

# **CSE 421**

## **Topological Ordering / Greedy**

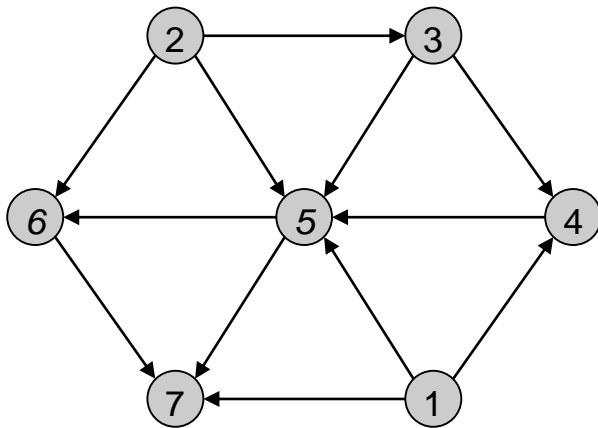
Shayan Oveis Gharan

# DAGs and Topological Ordering

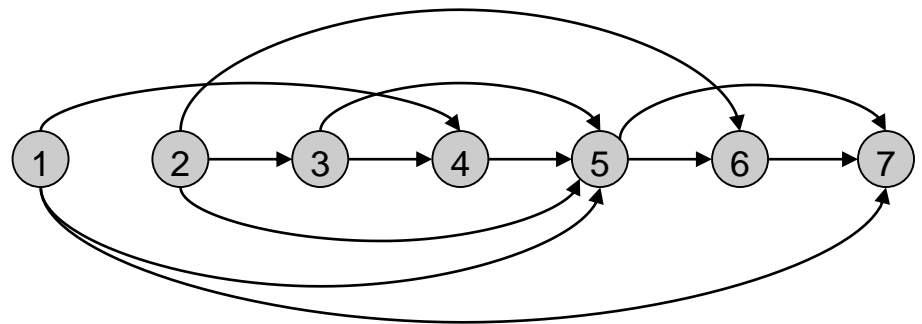
# Directed Acyclic Graphs (DAG)

A **DAG** is a directed acyclic graph, i.e., one that contains no directed cycles.

**Def:** A **topological order** of a directed graph  $G = (V, E)$  is an ordering of its nodes as  $v_1, v_2, \dots, v_n$  so that for every edge  $(v_i, v_j)$  we have  $i < j$ .



a DAG



a topological ordering of that DAG—  
all edges left-to-right

# DAGs: A Sufficient Condition

**Lemma:** If  $G$  has a topological order, then  $G$  is a DAG.

**Pf.** (by contradiction)

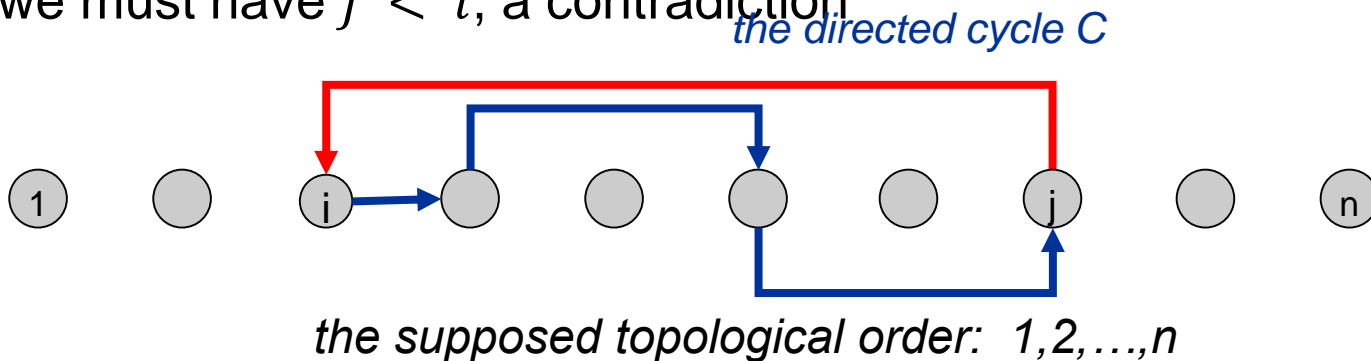
Suppose that  $G$  has a topological order  $1, 2, \dots, n$  and that  $G$  also has a directed cycle  $C$ .

Let  $i$  be the **lowest-indexed** node in  $C$ , and let  $j$  be the node just before  $i$ ; thus  $(j, i)$  is an (directed) edge.

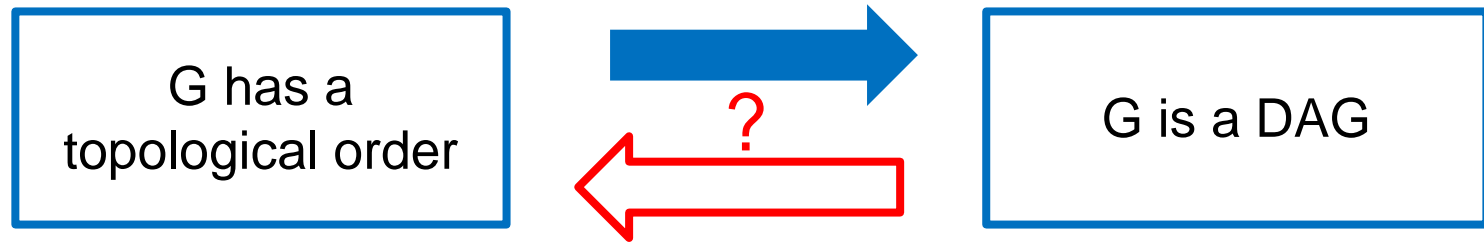
By our choice of  $i$ , we have  $i < j$ .

Have you seen this idea before?  
Yes! In analyzing Man-optimal stable matching

On the other hand, since  $(j, i)$  is an edge and  $1, \dots, n$  is a topological order, we must have  $j < i$ , a contradiction.



# DAGs: A Sufficient Condition



# Every DAG has a source node

**Lemma:** If  $G$  is a DAG, then  $G$  has a node with no incoming edges (i.e., a source).

**Pf.** (by contradiction)

Suppose that  $G$  is a DAG and it has no source

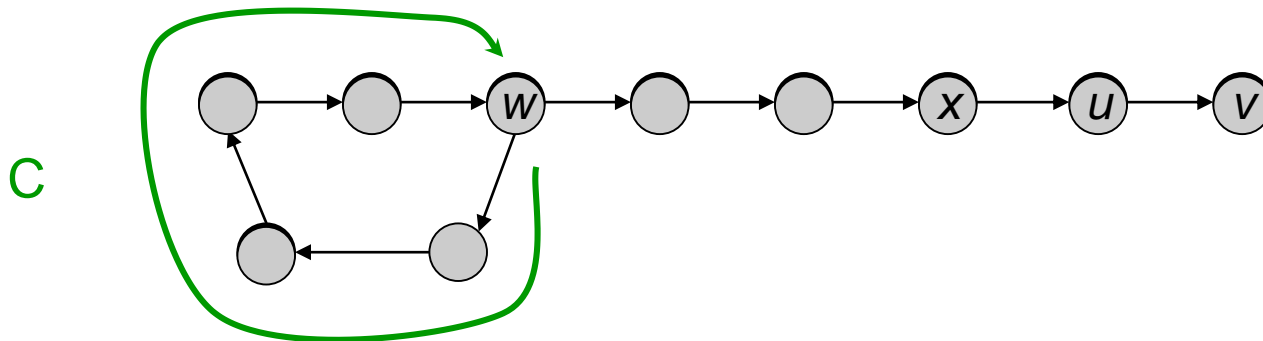
Pick any node  $v$ , and begin following edges **backward** from  $v$ . Since  $v$  has at least one incoming edge  $(u, v)$  we can walk backward to  $u$ .

Then, since  $u$  has at least one incoming edge  $(x, u)$ , we can walk backward to  $x$ .

Repeat until we visit a node, say  $w$ , twice.

Let  $C$  be the sequence of nodes encountered between successive visits to  $w$ .  $C$  is a cycle.

Is this similar to a previous proof?



# DAG $\Rightarrow$ Topological Order

**Lemma:** If  $G$  is a DAG, then  $G$  has a topological order

**Pf.** (by induction on  $n$ )

**Base case:** true if  $n = 1$ .

**IH:** Every DAG with  $n-1$  vertices has a topological ordering.

**IS:** Given DAG with  $n > 1$  nodes, find a source node  $v$ .

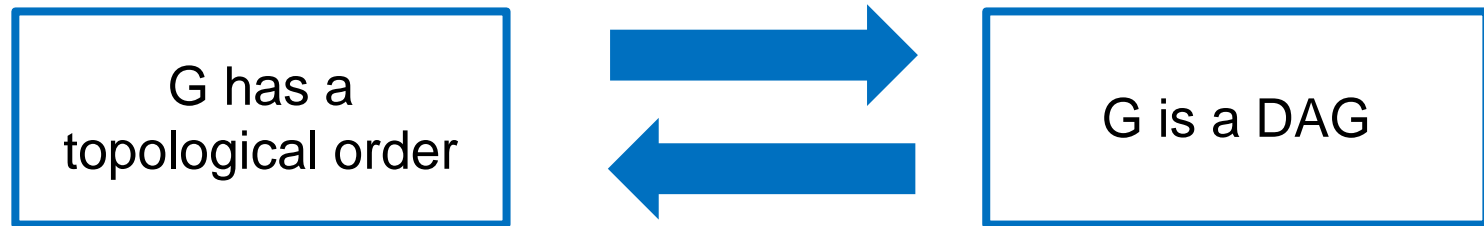
$G - \{v\}$  is a DAG, since deleting  $v$  cannot create cycles.

Reminder: Always remove vertices/edges to use IH

By IH,  $G - \{v\}$  has a topological ordering.

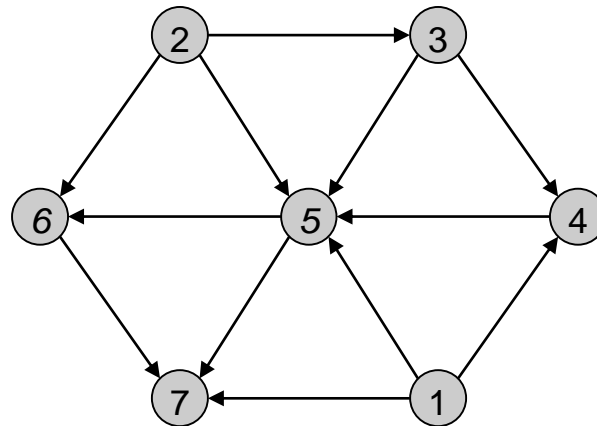
Place  $v$  first in topological ordering; then append nodes of  $G - \{v\}$  in topological order. This is valid since  $v$  has no incoming edges.

# A Characterization of DAGs

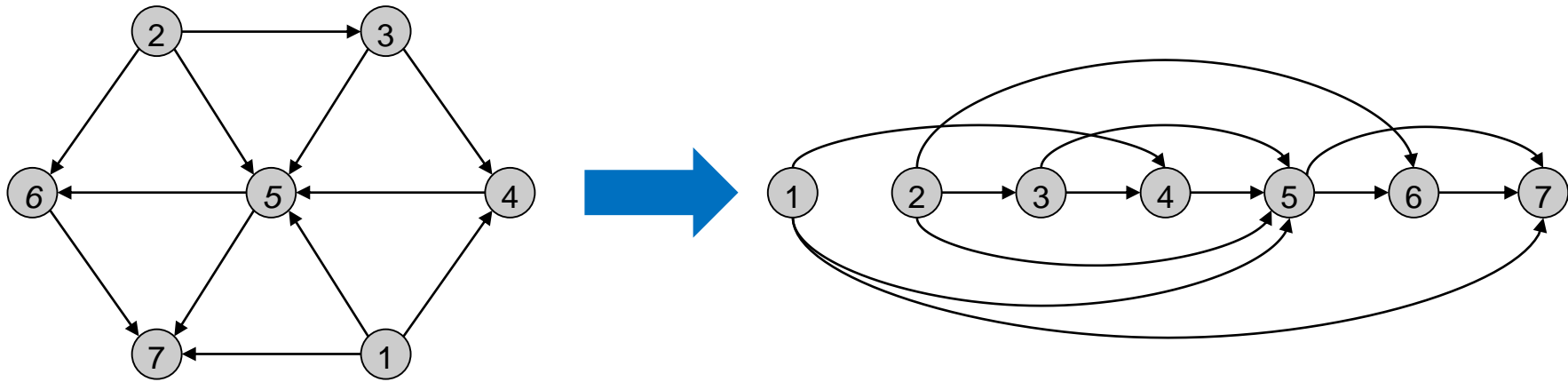




# Topological Order Algorithm: Example



# Topological Order Algorithm: Example



*Topological order: 1, 2, 3, 4, 5, 6, 7*

# Topological Sorting Algorithm

Maintain the following:

count[w] = (remaining) number of incoming edges to node w

S = set of (remaining) nodes with no incoming edges

Initialization:

count[w] = 0 for all w

count[w]++ for all edges (v,w) O(m + n)

S = S  $\cup$  {w} for all w with count[w]=0

Main loop:

while S not empty

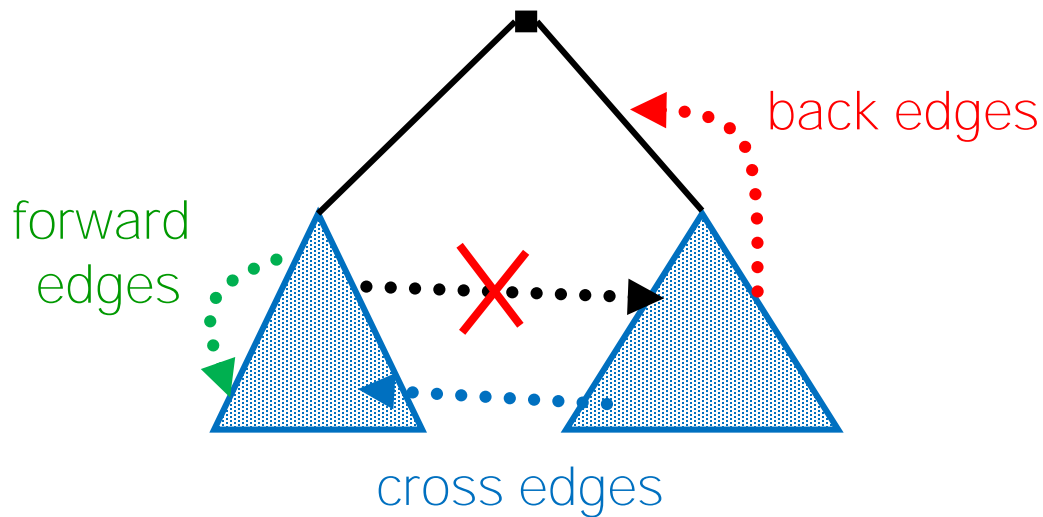
- remove some v from S
- make v next in topo order O(1) per node
- for all edges from v to some w O(1) per edge
  - decrement count[w]
  - add w to S if count[w] hits 0

Correctness: clear, I hope

Time: O(m + n) (assuming edge-list representation of graph)

# DFS on Directed Graphs

- Before DFS(s) returns, it visits all previously unvisited vertices reachable via directed paths from s
- Every cycle contains a back edge in the DFS tree



# Summary

- Graphs: abstract relationships among pairs of objects
- Terminology: node/vertex/vertices, edges, paths, multi-edges, self-loops, connected
- Representation: Adjacency list, adjacency matrix
- Nodes vs Edges:  $m = O(n^2)$ , often less
- BFS: Layers, queue, shortest paths, all edges go to same or adjacent layer
- DFS: recursion/stack; all edges ancestor/descendant
- Algorithms: Connected Comp, bipartiteness, topological sort

# Greedy Algorithms



**Coin Changing Problem  
Greedy Algorithm**

# Greedy Strategy

**Goal:** Given currency denominations: 1, 5, 10, 25, 100, give change to customer using *fewest* number of coins.

**Ex:** 34¢.



**Cashier's algorithm:** At each iteration, give the *largest* coin valued  $\leq$  the amount to be paid.

**Ex:** \$2.89.



# Greedy is not always Optimal

**Observation:** Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

**Counterexample.** 140¢.

Greedy: 100, 34, 1, 1, 1, 1, 1, 1.

Optimal: 70, 70.



**Lesson:** Greedy is short-sighted. Always chooses the most attractive choice at the moment. But this may lead to a dead-end later.



# Greedy Algorithms Outline

## Pros

- Intuitive
- Often simple to design (and to implement)
- Often fast

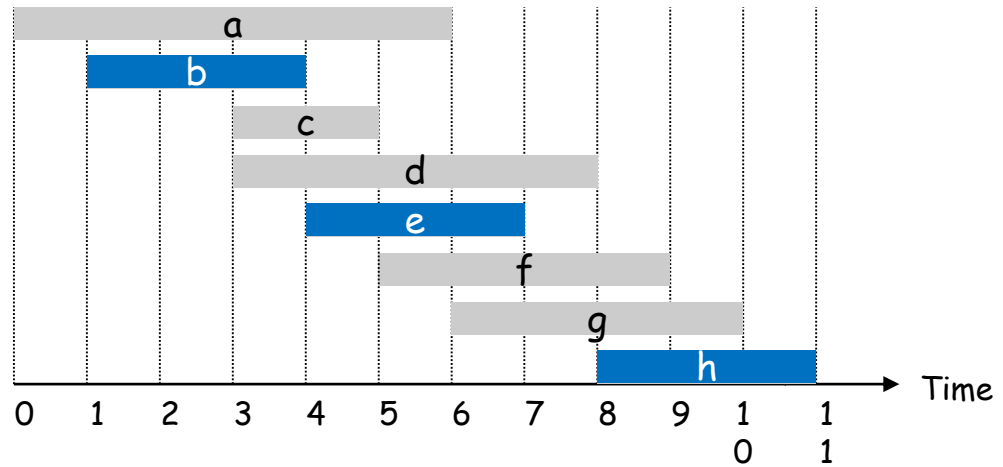
## Cons

- Often incorrect!

## Proof techniques:

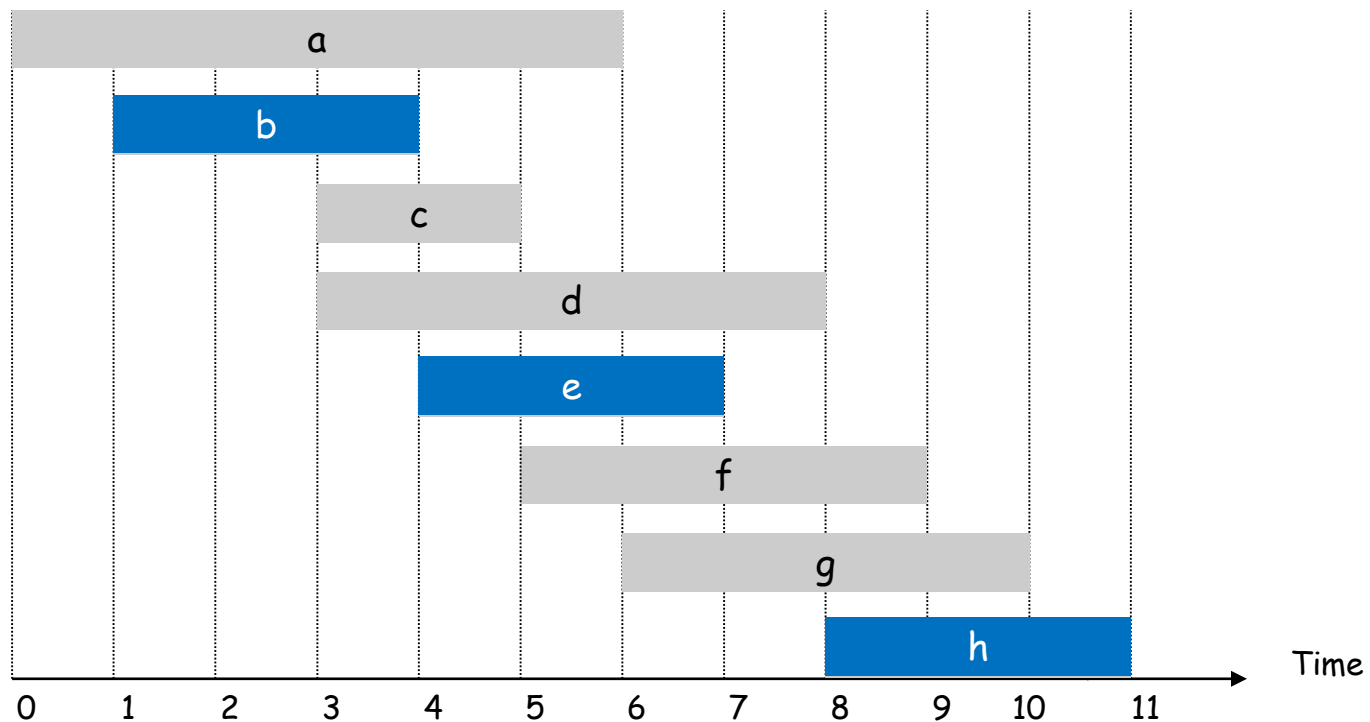
- Stay ahead
- Structural
- Exchange arguments

# Interval Scheduling



# Interval Scheduling

- Job  $j$  starts at  $s(j)$  and finishes at  $f(j)$ .
- Two jobs **compatible** if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



# Greedy Strategy

Sort the jobs in **some** order. Go over the jobs and take as much as possible provided it is compatible with the jobs already taken.

Main question:

- What order?
- Does it give the Optimum answer?
- Why?

# Possible Approaches for Inter Sched

Sort the jobs in **some** order. Go over the jobs and take as much as possible provided it is compatible with the jobs already taken.

[Earliest start time] Consider jobs in ascending order of start time  $s_j$ .

[Earliest finish time] Consider jobs in ascending order of finish time  $f_j$ .

[Shortest interval] Consider jobs in ascending order of interval length  $f_j - s_j$ .

[Fewest conflicts] For each job, count the number of conflicting jobs  $c_j$ . Schedule in ascending order of conflicts  $c_j$ .

# Greedy Alg: Earliest Finish Time

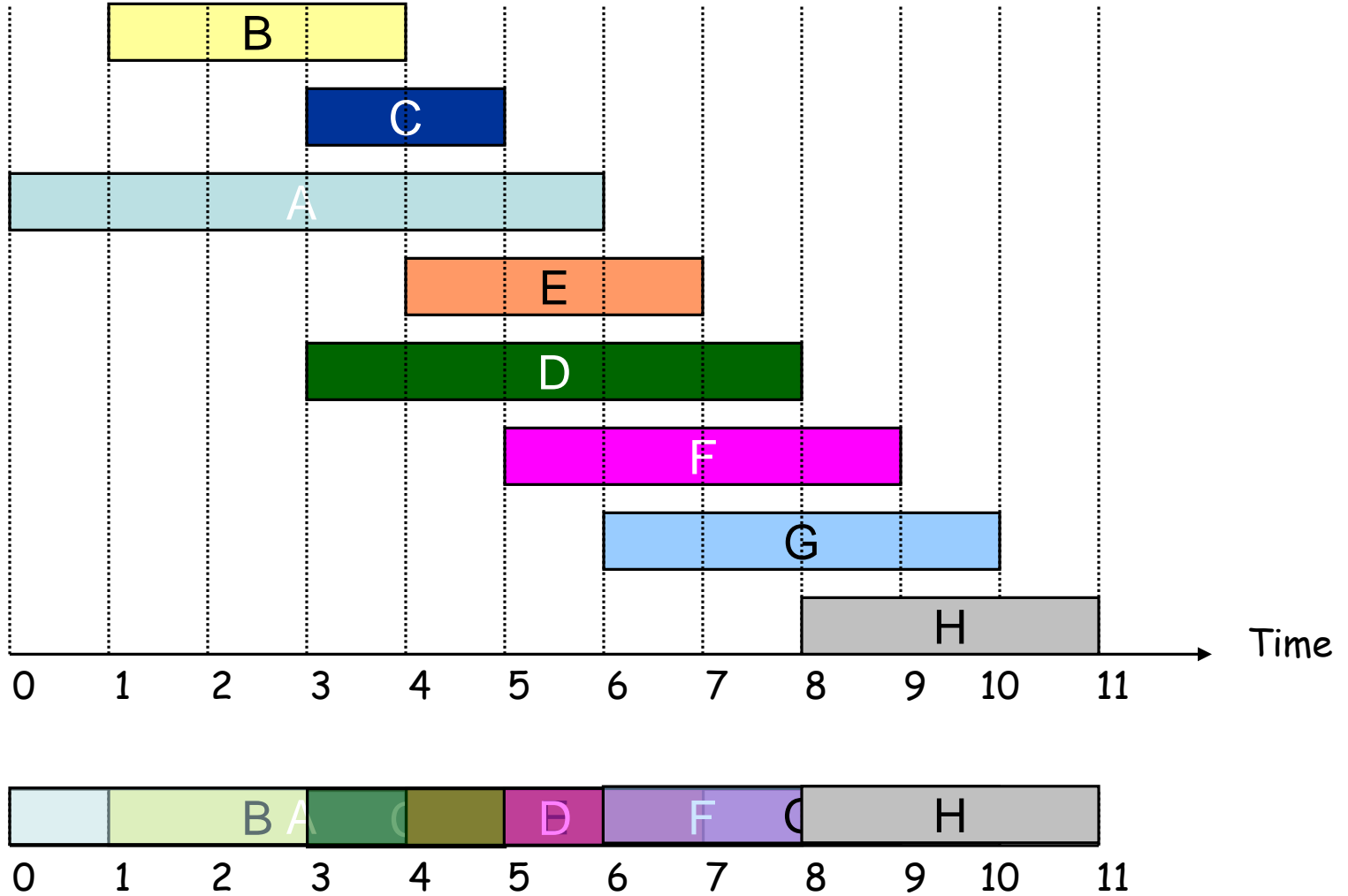
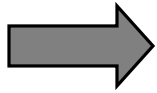
Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

```
1
Sort jobs by finish times so that  $f(1) \leq f(2) \leq \dots \leq f(n)$ .
A  $\emptyset$ 
for j = 1 to n {
    if (job j compatible with A)
        A  $A \cup \{j\}$ 
}
return A
```

**Implementation.**  $O(n \log n)$ .

- Remember job  $j^*$  that was added last to A.
- Job  $j$  is compatible with A if  $s(j) \geq f(j^*)$ .

# Greedy Alg: Example



# Correctness

**Theorem:** Greedy algorithm is optimal.

**Pf:** (technique: “Greedy stays ahead”)

Let  $i_1, i_2, \dots, i_k$  be jobs picked by greedy,  $j_1, j_2, \dots, j_m$  those in some optimal solution in order.

We show  $f(i_r) \leq f(j_r)$  for all  $r$ , by induction on  $r$ .

**Base Case:**  $i_1$  chosen to have min finish time, so  $f(i_1) \leq f(j_1)$ .

**IH:**  $f(i_r) \leq f(j_r)$  for some  $r$

**IS:** Since  $f(i_r) \leq f(j_r) \leq s(j_{r+1})$ ,  $j_{r+1}$  is among the candidates considered by greedy when it picked  $i_{r+1}$ , & it picks min finish, so  $f(i_{r+1}) \leq f(j_{r+1})$

Observe that we must have  $k \geq m$ , else  $j_{k+1}$  is among (nonempty) set of candidates for  $i_{k+1}$