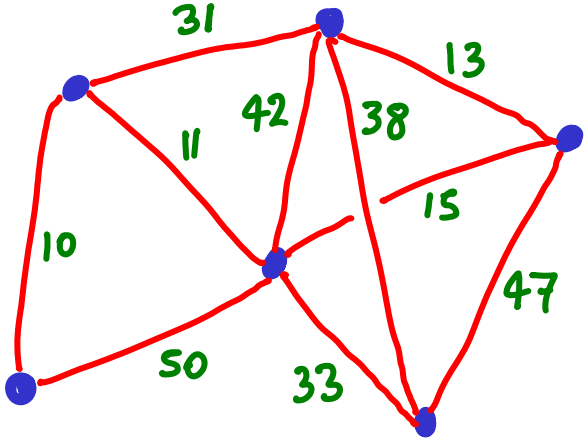
1) make forest of vertices $O(V)$
2) sort edges $O(E \log E)$
3) for each edge

CHECK ENDPOINTS
(same component?)
↳ No? Add edge.
MERGE.





KRUSKAL'S ALGORITHM for MST

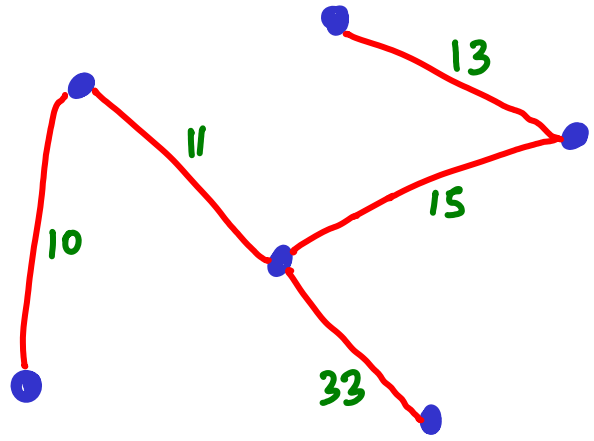


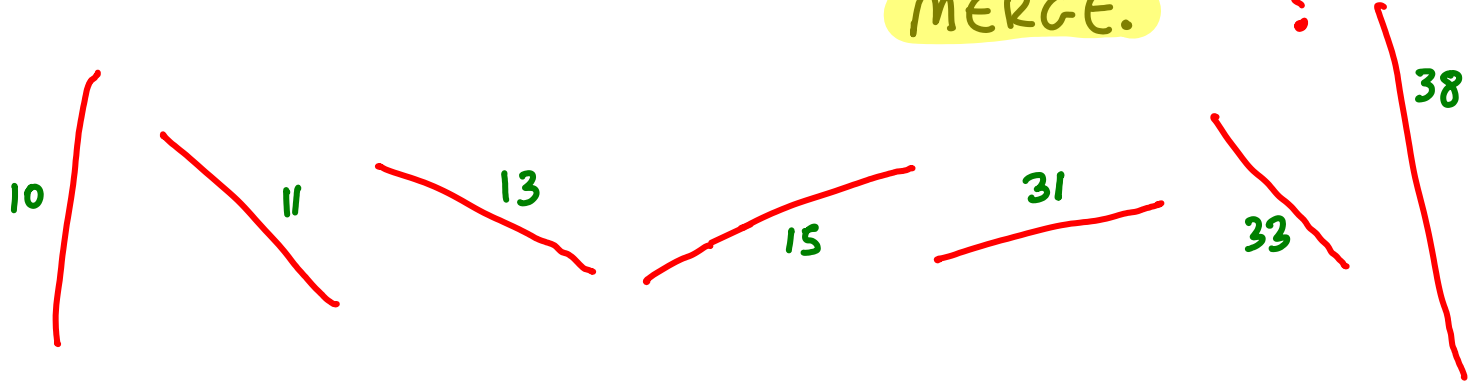
1) make forest of vertices O(V)
2) sort edges O(E log E)
3) for each edge .... O(E·?)

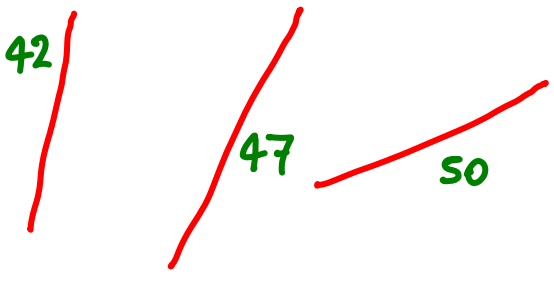
CHECK ENDPOINTS ?
(same component?)

↳ No? Add edge. O(1)

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

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

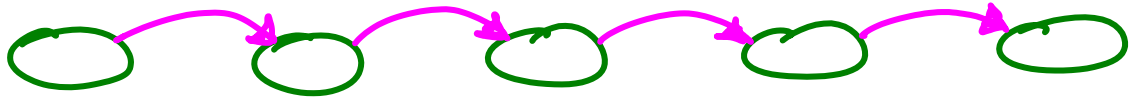
CHECK ENDPOINTS
(same component?)
↳ No? Add edge.
MERGE

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

↓ Kruskal's algo

↓ O(E) : O(V²)

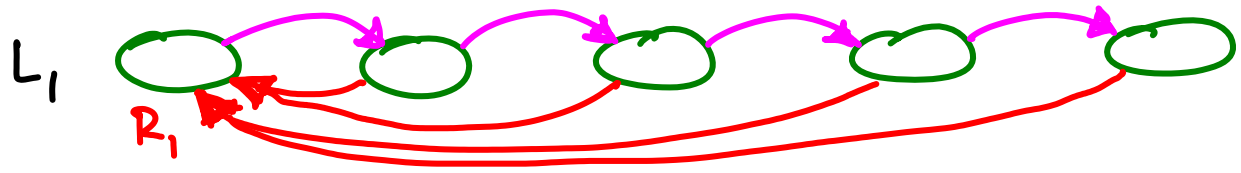
Store each component as a linked list

L_1 

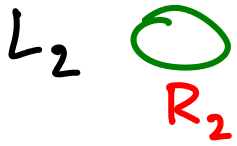
L_2 

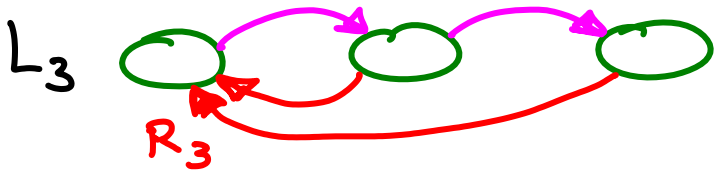
L_3 

Store each component as a linked list

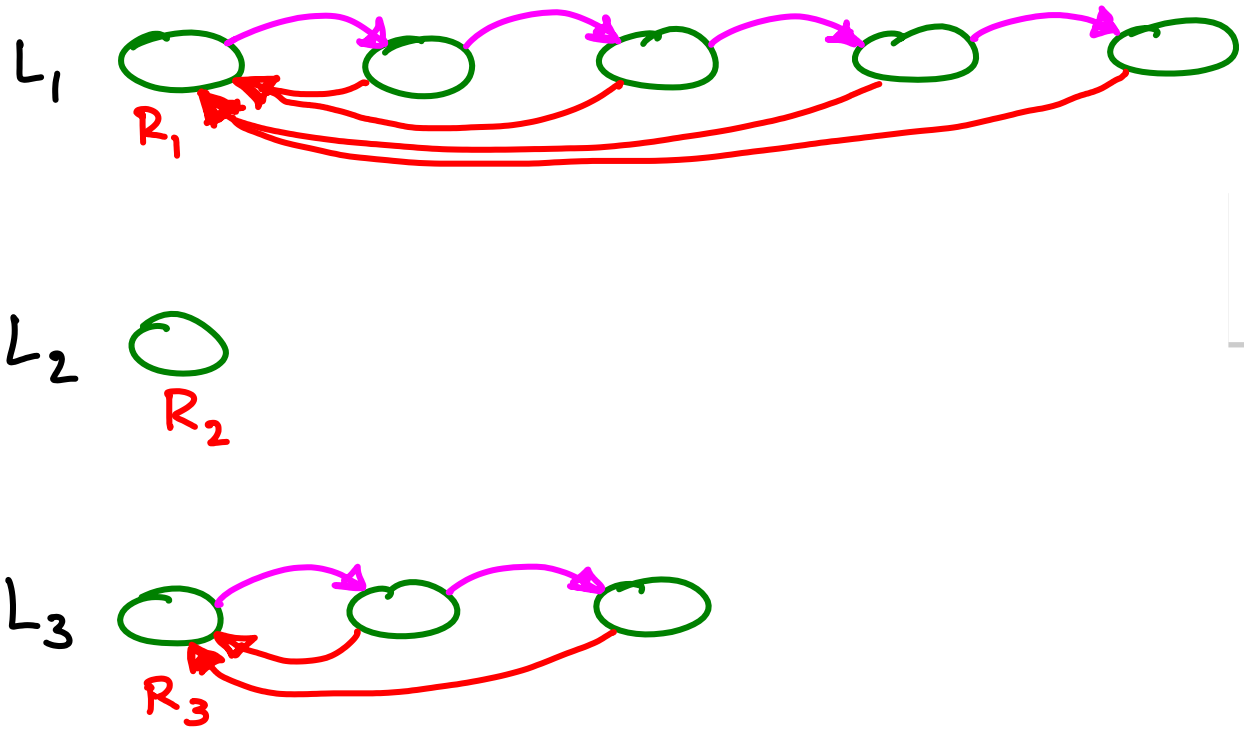


Every node points to one representative.





Store each component as a linked list

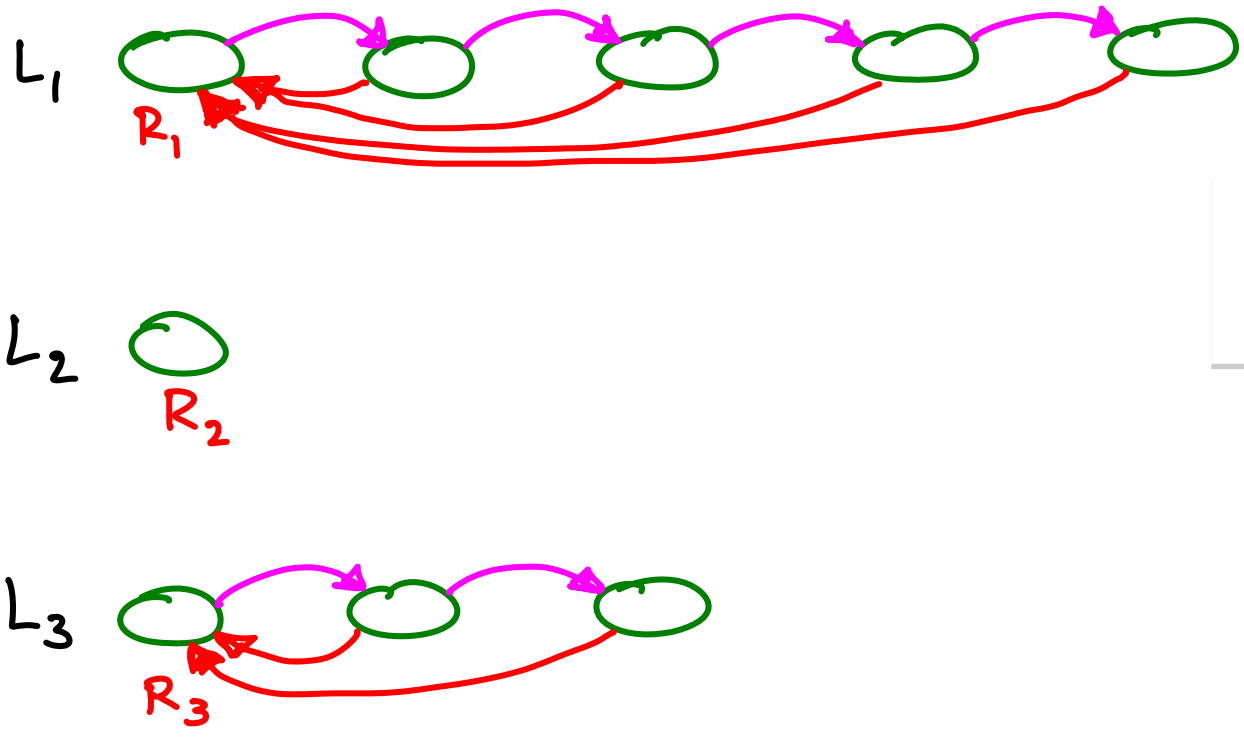


Every node points to one representative.

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

Store each component as a linked list

$n = |V|$



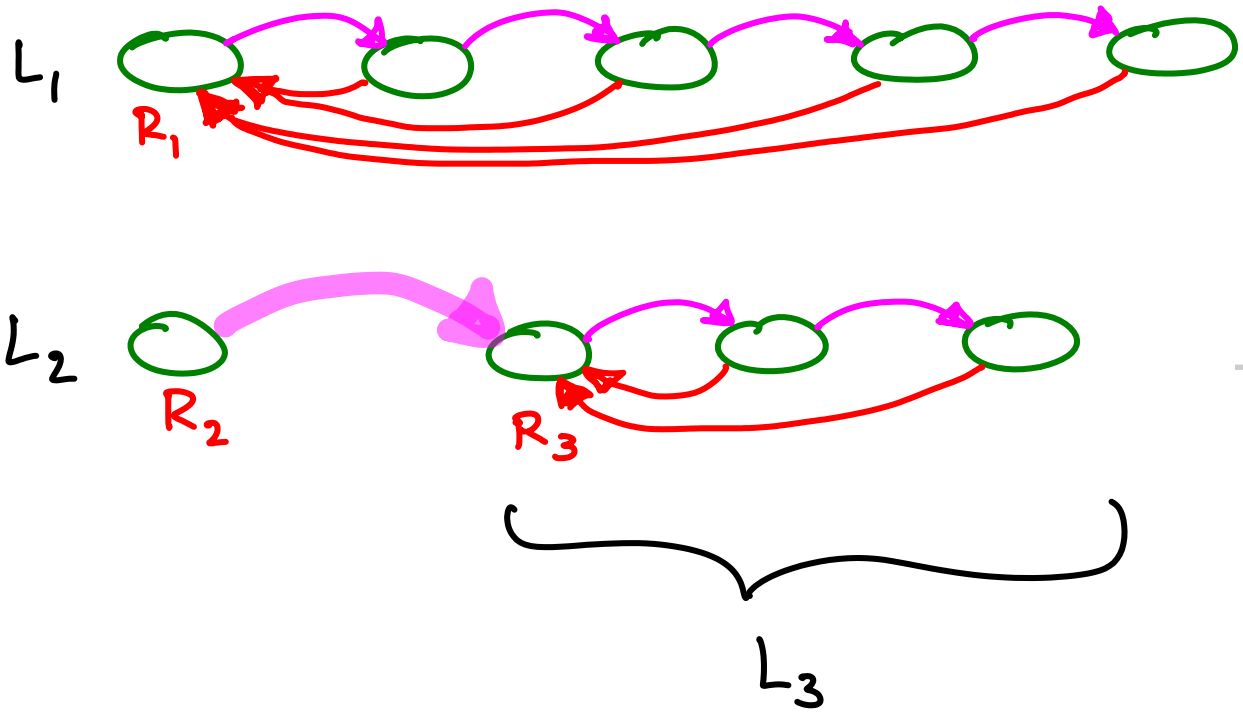
Every node points to one representative.

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

Merge : looks like $O(n)$

WHY?

Store each component as a linked list

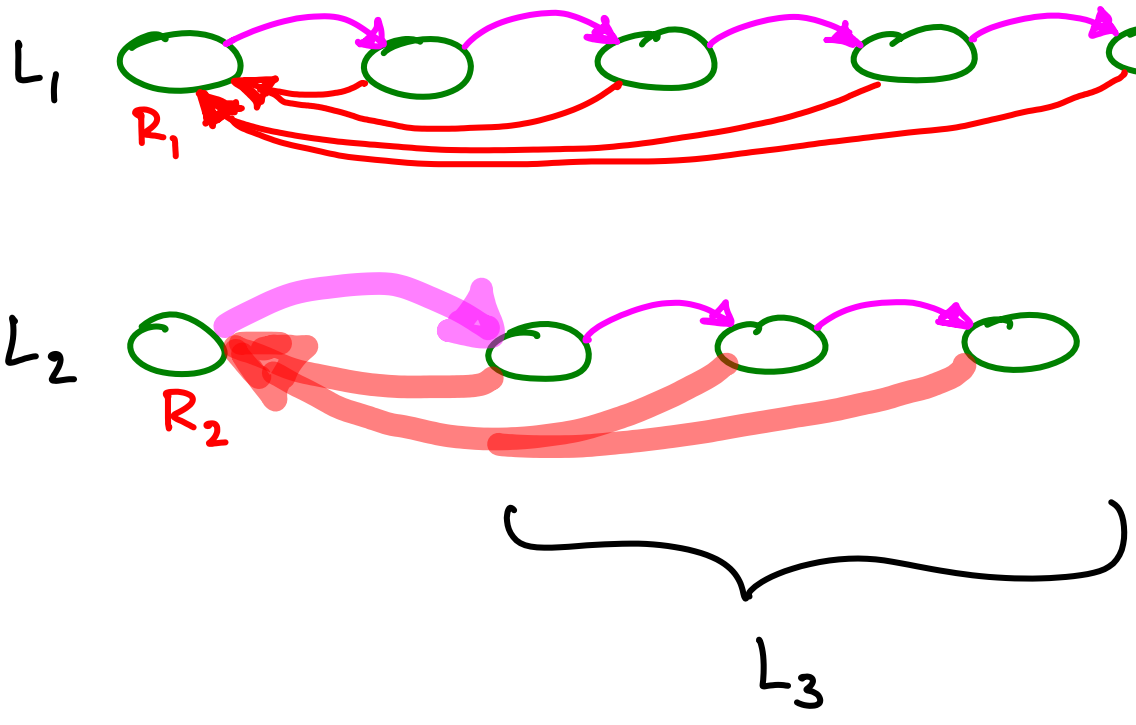


Every node points to one representative.

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

Merge : looks like $O(n)$
WHY?

Store each component as a linked list

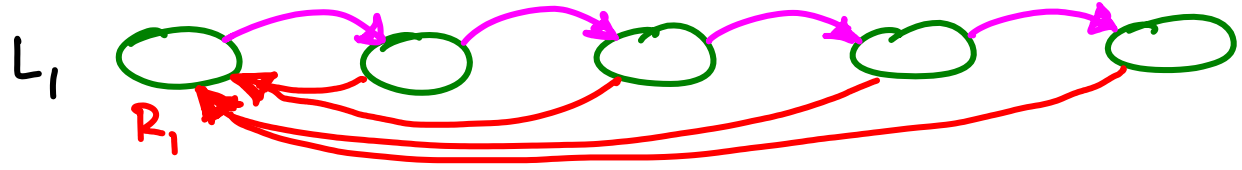


Every node points to one representative.

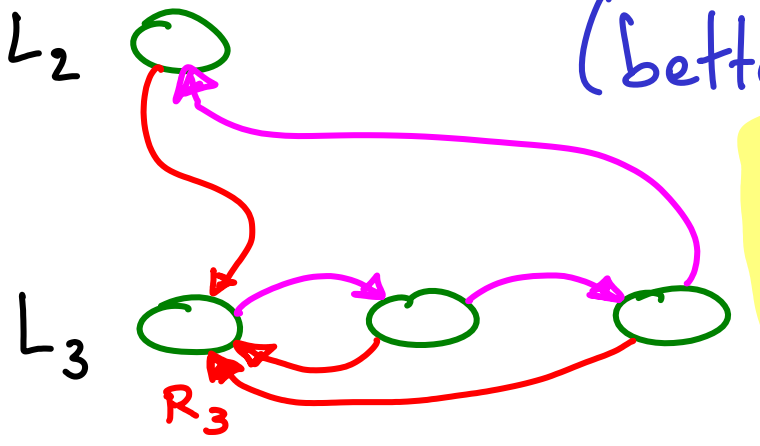
Query "same component":
check representatives : $O(1)$

Merge : looks like $O(n)$
WHY?

Store each component as a linked list



Every node points to one representative.



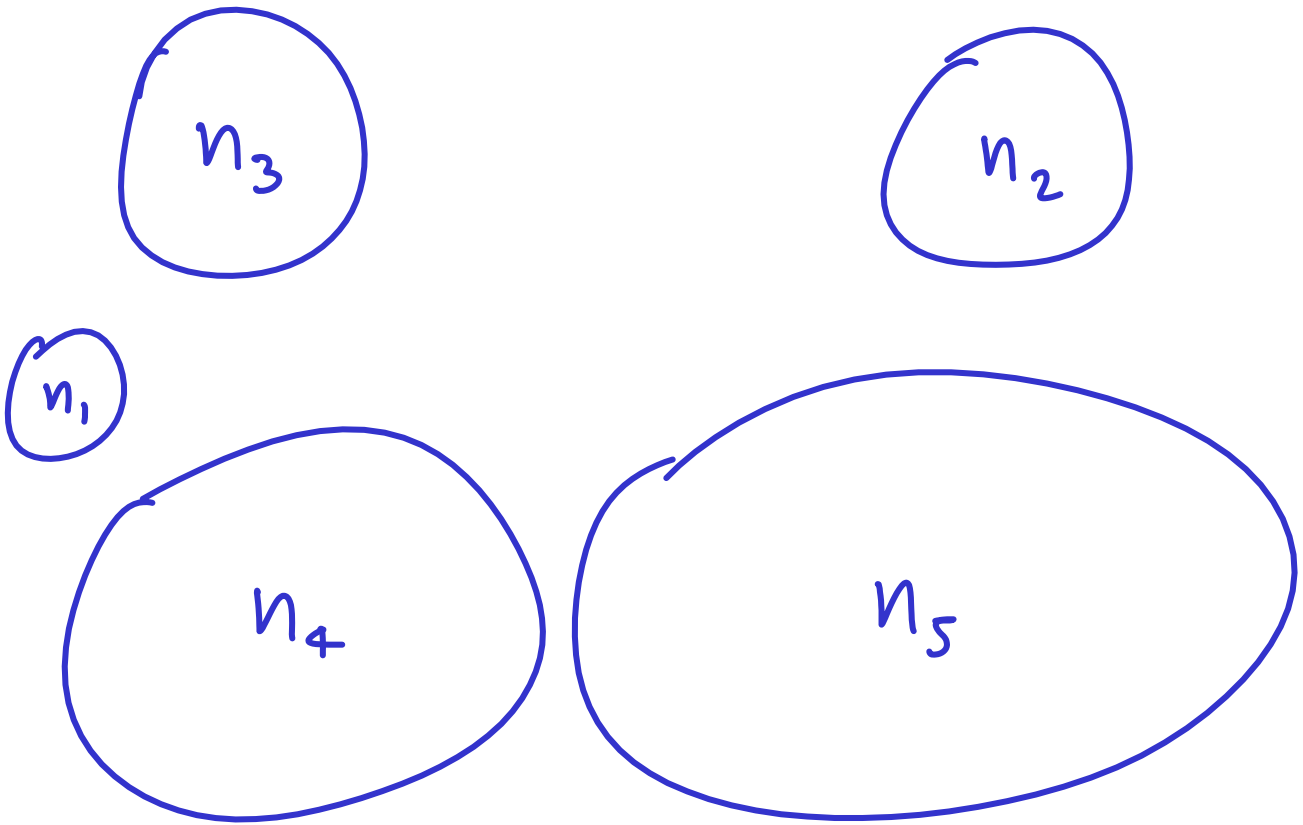
(better)

move small list to big list

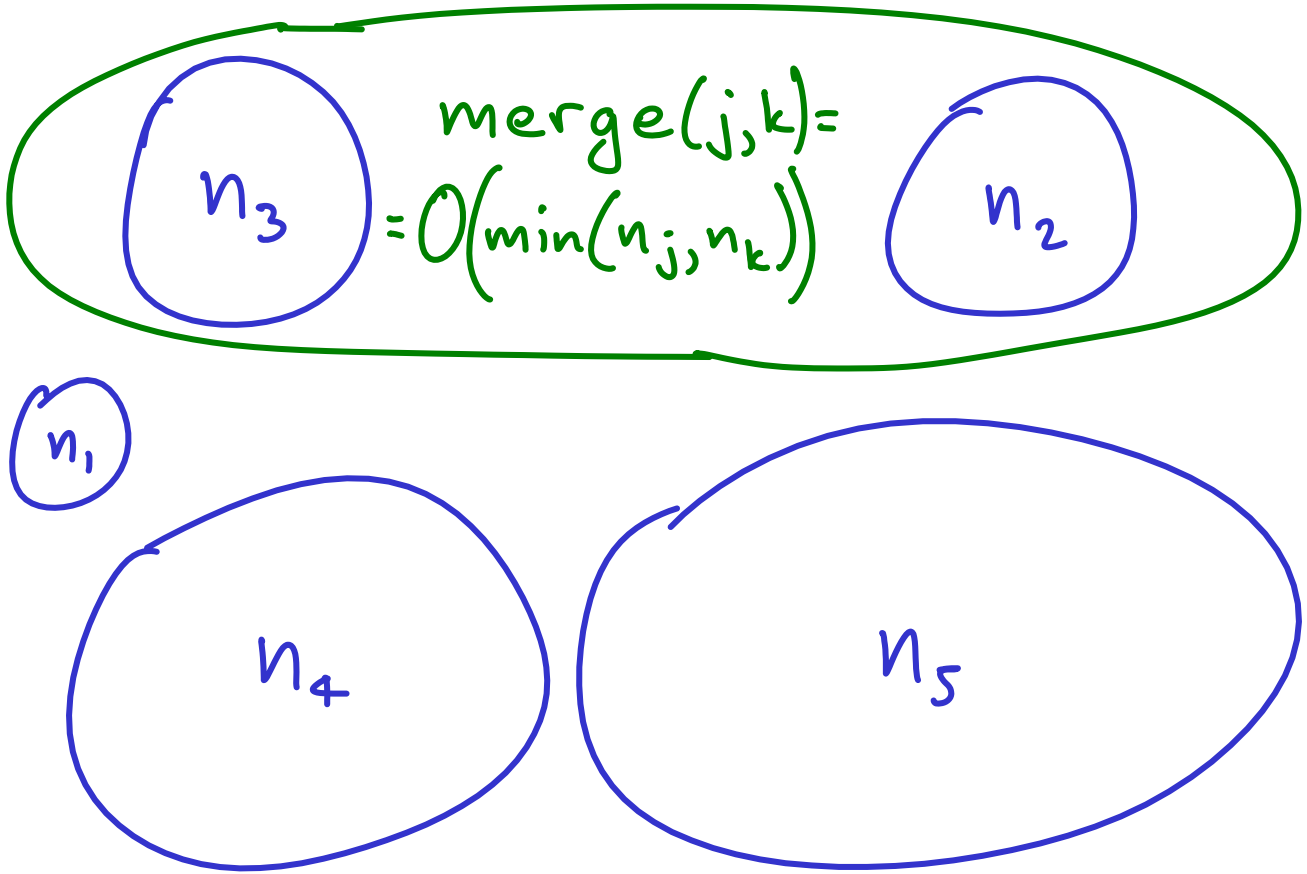
Query "same component":
check representatives : $O(1)$

Merge : looks like $O(n)$

$$\sum n_i = n$$



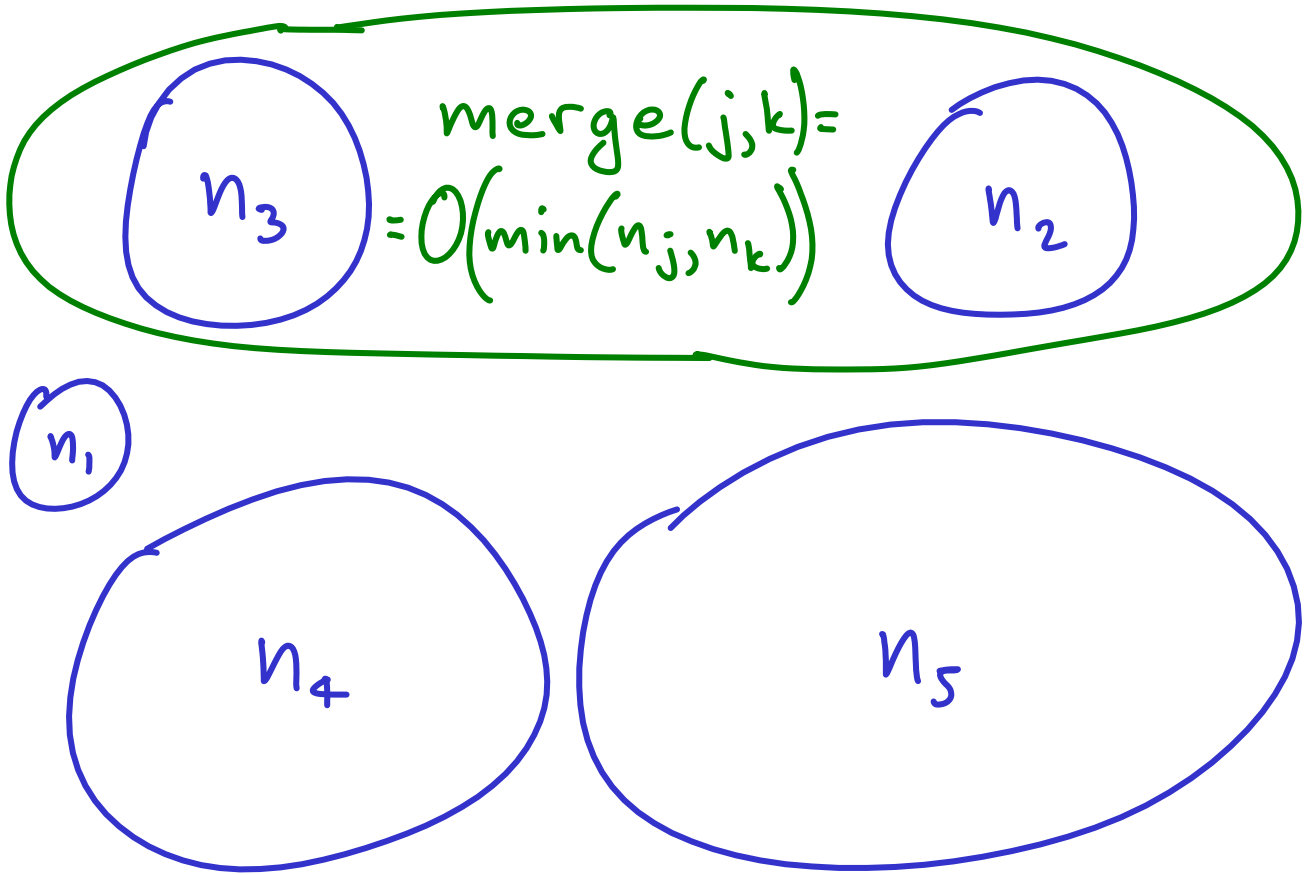
$$\sum n_i = n$$



$\sum n_i = n$

How many times can an item merge?

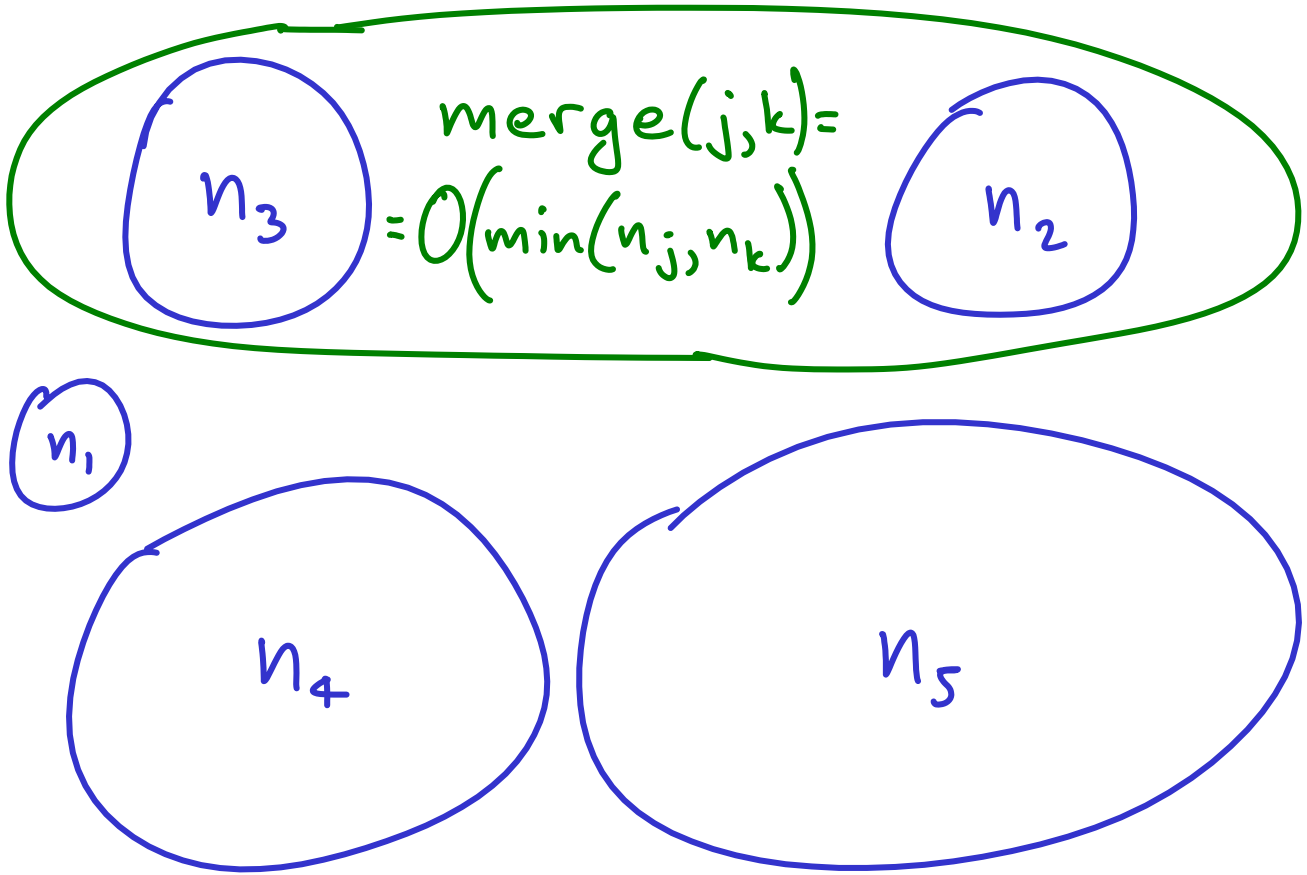
(= how many times can an item change representatives?)



$\sum n_i = n$

How many times can an item merge?

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



Component size
$\geq$ doubles each time

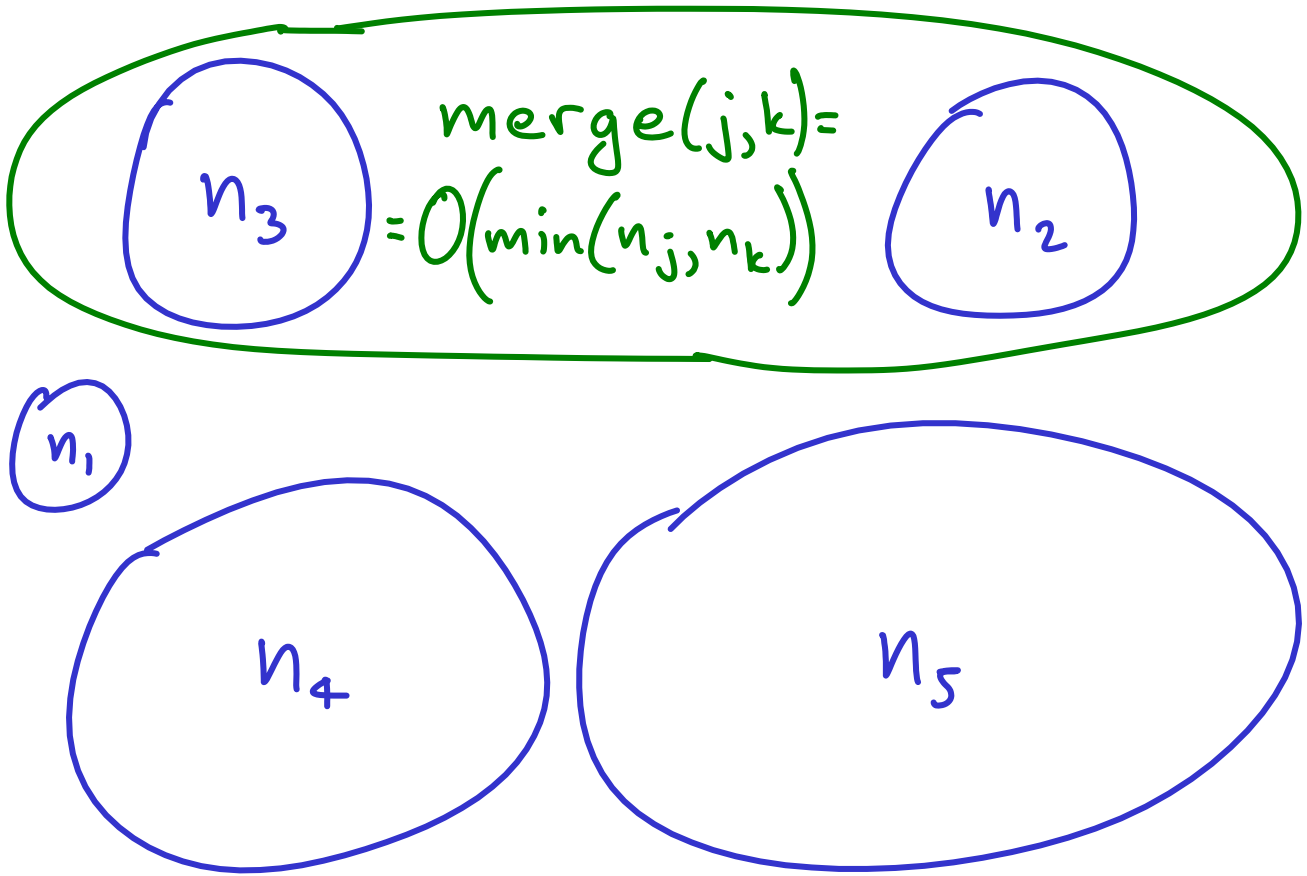
$\sum n_i = n$

How many times can an item merge?

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

→ $O(\log n)$

Component size doubles each time



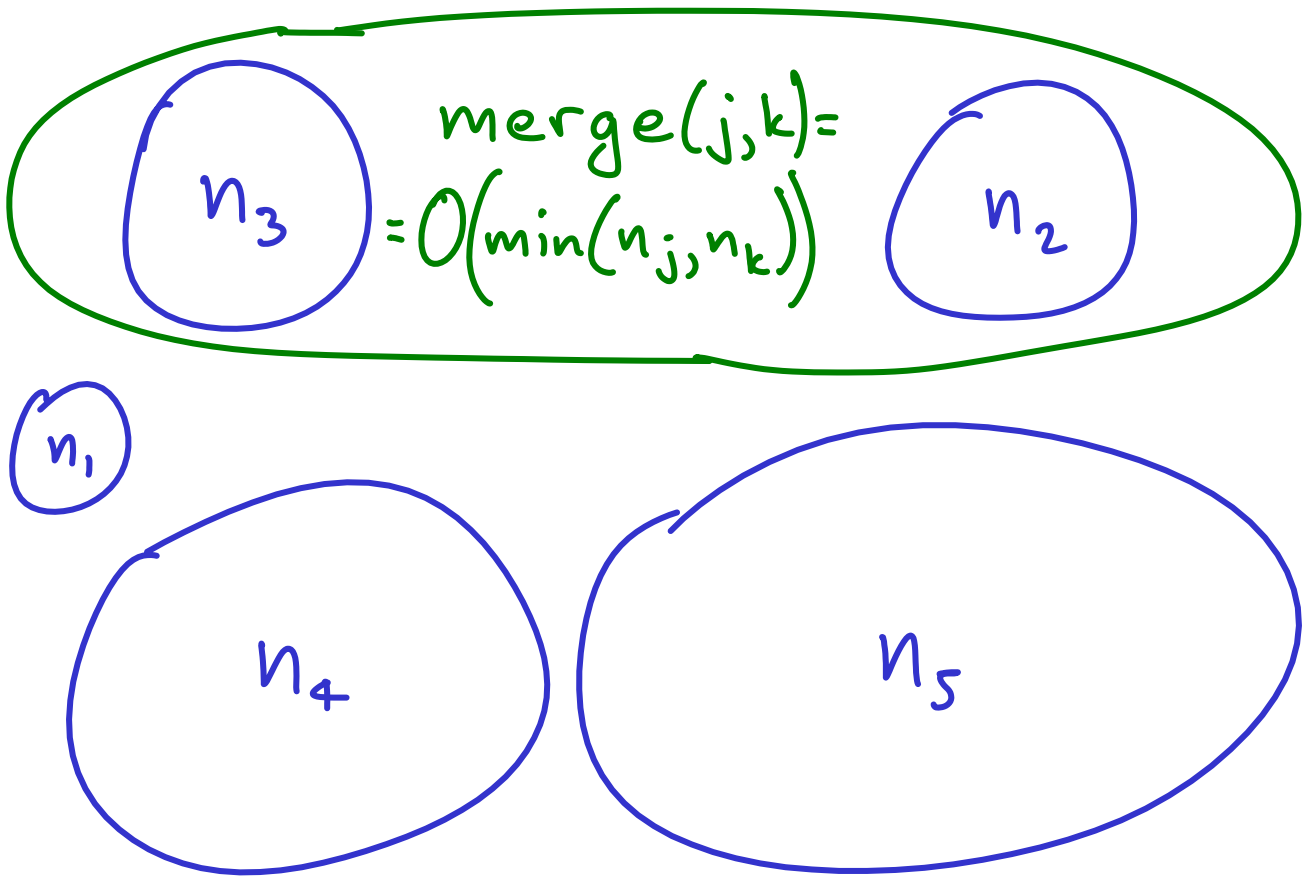
$\sum n_i = n$

How many times can an item merge?

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

→ $O(\log n)$

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?) → O(1)

↳ No? Add edge. MERGE → O(1)

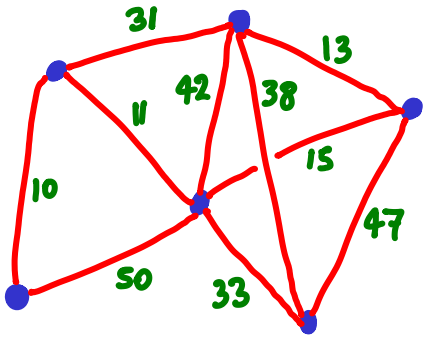
→ O(V) worst case
→ O(V log V) total

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

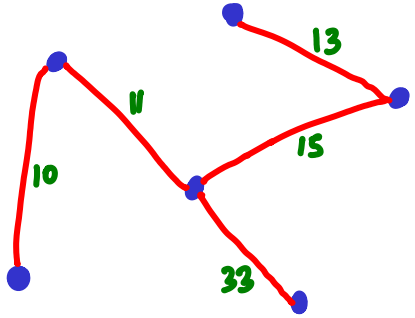
↓ Kruskal's algo

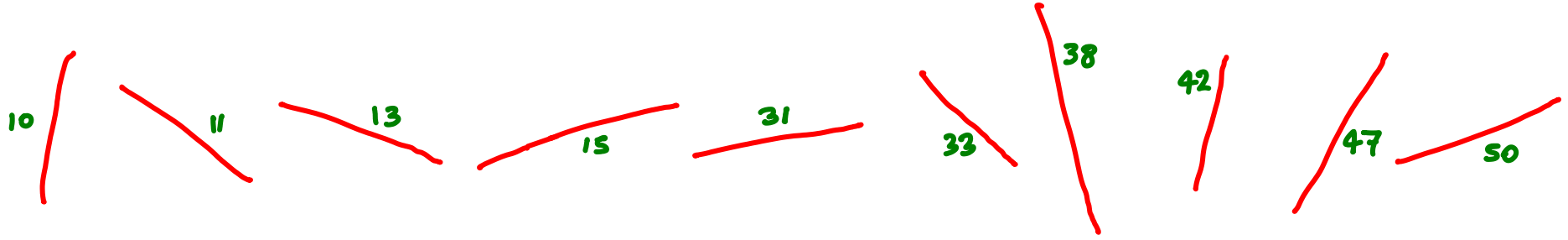
↓ O(E) checks
O(V) merges

KRUSKAL'S ALGORITHM for MST

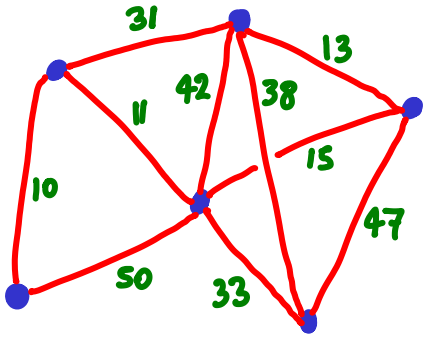


1) make forest of vertices $O(V)$
2) sort edges $O(E \log E)$
3) for each edge
   CHECK ENDPOINTS (same component?) } $O(E)$ total
   ↳ No? Add edge. MERGE → $O(V \log V)$ total





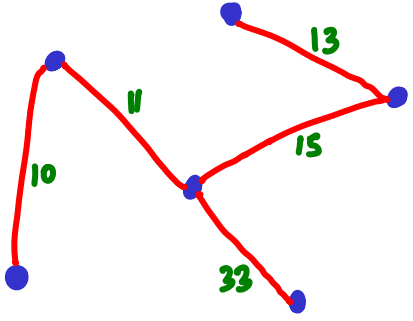
KRUSKAL'S ALGORITHM for MST

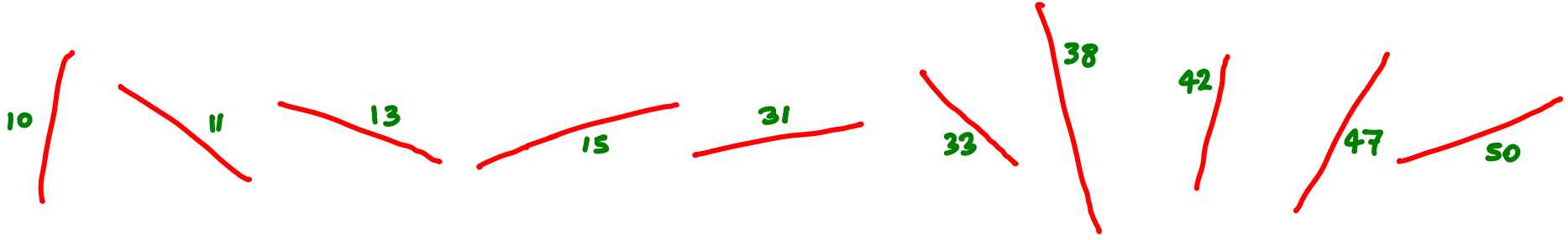


1) make forest of vertices $O(V)$
2) sort edges $O(E \log E) = O(E \log V)$ why?
3) for each edge
   CHECK ENDPOINTS (same component?)
   ↳ No? Add edge. MERGE

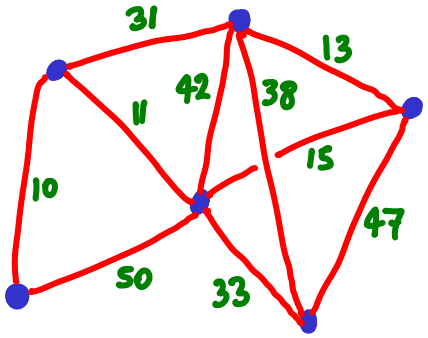
$O(E)$ total

$O(V \log V)$ total

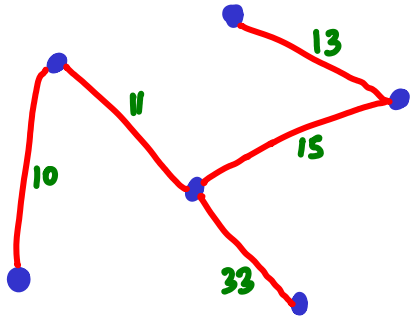


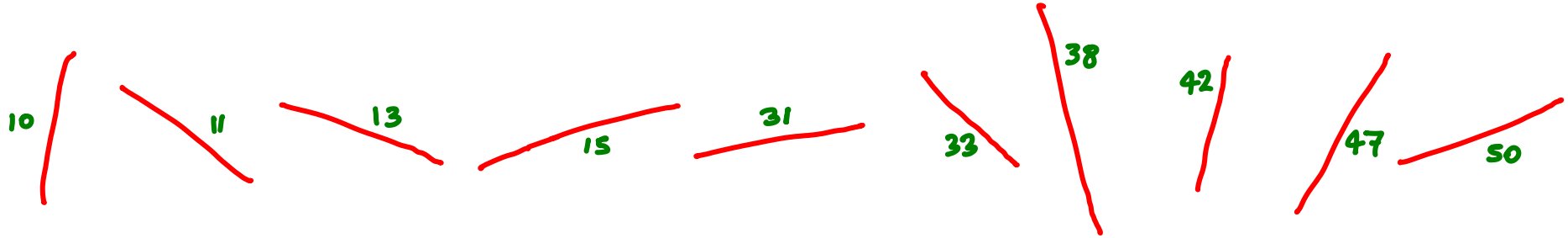


KRUSKAL'S ALGORITHM for MST

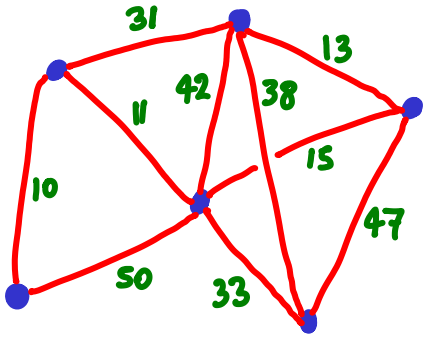


1) make forest of vertices $O(V)$
2) sort edges $O(E \log E) = O(E \log V)$ $\log E = O(\log V^2)$
3) for each edge
   CHECK ENDPOINTS (same component?) } $O(E)$ total
   ↳ No? Add edge. MERGE → $O(V \log V)$ total





KRUSKAL'S ALGORITHM for MST

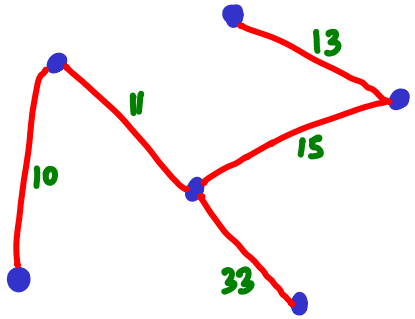


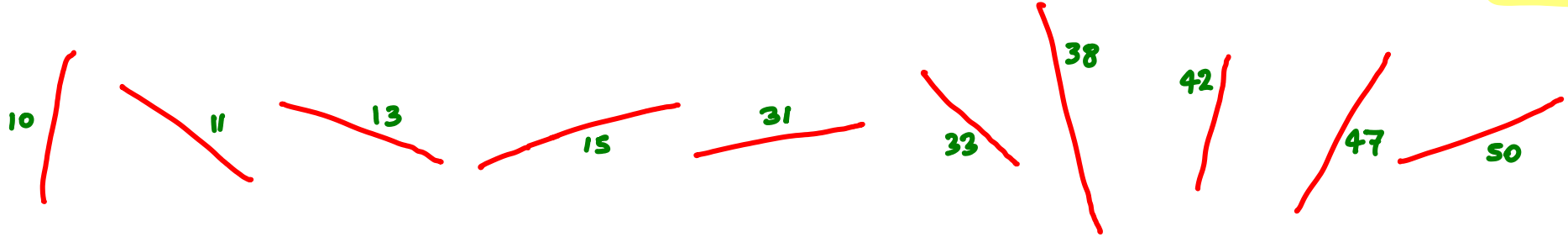
1) make forest of vertices $O(V)$
2) sort edges $O(E \log E) = O(E \log V)$ $\log E = O(\log V^2)$
3) for each edge
   CHECK ENDPOINTS (same component?)
   ↳ No? Add edge. MERGE

$O(E)$ total

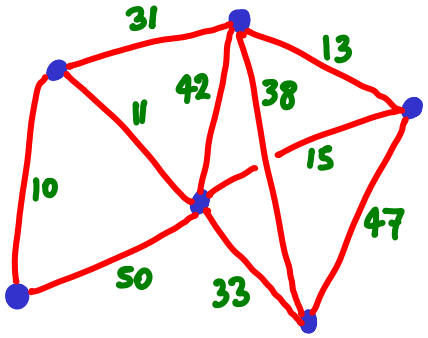
$O(V \log V)$ total $= O(E \log V)$

why?





KRUSKAL'S ALGORITHM for MST



1) make forest of vertices $O(V)$

2) sort edges $O(E \log E) = O(E \log V)$ $\log E = O(\log V^2)$

3) for each edge

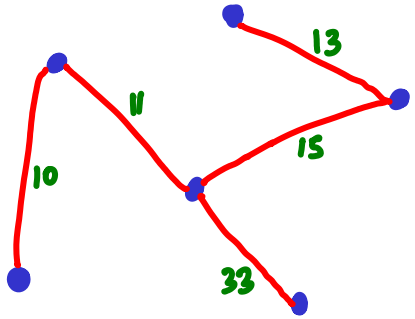
CHECK ENDPOINTS (same component?)

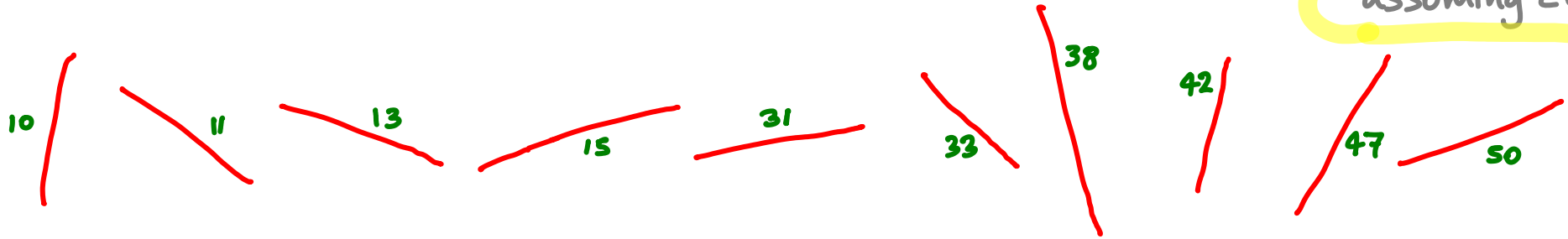
↳ No? Add edge. MERGE

} $O(E)$ total

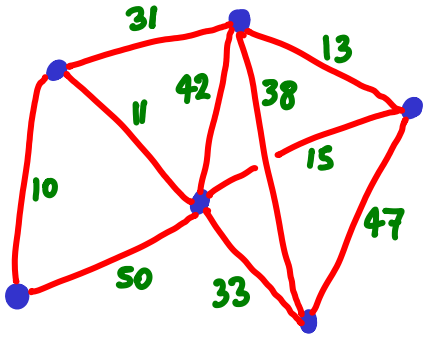
$O(V \log V)$ total $= O(E \log V)$

$E \geq V-1$ assuming connected G





KRUSKAL'S ALGORITHM for MST → 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?)
   ↳ No? Add edge. MERGE

$O(E)$ total

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

$E \geq V-1$
assuming connected G