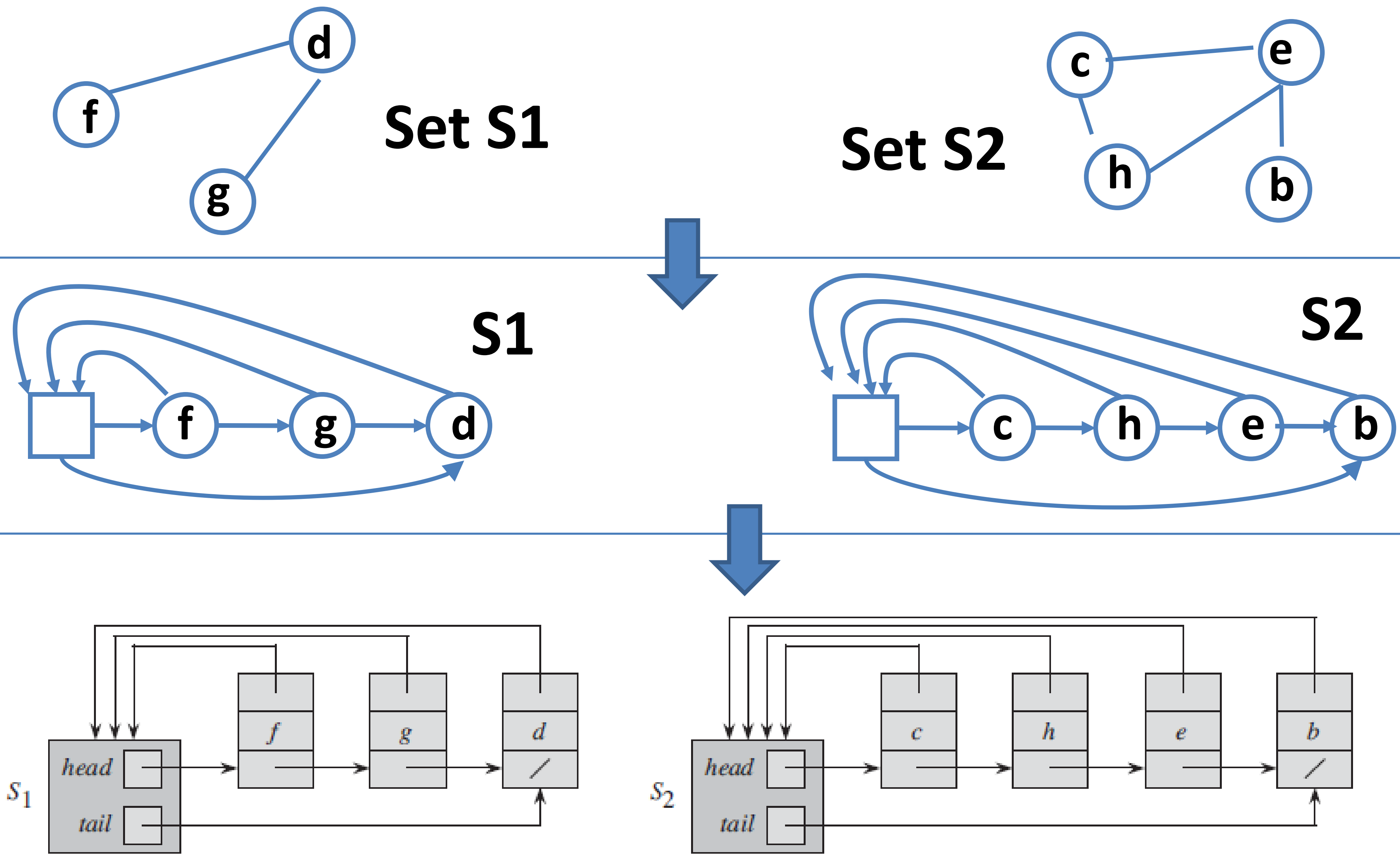
Lecture 9

Disjoint Set Data Structures

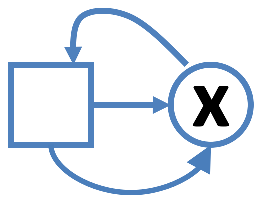
Properties of Disjoint Set Data Structures

- Suitable for data in different sets that are disjoint, for example, connected components of a graph
- Each set represented by one linked list
- Elements of a set are in a any order in the list
- Each node points **to next node** and also **to the head**
- There is a **pointer from head to tail**
- **Main Operations:**
  - MakeSet (x): makes a new list with only one node with x
  - FindSet (x): gives the head of the list containing x
  - Union (x, y): merge two lists containing x and y. Actually, Union(x,y) = Union(FindSet(x),FindSet(y)).

Example

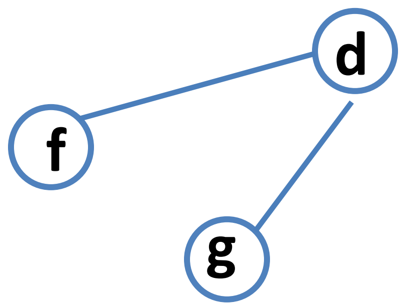


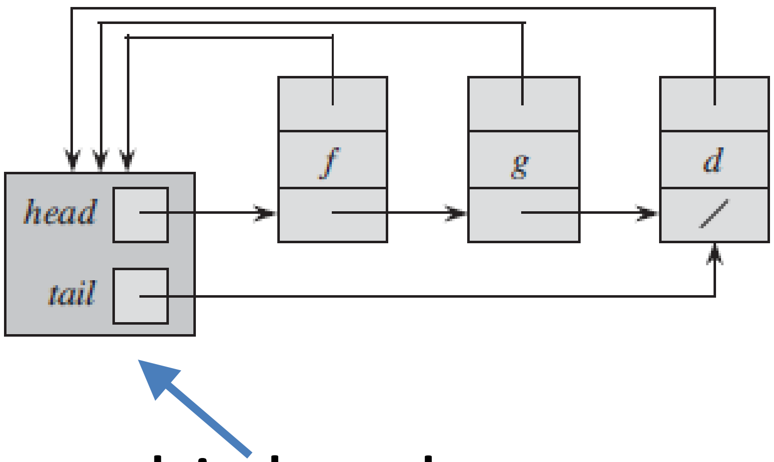
MakeSet (x)



Cost: O(1)

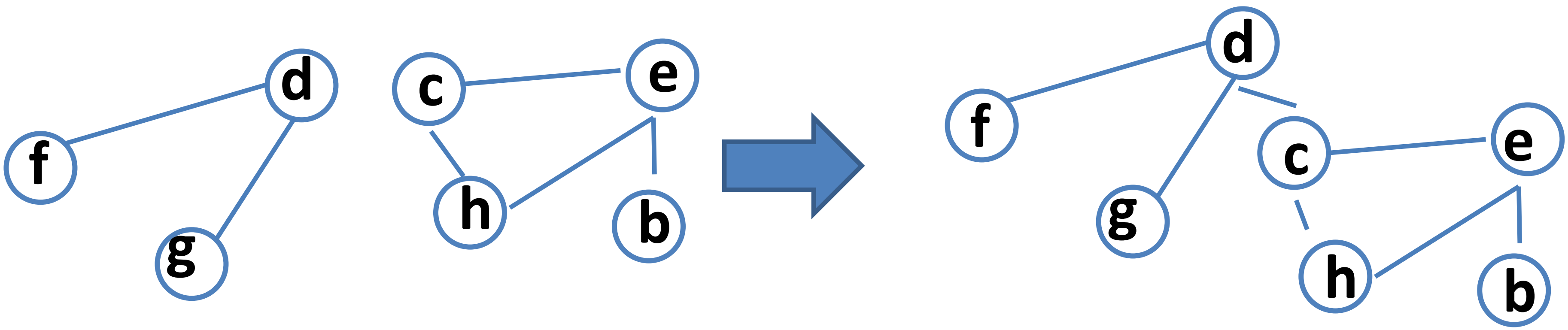
FindSet (s)

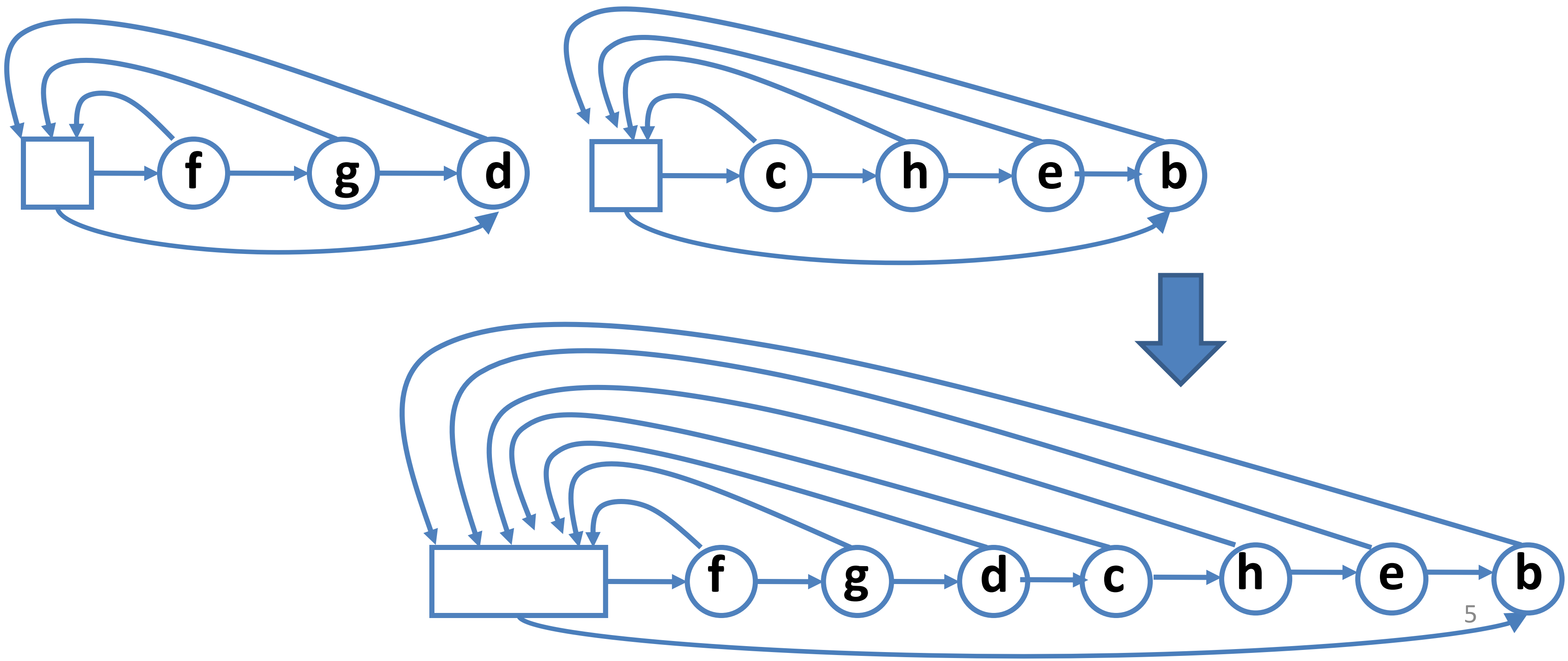




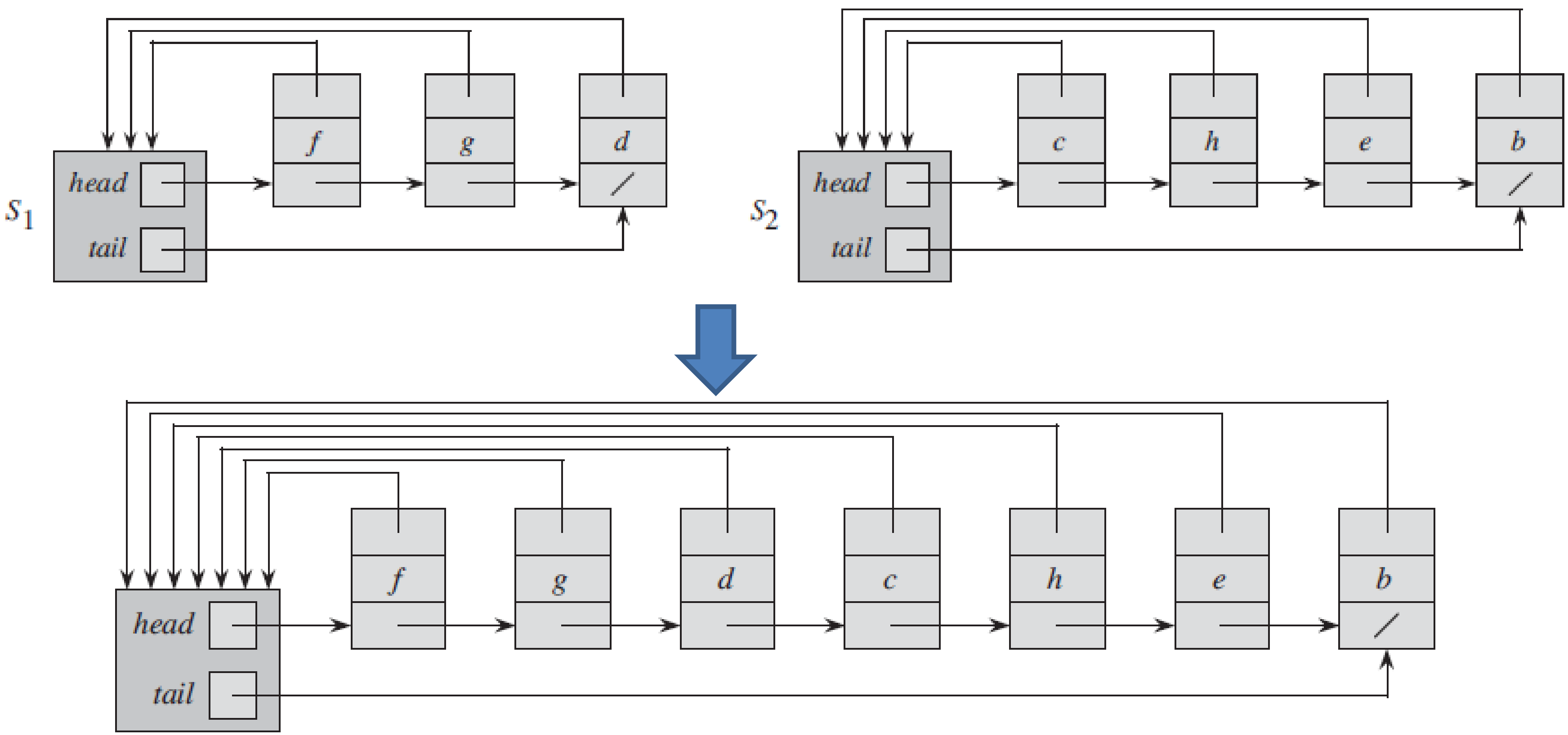
- Returns a pointer to this head
- Cost: O(1)

Example: Union (d, b)





Example: Union (d, b)



- Cost in this example: >= 4, because we changed the pointer of c, h, e, b and also the tail pointer
- In general, cost is: size of the list merged

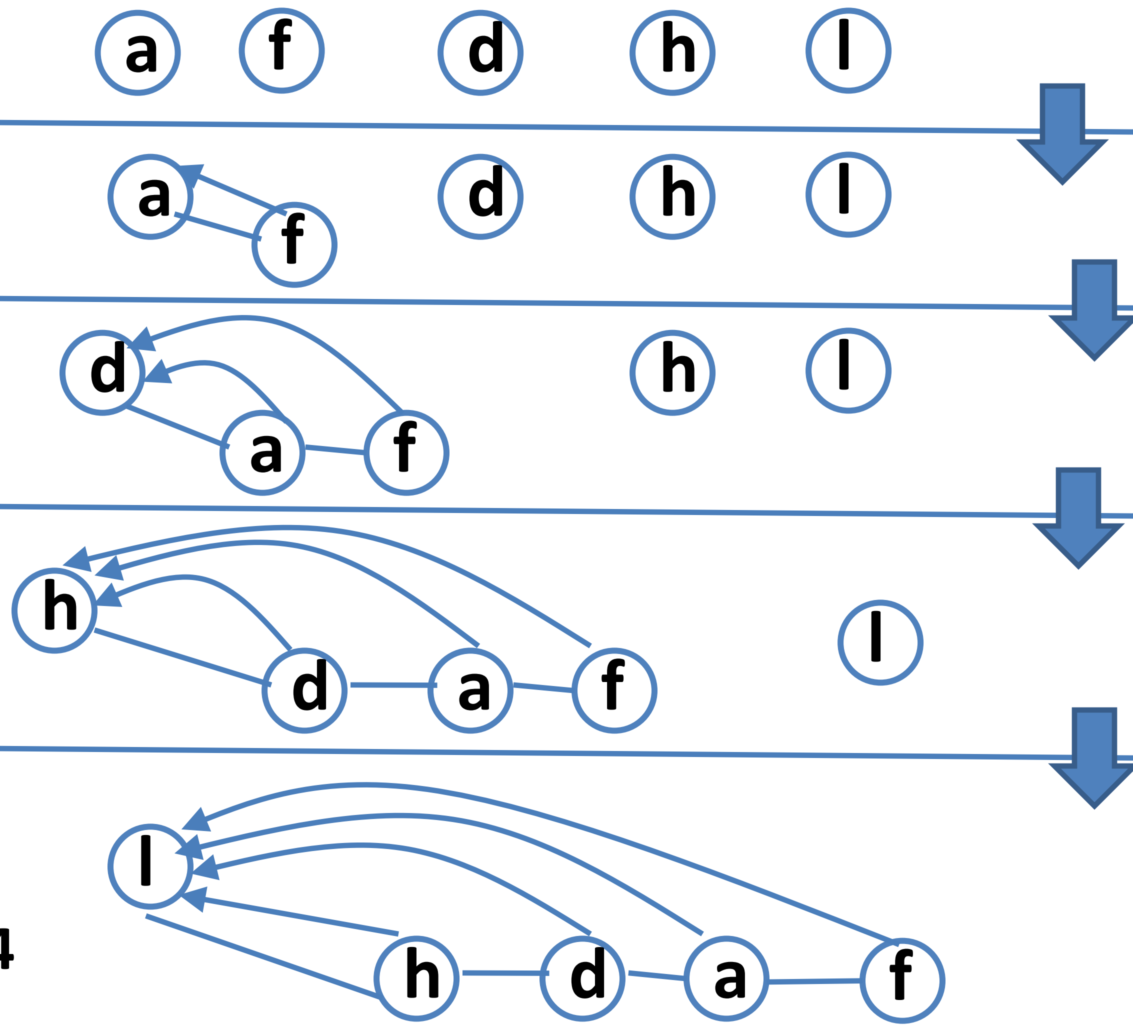
Disjoint Set Union of n elements: Worst Case Example

Union (a ,f)
Cost: 1

Union (d, f)
Cost: 2

Union (h, d)
Cost: 3

Union (i, d)
Cost: 4
Total Cost: 1+2+3+4



Disjoint Set Union of n elements: Worst Case Analysis

- Following example shows $O(n^2)$ time for Union operations on n elements
- Suppose that the size of S_1, S_2, S_3, … S_n are 1.
- Then the following n union operations take time:

Union(S_2, S_1)	**cost:** 1, because merge 1 elements of S1 with S2
Union(S_3, S_2)	**cost:** 2, because merge 2 elements of S2 with S3
Union(S_4, S_3)	**cost:** 3, because merge 3 elements of S3 with S4
…..	
Union(S_n, S_{n-1})	**cost:** n-1, because merge n-1 elements of S_{n-1} with S_n

Total cost: $1+2+3+…+(n-1) = n(n-1)/2 = O(n^2)$

Disjoint Set Union of n Elements: Improved Technique

- Following strategy gives O(n log n) time for Unions
- **Strategy:** When merging two sets, always merge the smaller set to the bigger set.
- **Running time:**
  - For one element x, its pointer is updated O(log n) time. Why?
    - Because, after 1 merge: the merged set size >= 1+1 = 2 = 2^1
    - Next time, if x's pointer is updated, then it is in the smaller set because of the merging strategy. So, second time the resulting set size is >= 2+2 = 4 = 2^2.
    - Similarly, for third time, when x's pointer is updated, the resulting set size is >= 4+4 = 8 = 2^3.
    - In this way, after at most log n time, the resulting set will be $2^{\log n}$ = n, and the elements finished, so no more merging possible.
  - For one element x, the link update is at most O(log n) time. For all n elements, it is O(n log n). See the video for an example.

Disjoint Set Union : Improved Technique Example

At the beginning:

How many link update for f?

Union (a ,f)

Link update for f = 1

Size of merged set >= 2 = 2^1

Union (h, f)

Link update for f: 1

Size of merged set >= 4 = 2^2

Union (d, f)

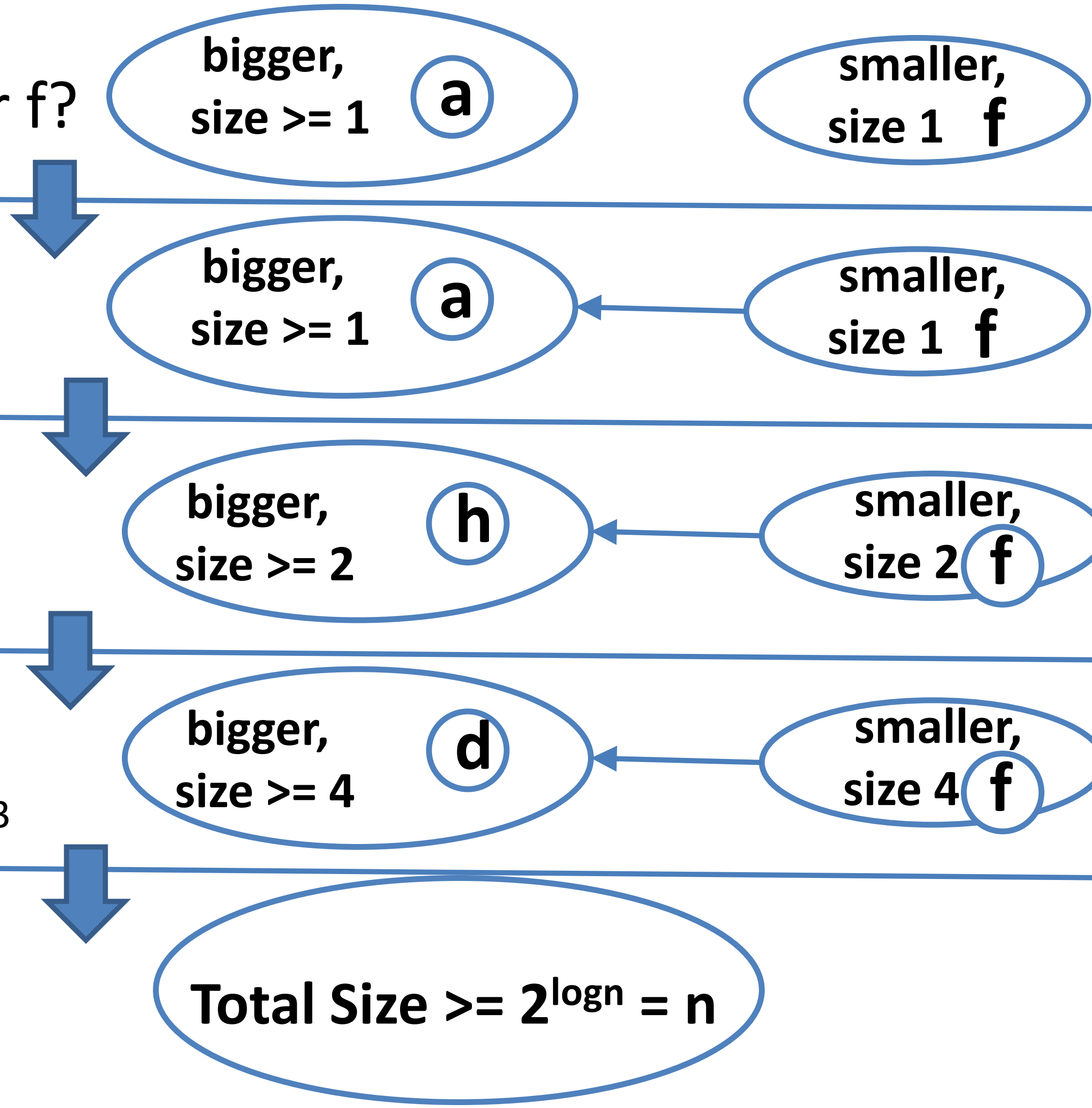
Link update for f: 1

Size of merged set >= 8 = 2^3

After log n steps ....

Union (..., f)

Link update for f: 1



Total link update for f: 1+1+1... log n times = O(log n)

Total link updates for n elements: n*O(log n) = O(n log n)

Summary of Disjoint Set Data Structures

- Suitable for data in different sets that are disjoint, for example, connected components of a graph
- **Cost of Major Operations:**
  - MakeSet (x): O(1)
  - FindSet (x): O(1)
  - Union for n elements: O(n log n) by improved strategy