MINIMUM SPANNING TREE ALGORITHMS

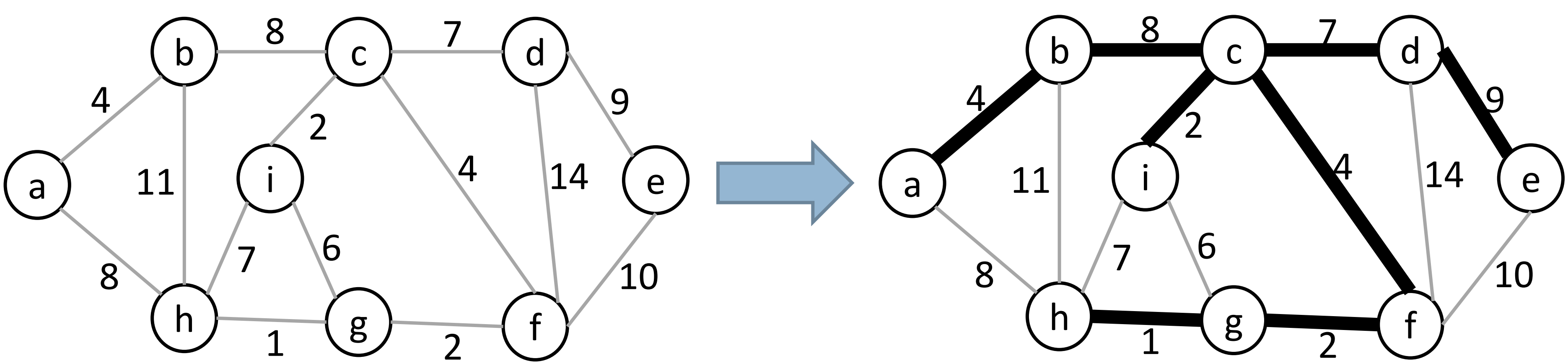
Mohammad Minhazul Haq

Computer Science & Engineering

The University of Texas at Arlington

Minimum Spanning Tree

- **Given:**
  - a Connected, Undirected graph G(V, E)
  - each edge $(\mathrm{u}, \mathrm{v}) \in E$ has a weight w(u, v)
- **Find:**
  - Acyclic subset (tree) T ⊆ E that:
    - connects all vertices, and
    - total weight $\mathrm{w(T)} = \sum_{(u,v)\in T} w(u, v)$ is minimized



Algorithm 1: Kruskal

<u>MST-KRUSKAL (G, w)</u>

```
A = Ø
for each vertex v ∈ G.V
        MAKE-SET(v)
sort the edges of G.E into non-decreasing order by weight w
for each edge (u, v) ∈ G.E taken in non-decreasing order by weight
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
return A
```

O(1)

|V| MAKE-SET

O(E lgE)

O(E)
FIND-SET &
UNION

<u>Running time:</u>

O(E lgE) + O((V+E)$\alpha(V)$), where α is very slowly growing function

= O(E lgE) + O(E$\alpha(V)$), since G is connected |E|≥|V|-1

= O(E lgE) + O(E lgV)

= O(E lgE) + O(E lgE)

= **O(E lgE)**

Algorithm 2: Prim

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

O(V)

O(V), for Binary Heap

O(V lgV), for Binary-heap
$O(V^2)$, for Linear-search

O(E lgV), for Adj. List + Binary Heap
$O(V^2)$, for Adj. Matrix + Linear Search

<u>**Running time (Adj. List + Binary Heap):**</u>

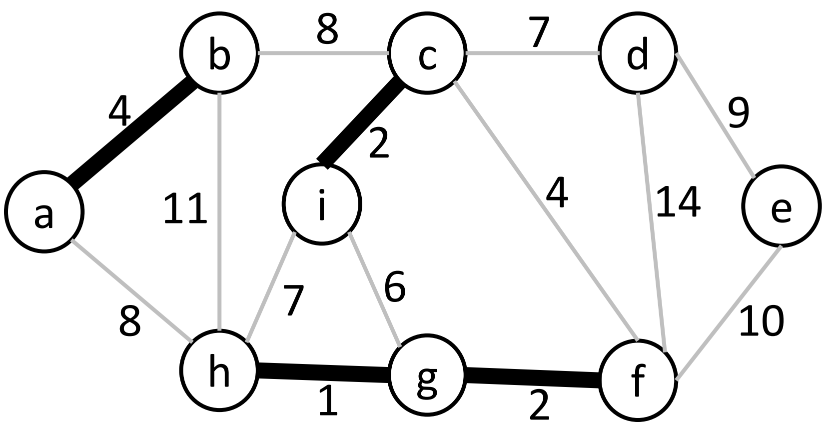
O(V lgV) + O(E lgV)

= **O(E lgV)**

<u>**Running time (Adj. Matrix + Linear Search):**</u>

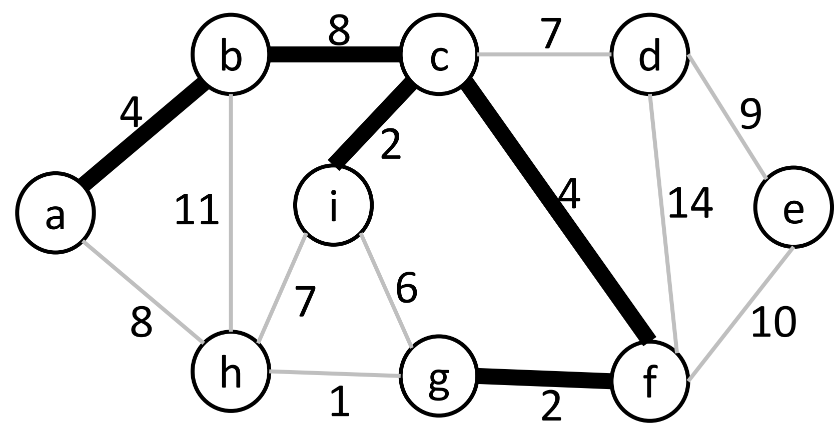
$O(V^2)$

Visualization of an Intermediate step



A = {(g,h), (c,i), (f, g), (a,b)}

Kruskal's algorithm



Q = {h, d, e}

Prim's algorithm

Similarities between Kruskal & Prim

- **Greedy algorithm**
- Follow a **GENERIC-MST(G, w)** algorithm
- Solution Tree (T) may or may not be **unique**
  - Kruskal’s algorithms
    - Two edges with same weight, w
  - Prim’s algorithm
    - Arbitrary root vertex, r
    - Two vertices in min-priority-queue having same keys

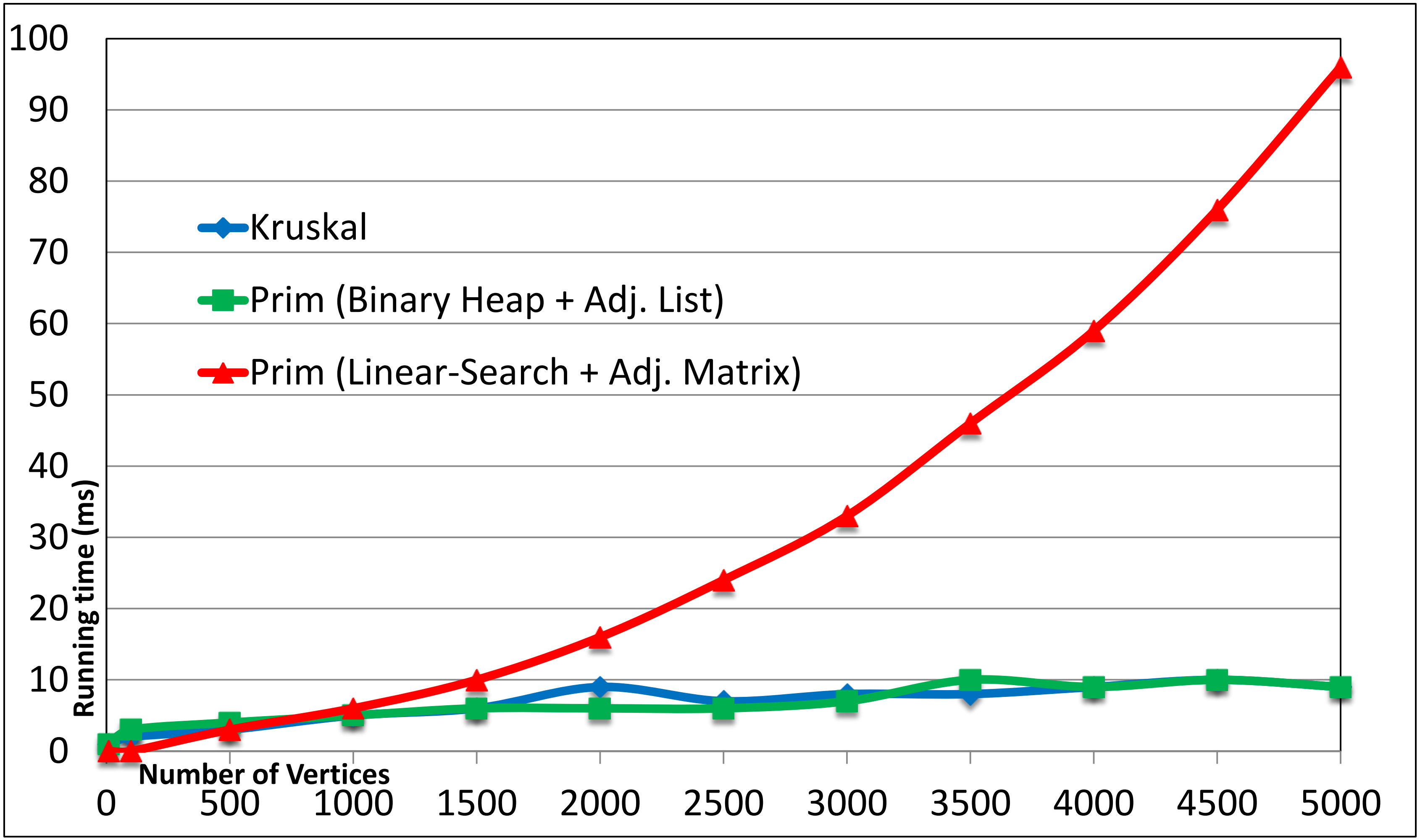
Differences between Kruskal & Prim

	Kruskal algorithm	Prim algorithm
Input	(G, w)	(G, w, r)
Output	A	No output. We can get MST from: A = {(v, v.π): v∈V-{r}}
Growing subset of a MST, A	a forest (one/more components)	a single tree
Safe Edge added to A	a least-weight edge that connects two distinct components	a least-weight edge that connects the tree to a vertex not in the tree
Running time	Disjoint-set & Edge-list: **O(E lgE)**	Binary min-heap & Adj. List: **O(E lgV)** Linear-Search & Adj. Matrix: **O(V^2)**
Running time dependency on Data Structures	Depends on the implementation of Disjoint-set data structure (Union-by-rank and Path-compression heuristic)	Depends on: 1. the implementation of min-priority-queue, Q 2. how we store the graph

Which one is better? When?

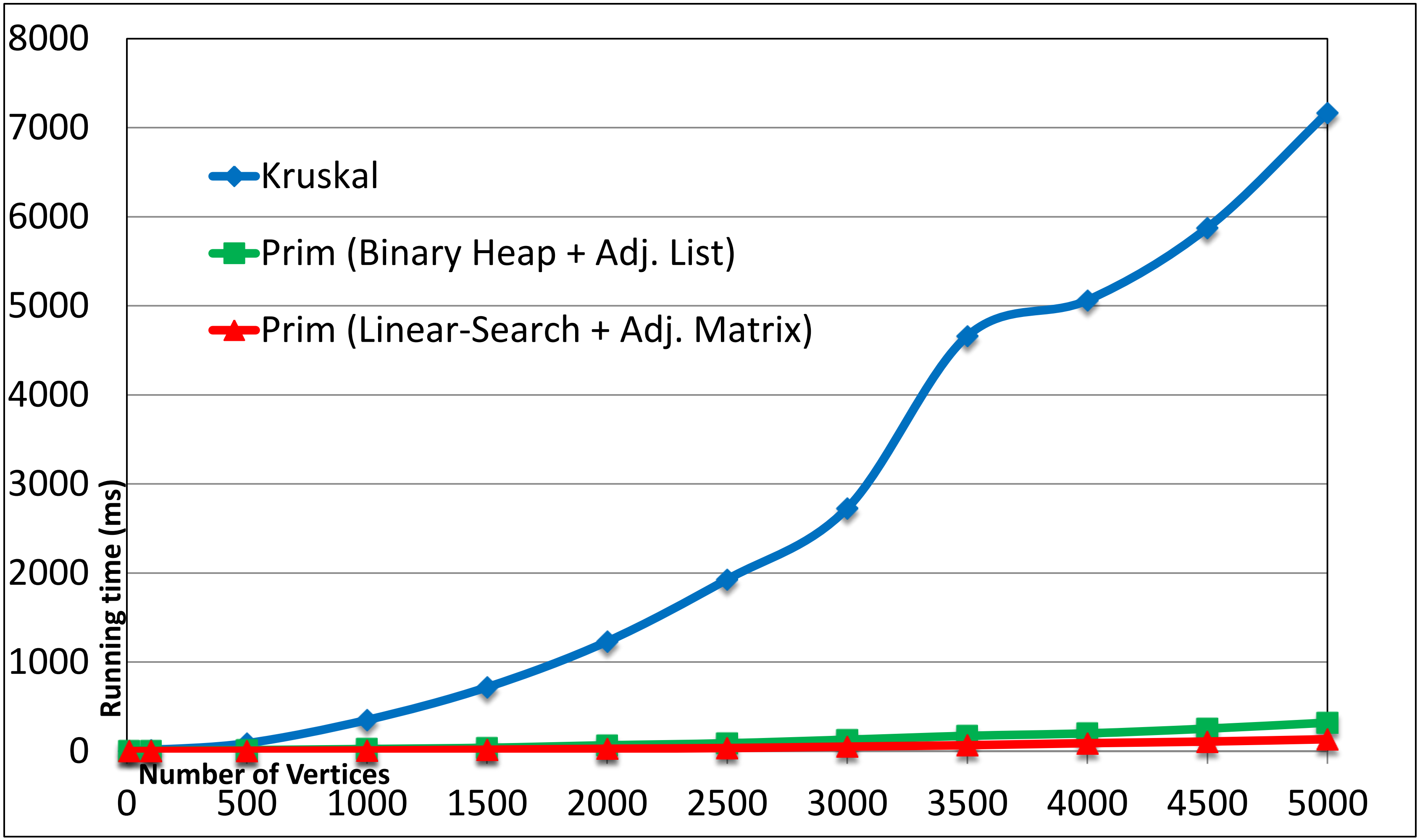
	Relationship between E and V	Kruskal	Prim (Binary-heap + Adj. List)	Prim (Linear-search + Adj. Matrix)
In General	$\lvert E\rvert \geq \lvert V\rvert$-1	O(E lgE)	O(E lgV)	$O(V^2)$
Connected **Sparse** graph, **Cycle** graph	$\lvert E\rvert \cong \lvert V\rvert$	$\mathbf{O(V\ lgV)}$	$\mathbf{O(V\ lgV)}$	$O(V^2)$
Dense graph, **Complete** graph	$\lvert E\rvert \cong \lvert V\rvert^2$	$O(V^2\ lgV)$	$O(V^2\ lgV)$	$\mathbf{O(V^2)}$

Sparse Graph



Machine Configuration: Intel Core-i5-5200U CPU @ 2.20GHz; 8.00 GB RAM

Dense Graph



Machine Configuration: Intel Core-i5-5200U CPU @ 2.20GHz; 8.00 GB RAM

Further Improvements of Prim's algorithm

- Use **Fibonacci heap** to implement min-priority-queue, Q
  - DECREASE-KEY operation: O(1) amortized time
- Running time of Prim's algorithm improves to **O(E + V lgV)**
  - Sparse graph: **O(V lgV)**
  - Dense graph: **O($\mathbf{V^2}$)**

THANK YOU