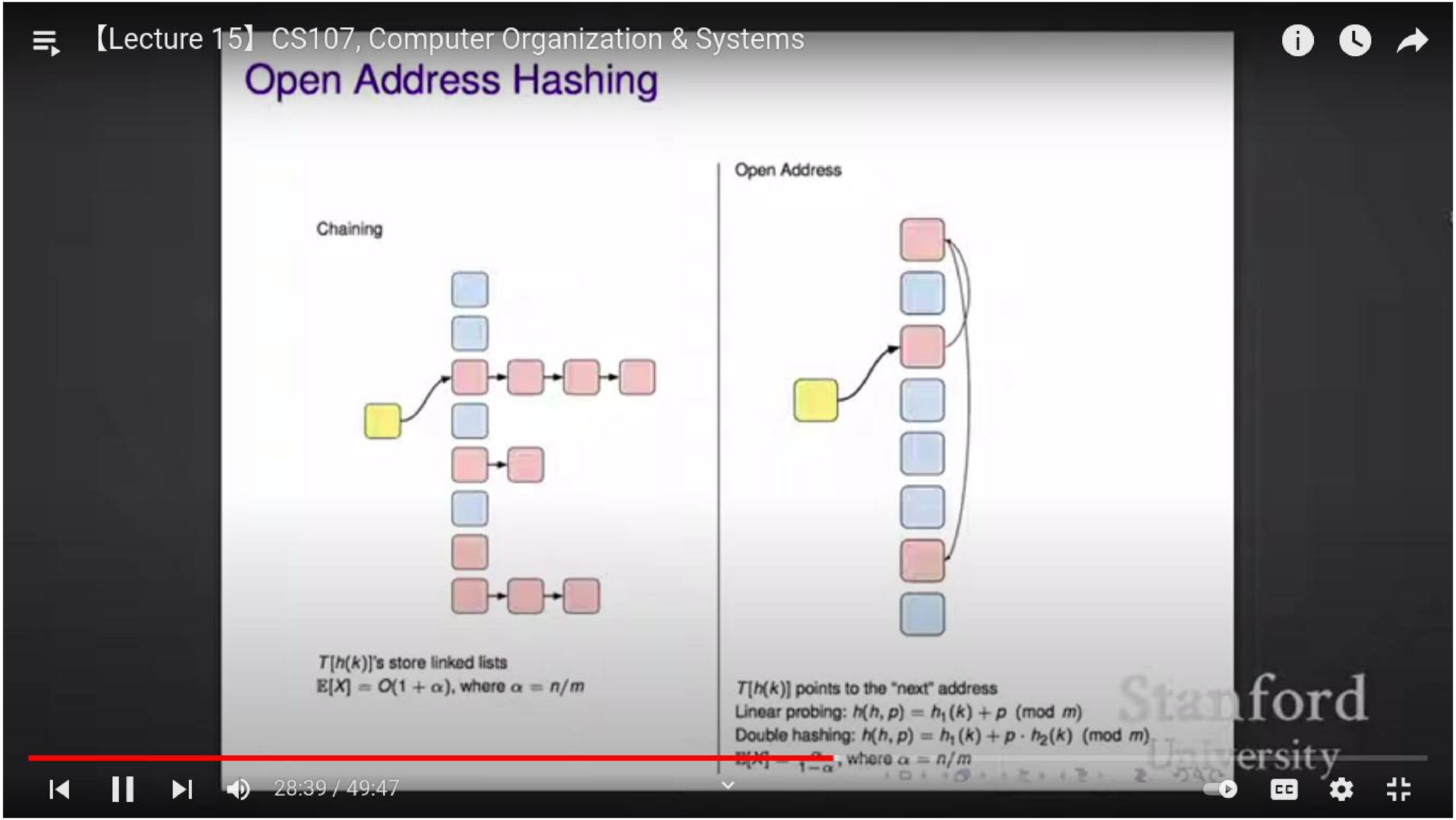

【Lecture 15】CS107, Computer Organization & Systems
Open Address Hashing
Chaining
Open Address
$T[h(k)]$'s store linked lists
$\mathbb{E}[X] = O(1+\alpha)$, where $\alpha = n/m$
$T[h(k)]$ points to the "next" address
Linear probing: $h(h, p) = h_1(k) + p \pmod{m}$
Double hashing: $h(h, p) = h_1(k) + p \cdot h_2(k) \pmod{m}$
where $\alpha = n/m$
Stanford University
28:39 / 49:47

Hash Table

Storing elements by their keys in $U = \{0, 1, ..., m-1\}$.

- Element e hashes to a key $h(e) \in U$
  - Stored at $T[h(e)]$
- Elements $e \neq e'$ but $h(e) = h(e')$
  - Collision resolved by *chaining*
  - $T[i]$'s are linked lists

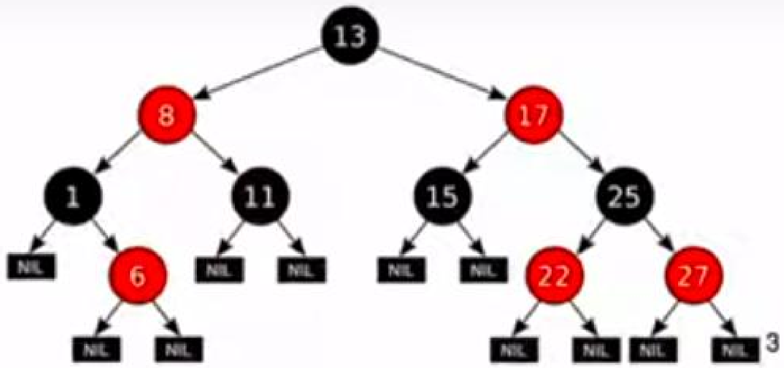
Runtime analysis:

- Worst case: $\Theta(n)$
  - All elements hash to the same key
- Uniformly random hash functions
  - Probability of hashing into each key is $1/m$
  - Expected time $O(1 + n/m)$, where n is the number of elements

Advice:

- Improve your skills on probabilities

Red-Black Tree



- Balance the tree by coloring its nodes. $h = O(\log n)$
- Coloring rules:
  - Leaves are NILs (not the *real* leaves)
  - Node is black, NILs are black
  - ***All*** children of red nodes are black
  - Number of black nodes on all root-leaft paths are equal
- Operations:
  - ***A mess.*** Good candidates for your cheatsheet.

[3] Credit: Wikipedia

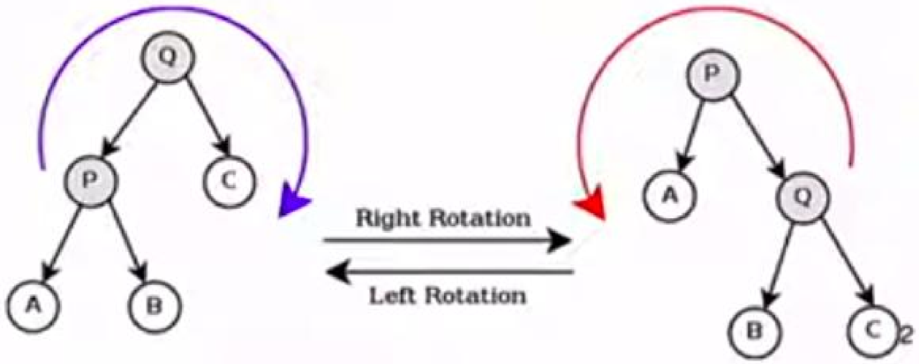
Binary Search Tree in 1 slide

Rules: each node p has left, right sub-trees, both or none

- Things in the $p.left$ of p are *strictly* less than $p.val$
- Things in the $p.right$ of p are *strictly* greater than $p.val$

Operation	How?	Complexity
Search	compare and go	$O(h)$
Insert	search and decide left or right child	$O(h)$
Delete	search and remove; check for children	$O(h)$

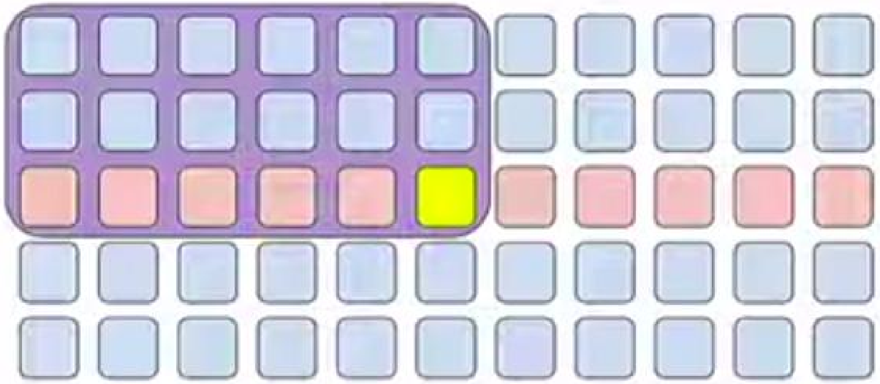
Table: Basic operations on Binary Search Trees



[2]Credit: Wikipedia

Order Statistics and Selection Algorithm

Given $a_1, a_2, ..., a_n$. Want the k^{th} smallest element



- Idea: find the *median of medians*
- Don't worry if n is not divisible by 5
  - You can always add very large numbers to a
- $T(n) = T(n/5+1) + T(7n/10+5) + c \cdot n \Longrightarrow T(n) = O(n)$

【Lecture 15】CS107, Computer Organization & Systems

Decision Tree and Sorting Lower Bound of Sorting

Decision Tree:

- Represents an algorithm
- Starts at root, follows rules and ends at leaves
- Runtime = length of the root-leaf path taken
- Shameless self-reference: Piazza @246

Lower bound of comparison-based sorting algorithms:

- Sorting algorithms that don't assume anything about the values they sort
- Runtime $\Omega(n \log n)$
  - Proof: Based on Decision Tree, refer to Lecture 6

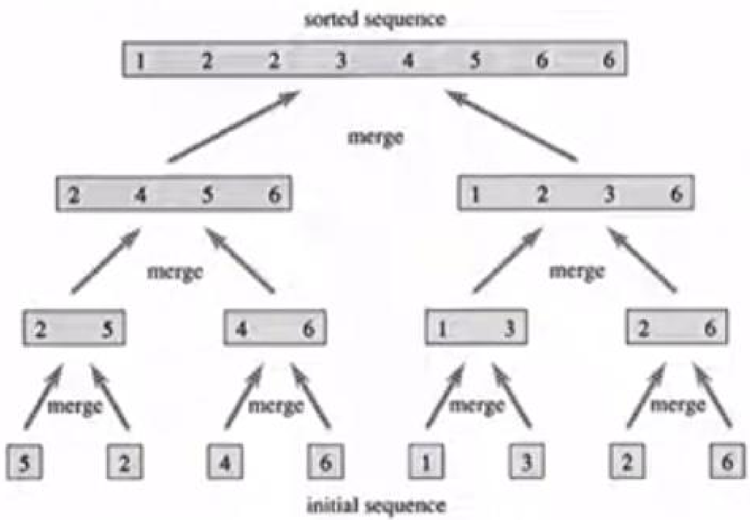
17:05 / 49:47

Quick Sort

- Idea: Divide and Conquer
- To sort a contiguous interval $a_{i \ldots j}$:
  - Find a random pivot $p = a_k$ for any $i \leq k \leq j$
  - Break $a_{i \ldots j}$ into two halves: $a_{<p}$ and $a_{>p}$
  - Sort each of them recursively
- Worst case complexity $O(n^2)$
  - Always pick the smallest element
  - Almost never happens with uniform random pivot selection
- Expected runtime $O(n \log n)$
  - Proof: analyzing the comparisons between randomly selected pivots and other elements

Merge Sort

- Idea: Divide and Conquer
- To sort a contiguous interval $a_{i...j}$:
  - Break it into two halves: $a_{i...m}$ and $a_{m+1...j}$
  - Sort each of them recursively
  - Merge them



[1]

- $T(n) = 2 \cdot T(n/2) + O(n) \Longrightarrow T(n) = O(n \log n)$

[1] Credit: http://www.ccodechamp.com/

Divide and Conquer

Idea:

- Break task T into T_1, T_2, ..., T_k
- Solve each of them
- Merger their results

Examples: Merge Sort, Quick Sort, etc.

Misc

- Don't care about $\lfloor\cdot\rfloor$, $\lceil\cdot\rceil$
- Don't care if the numbers are less than 1
- Make appropriate replacements
  - $T(n) = T(n/3) + T(n/4) + O(1)$
    $\longrightarrow T(n) = T(n/3) + T(n/4) + c$
  - $T(n) = T(n/5) + T(n/7) + O(n^5)$
    $\longrightarrow T(n) = T(n/5) + T(n/7) + c \cdot n^5$

【Lecture 15】CS107, Computer Organization & Systems

Recursion Tree

Given a recurrence: $T(n) = T(n_1) + T(n_2) + \cdots + T(n_k) + f(n)$

- Idea: count the work at each level / node / etc. in the resulting recursion tree
- Only useful for manageable values of k
  - Each node the recursion tree doesn't have too many children

9:38 / 49:47

Examples

Solve the following recurrences

- $T(n) = 5 \cdot T(n-1) + 1$
- $T(n) = T(n/2) + T(n/4) + T(n/8) + n^2$
- $T(n) = 5 \cdot T(n/10) + \log \log n$
- Fibonacci recurrence: $T(n) = T(n-1) + T(n-2)$

Strategies:

- See if you can use Master theorem. If so, how?
- Need to guess? Look at $T(n)$ for small values of n.

Want more practice?

- http://jeffe.cs.illinois.edu/teaching/algorithms/notes/99-recurrences.pdf

Substitution Method

Given a recurrence: $T(n) = T(n_1) + T(n_2) + \cdots + T(n_k) + f(n)$

Important: $n_1, n_2, ..., n_k < n$

- ***Guess*** an upper bound $g(n)$
  - Might not be tight. You will get partial credits...
- ***Prove*** by induction that $T(n) = O(g(n))$
  - Want to find n_0, c such that $\forall n \geq n_0$, $T(n) \leq c \cdot g(n)$
  - Base case: $T(n_0) \leq c \cdot g(n_0)$
    - No need to care about this. Just find n_0 later
  - Induction step: Assume $T(n') \leq c \cdot g(n')$ for all $n' < n$
    - Use the recurrence

$$
\begin{aligned}
T(n) &= T(n_1) + T(n_2) + \cdots + T(n_k) + f(n) \\
&\leq c \cdot (g(n_1) + g(n_2) + \cdots + g(n_k)) + f(n)
\end{aligned}
$$

    - Choose c such that

$$(g(n_1) + g(n_2) + \cdots + g(n_k)) + f(n) \leq c \cdot g(n)$$

Master Theorem

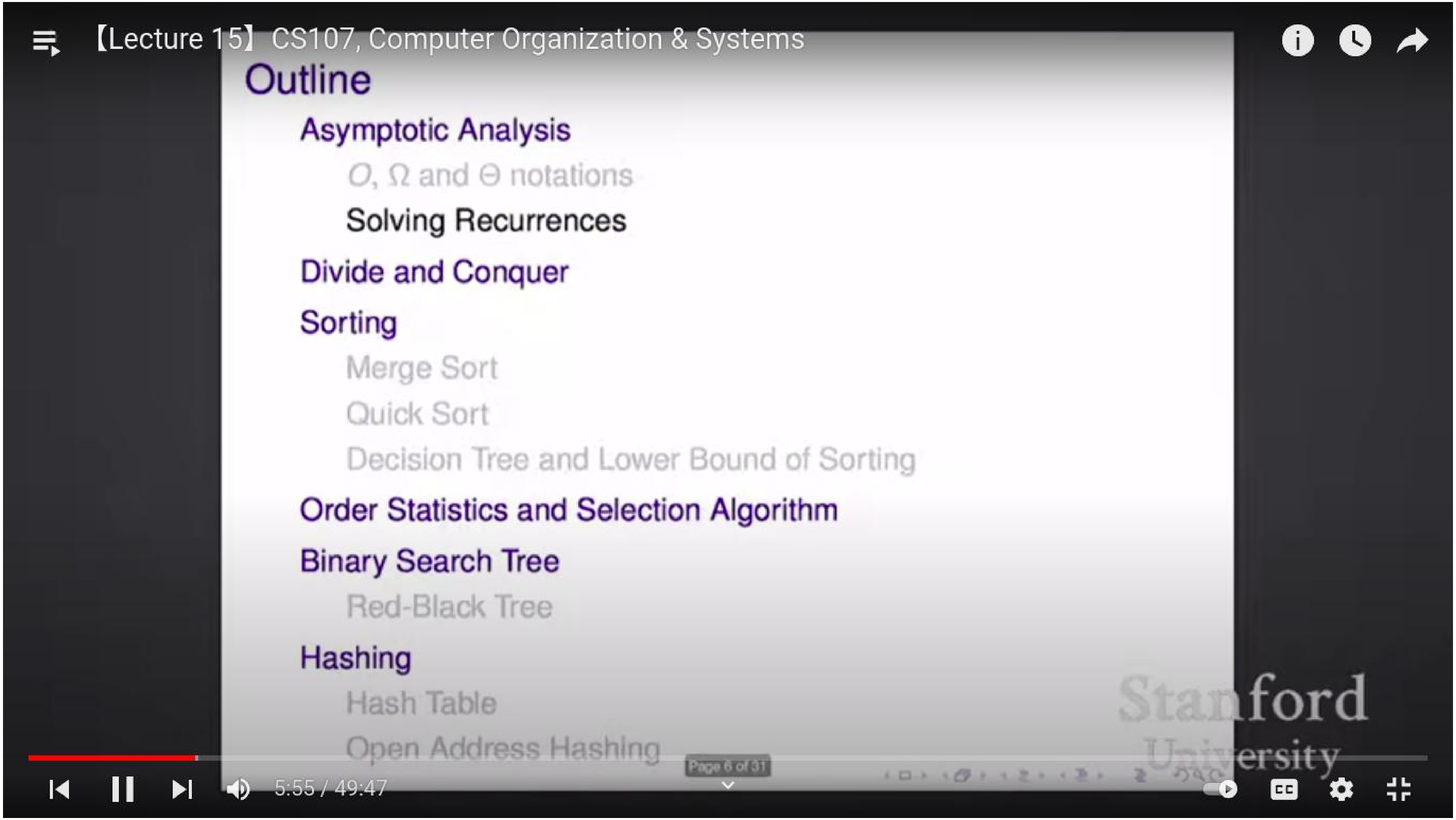
Theorem (Master Theorem)

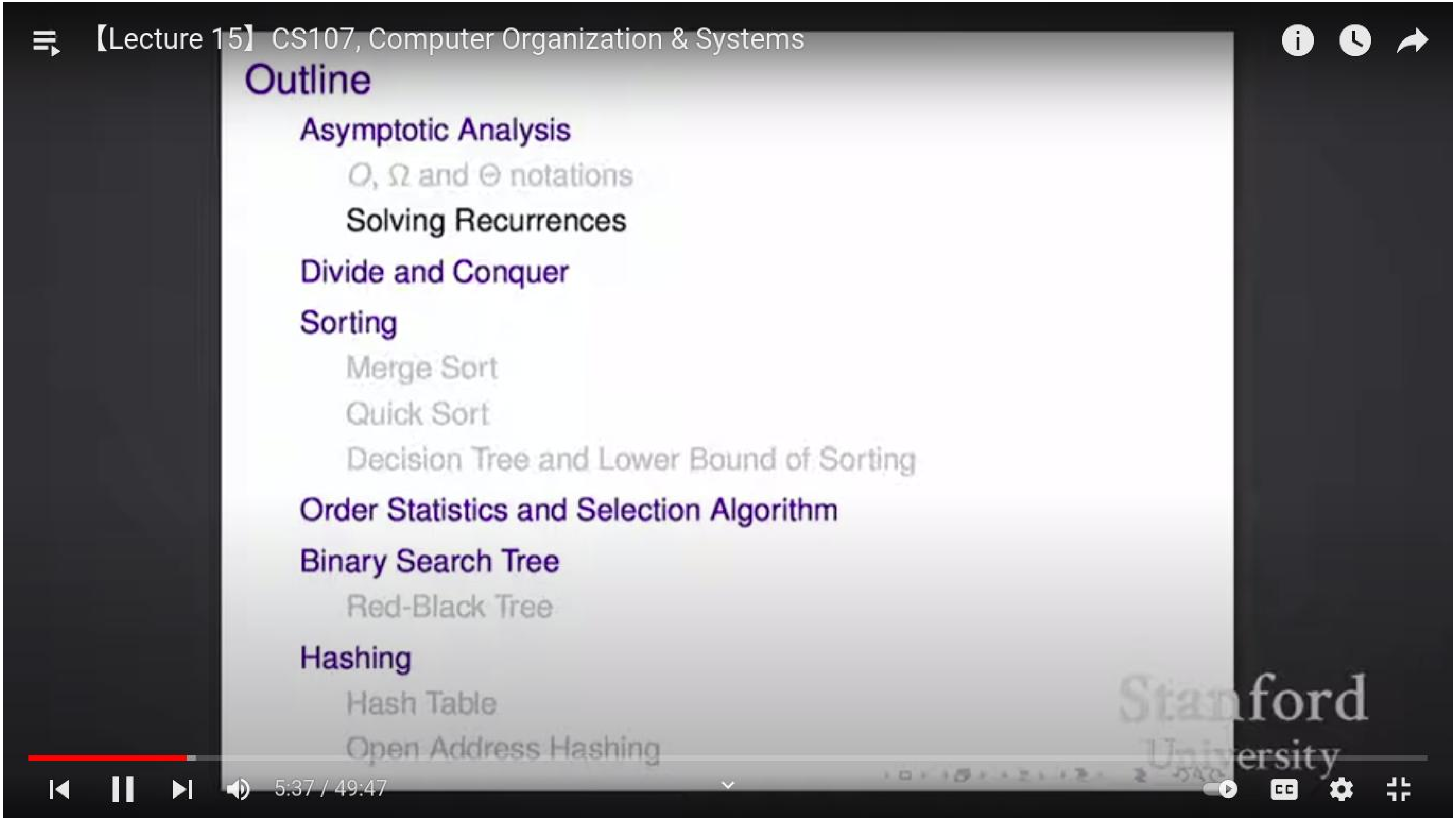
Let $T(n) = a \cdot T(n/b) + f(n)$*, where* $a \geq 1$, $b > 1$. *Then*

Conditions	$T(n)$
$f(n) = O(n^{\log_b a-\varepsilon})$	$\Theta(n^{\log_b a})$
$f(n) = \Theta(n^{\log_b a})$	$\Theta(f(n) \log n)$
$f(n) = \Omega(n^{\log_b a+\varepsilon})$ ***and*** $a \cdot f(n/b) < c \cdot f(n)$	$\Theta(f(n))$

Advice:

- Straightforward result follows from the recurrence
- Be careful with $\varepsilon > 0$ and $c < 1$ in Case 1 and Case 3


【Lecture 15】CS107, Computer Organization & Systems
Outline
Asymptotic Analysis
O, Ω and Θ notations
Solving Recurrences
Divide and Conquer
Sorting
Merge Sort
Quick Sort
Decision Tree and Lower Bound of Sorting
Order Statistics and Selection Algorithm
Binary Search Tree
Red-Black Tree
Hashing
Hash Table
Open Address Hashing
Stanford University
5:55 / 49:47


【Lecture 15】 CS107, Computer Organization & Systems
Outline
Asymptotic Analysis
O, Ω and Θ notations
Solving Recurrences
Divide and Conquer
Sorting
Merge Sort
Quick Sort
Decision Tree and Lower Bound of Sorting
Order Statistics and Selection Algorithm
Binary Search Tree
Red-Black Tree
Hashing
Hash Table
Open Address Hashing
Stanford University
5:37 / 49:47

Example

Analyze the following algorithm

```
1: function MatrixMul(A ∈ ℝ^{m×n}, B ∈ ℝ^{n×p})
2:     C ← Array[m, p]
3:     for i = 1 to m do
4:         for j = 1 to p do
5:             C_{i,j} ← 0
6:         end for
7:     end for
8:     for i = 1 to m do
9:         for j = 1 to p do
10:            for k = 1 to n do
11:                C_{i,j} ← C_{i,j} + A_{i,k} · B_{k,j}
12:            end for
13:        end for
14:    end for return C
15: end function
```

- Lines 3-7:

$$O(m) \cdot O(p) = O(mp)$$

- Lines 8-14:

$$O(m) \cdot O(p) \cdot O(n) = O(mnp)$$

- Overall:

$$O(mp) + O(mnp) = O(mnp)$$

O, Ω and Θ notations

Let $f, g : \mathbb{R}^+ \to \mathbb{R}^+$

- $f(n) = O(g(n)) \Leftrightarrow \exists c, n_0 : \forall n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) = \Omega(g(n)) \Leftrightarrow \exists c, n_0 : \forall n \geq n_0, f(n) \geq c \cdot g(n)$
- $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

In practice

- For algorithm analysis, we (mostly) care about $O(\cdot)$
- Some useful stuffs:
  - $O(f(n)) + O(g(n)) = O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$
    - Useful to breakdown algorithm into steps
  - $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$
    - Useful when analyzing loops

so if you have two function,

Outline

- Asymptotic Analysis
  - O, Ω and Θ notations
  - Solving Recurrences
- Divide and Conquer
- Sorting
  - Merge Sort
  - Quick Sort
  - Decision Tree and Lower Bound of Sorting
- Order Statistics and Selection Algorithm
- Binary Search Tree
  - Red-Black Tree
- Hashing
  - Hash Table
  - Open Address Hashing

I'm going to give you guys a review of all the topics in your glass sofa.

Semi-connected Graphs

Problem

A directed graph $G = (V, E)$ is ***semi-connected*** *if for all $u, v \in V$, either we can go from u to v, or we can go from v to u. Given $G = (V, E)$, determine whether it is semi-connected in $O(m + n)$.*

- First, consider only directed acyclic graphs (DAG)

【Lecture 14】 CS107, Computer Organization & Systems

Outline

- Graph Traversals
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)
- Shortest Paths
  - Dijkstra
  - Bellman-Ford
  - Floyd-Warshall
- Dynamic Programming
- Minimum Spanning Tree
  - Boruvka
  - Kruskal
  - Prim
- Global MinCut and Karger's Algorithm
- Minimum st-cut and MaxFlow
- **Problem Solving**

Stanford University

16:48 / 54:33

【Lecture 14】CS107, Computer Organization & Systems

Ford-Fulkerson method

```
procedure MaxFlow(G = (V, E), s, t ∈ V)
    f(u, v) ← 0
    G_f ← G
    while t is reachable from s in G_f do
        P ← st-path
        F ← min weight on P
        for (u, v) ∈ P do
            if (u, v) ∈ E then
                f(u, v) ← f(u, v) + F
            else
                f(v, u) ← f(v, u) − F
            end if
        end for
    end while
end procedure
```

- How to find the st-paths? DFS / BFS
- Runtime: $O((m + n) \cdot \#\text{runs})$

14:15 / 54:33

【Lecture 14】CS107, Computer Organization & Systems

Minimum *st*-cut and Max *st*-flow

	Min *st*-cut	Max *st*-flow
Input	$G = (V, E)$ and $s, t \in V$	
Output	a subset $s \in S, t \in V \setminus S$	$f(u, v), \forall (u, v) \in E$
Requires	min sum weights	$\sum_u c(u, v) = \sum_u c(v, u)$

Theorem

For any $G = (V, E)$ and $s, t \in V$,

$$MinCut_G(s, t) = MaxFlow_G(s, t)$$

12:10 / 54:33

Karger's Algorithm

procedure $\textsc{Karger}(G = (V, E), s \in V)$
    for $i = 1$ to $|V| - 2$ **do**
        $(u, v) \leftarrow$ random edge
        $\textsc{Contract}(u, v)$
    end for
end procedure

- Correct iff always pick good edges
  - $P((u, v) \text{ is good}) \geq \frac{n-2}{n}$
  - $P(correctcut) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{1}{3} = \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}$
- Runtime: $O(n^2)$
  - Each $\textsc{Contract}(u, v)$ takes $O(n)$
  - $n - 2$ contractions

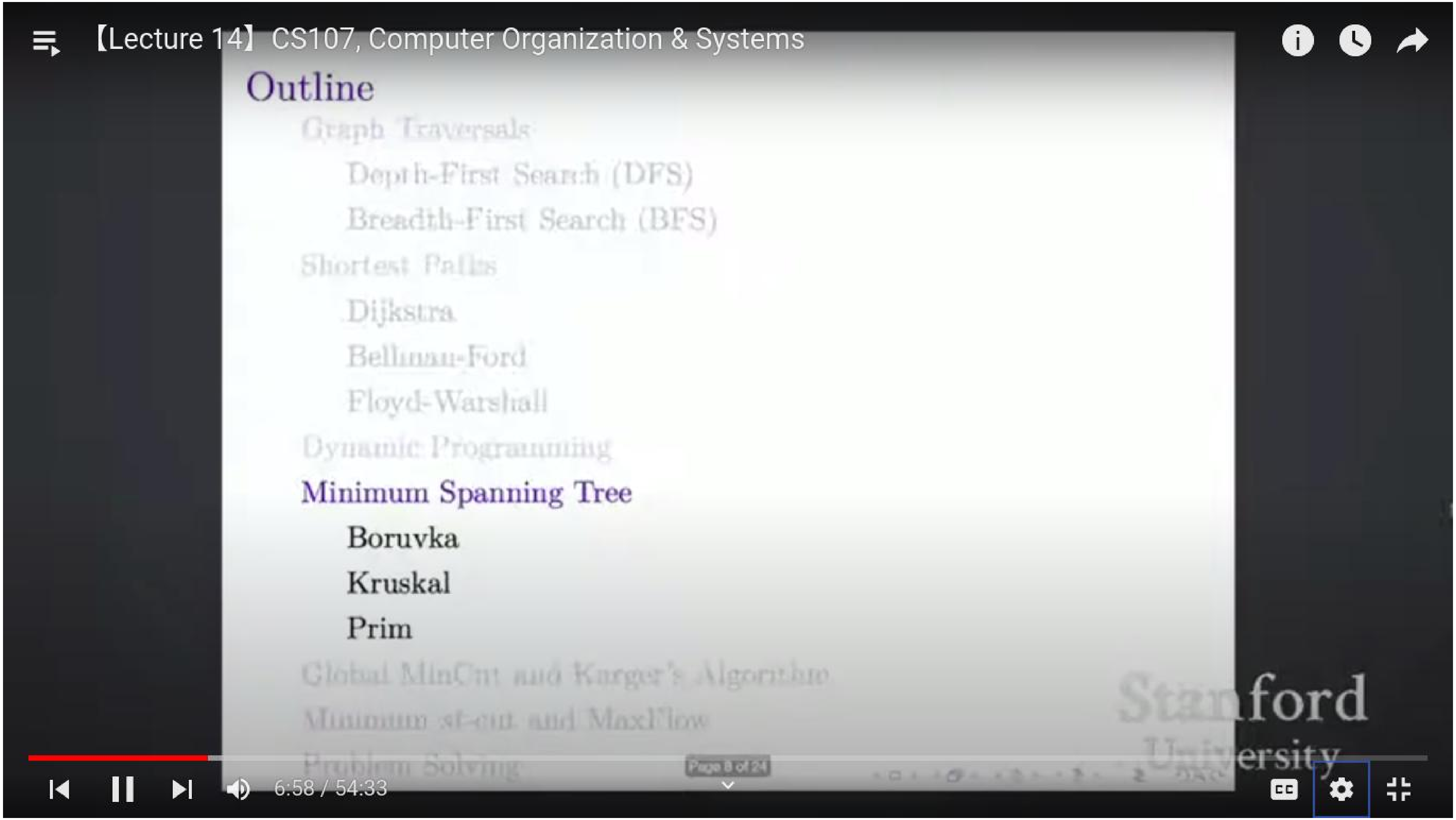
Global Minimum Cut

Problem

- *Input:* $G = (V, E)$
- *Output: a partition* $V = S \cup (V \setminus S)$ *that minimizes the number of edges between* S *and* $V \setminus S$

Minimum Spanning Tree algorithms

- Boruvka
  - Maintain a set of disjoint trees
  - Each tree picks the edge with smallest weight out of it and merge
  - $O(m \log n)$
- Kruskal
  - Sort all the edges by their weights. Choose edges of unmerged vertices
  - $O(m(\log m + f^{-1}(n)))$, where $f^{-1}(n)$ is the inverse Ackermann function
- Prim
  - Similar to Dijkstra, with $d(v) \leftarrow w(v, \pi(v))$
  - $O(m + n \log n$ with Fibonacci heap


【Lecture 14】 CS107, Computer Organization & Systems
Outline
Graph Traversals
Depth-First Search (DFS)
Breadth-First Search (BFS)
Shortest Paths
Dijkstra
Bellman-Ford
Floyd-Warshall
Dynamic Programming
Minimum Spanning Tree
Boruvka
Kruskal
Prim
Global MinCut and Karger's Algorithm
Minimum st-cut and MaxFlow
Problem Solving
Stanford
University
6:58 / 54:33

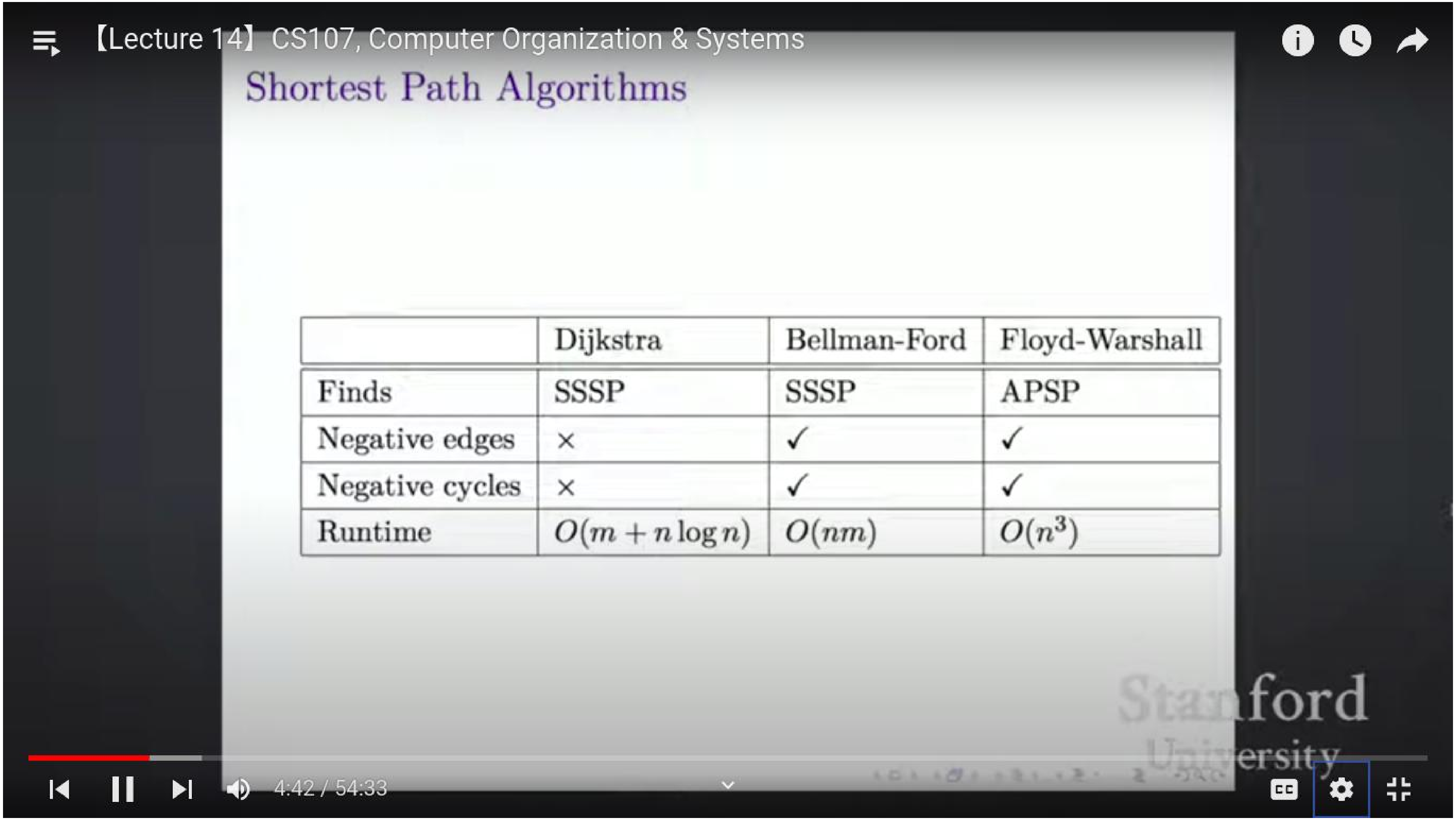
【Lecture 14】CS107, Computer Organization & Systems

Dynamic Programming

- Repetitions of sub-problems
- Figure what to compute
  - Base case
  - Recurrence
- Somewhat similar to divide and conquer

Stanford University

5:49 / 54:33



Shortest Path Algorithms

	Dijkstra	Bellman-Ford	Floyd-Warshall
Finds	SSSP	SSSP	APSP
Negative edges	$\times$	$\checkmark$	$\checkmark$
Negative cycles	$\times$	$\checkmark$	$\checkmark$
Runtime	$O(m + n \log n)$	$O(nm)$	$O(n^3)$

【Lecture 14】 CS107, Computer Organization & Systems

Outline

- Graph Traversals
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)
- **Shortest Paths**
  - Dijkstra
  - Bellman-Ford
  - Floyd-Warshall
- Dynamic Programming
- Minimum Spanning Tree
  - Boruvka
  - Kruskal
  - Prim
- Global MinCut and Karger's Algorithm
- Minimum *st*-cut and MaxFlow
- Problem Solving

Stanford University

4:34 / 54:33