CS340 - Analysis of Algorithms

Greedy Algorithms

Logistics:

Upcoming deadlines:

- Lab1 due Tomorrow
- HW2 due next Monday (9/22)

Agenda

1. Warmup: summations

2. Graphs Part 2 continued

3. Greedy Algorithms
    a. Interval Scheduling

Warmup

A QC factory worker is tasked with inspecting a batch of M&Ms. The pieces of candy are ordered and come by on a conveyor belt one by one. To save time, the worker does not check every piece of candy. Instead, they check the 1st piece of candy, then the 3rd piece, then they allow 4 to pass by before checking the 8th, then they allow 9 to pass by before checking the 18th.

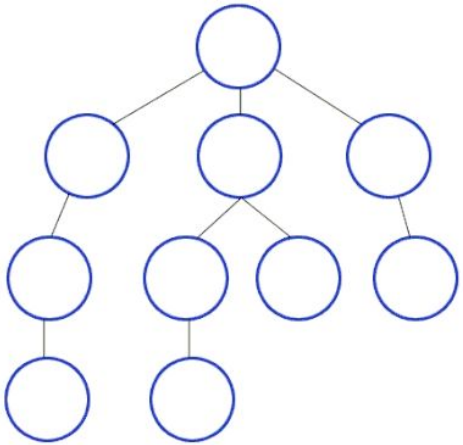
Q1: After 100 pieces of candy have passed by, how many checks have they done?

Q2: What candy piece will be inspected on the 21st check?

Q3: In a batch of 100 M&Ms, if inspected an M&M takes 1 minute, how much time does the worker save by not inspecting each piece?

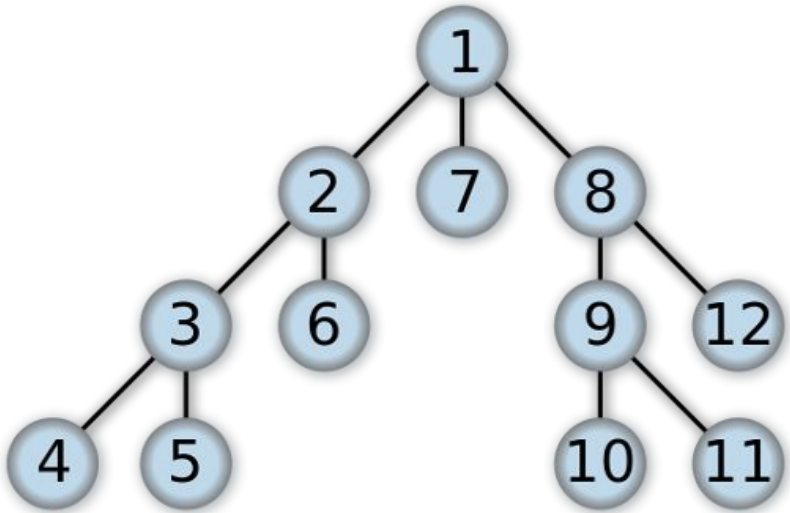
Depth First Search (DFS)

- start at root node and explore as far as possible along each branch
- Recursive algorithm



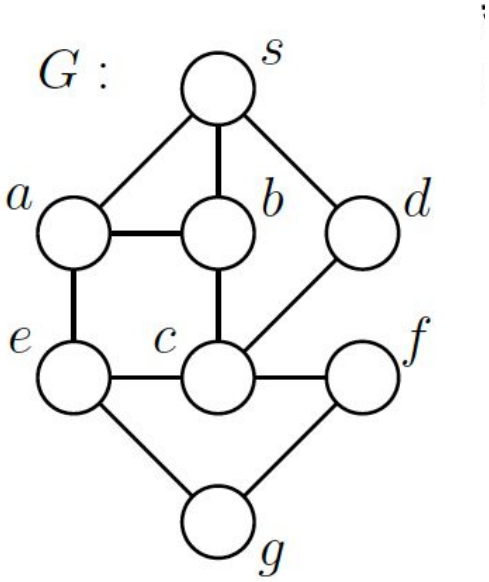
DFS Trace 1

```
DFSG(G) {
  set mark to all false
  for each (v in V) {
    if (!mark(v))
      DFS(v)
  }
}
DFS(u) {
  mark[u] = true  // print u
  for each (v in Adj[u]) {
    if (!mark[v]){
      DFS(v)
    }
  }
}
```

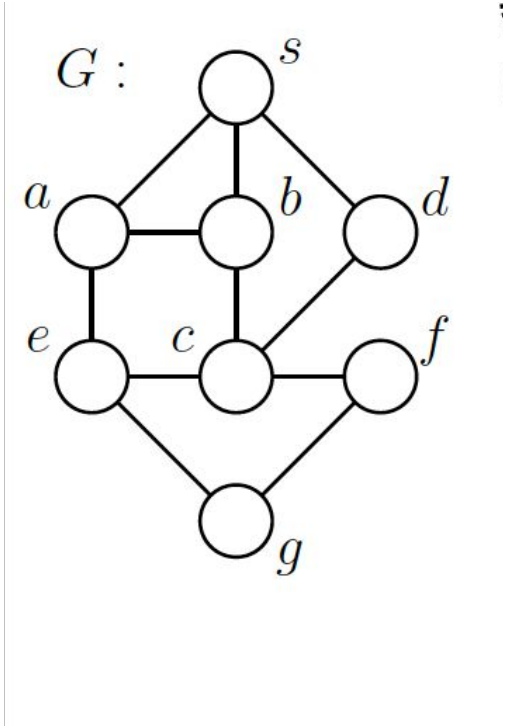


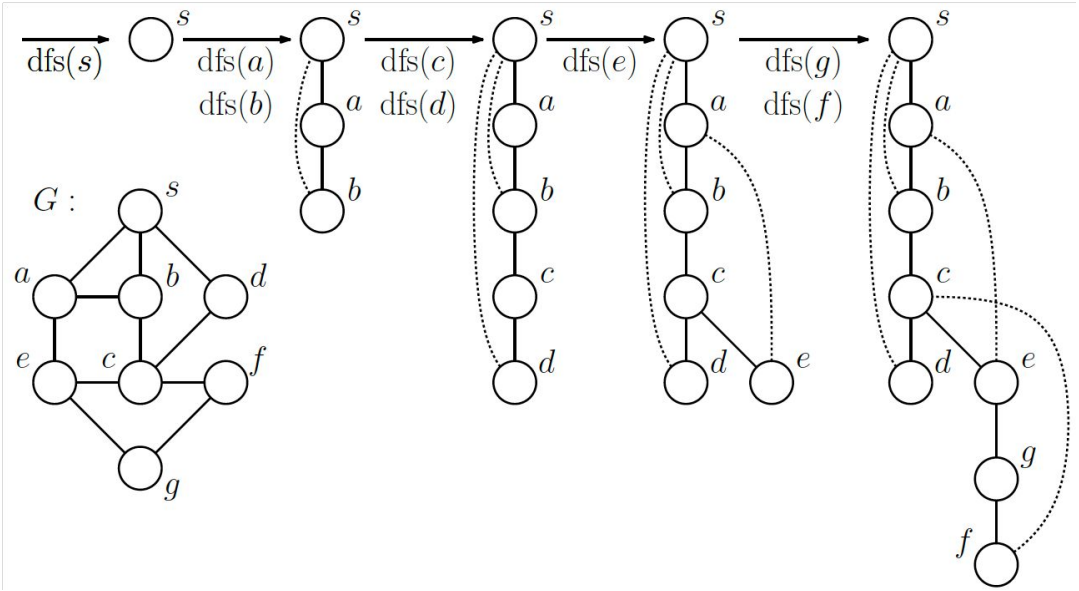
DFS Trace 2

```
DFSG(G) {
  set mark to all false
  for each (v in V) {
    if (!mark(v))
      DFS(v)
  }
}
DFS(u) {
  mark[u] = true  // print u
  for each (v in Adj[u]) {
    if (!mark[v]){
      DFS(v)
    }
  }
}
```



DFS Trace 2





DFS Runtime Analysis

$|V| = n$, $|E| = m$

1. Wrapper:
   a. O(n)

2. Traversal:
   a. DFS is called once per vertex
   b. for-loop: we visit each child of that vertex
      i. Number of iterations depends on the *degree*

$T(n) = n + \sum_{u \in V} (\deg(u) + 1)$

$= n + (\sum_{u \in V} \deg(u)) + n = 2n + \sum_{u \in V} \deg(u)$

$= 2n + 2m$

$= O(n + m)$

DFS with more record keeping

Let’s add code to track:

1. The start and finish time of node processing
    a. Time is the iteration # (kind of...)
        i. Time increases before and after node is processed
    b. Start is the node is printed / marked
    c. End is when all of its children have been processed
2. The predecessor (parent) of each node

DFS with more record keeping

- start times array: `s`
- finish times array: `f`
- predecessors array: `pred`

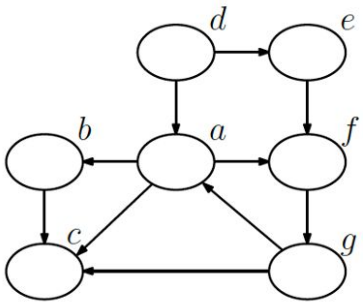
```
DFSG(G) {
  time = 0
  for each (u in V) {
    mark[u] = unseen
  }
  for each (u in V) {
    if (mark[u] == unseen)
      DFS(u)
  }
}
```

```
DFS(u) {
  mark[u] = seen
  s[u] = time++
  for each (v in Adj[u]) {
    if (mark[v] == unseen){
      pred[v] = u
      DFS(v)
    }
  }
  mark[u] = finished
  f[u] = time++
}
```

DFS with more record keeping

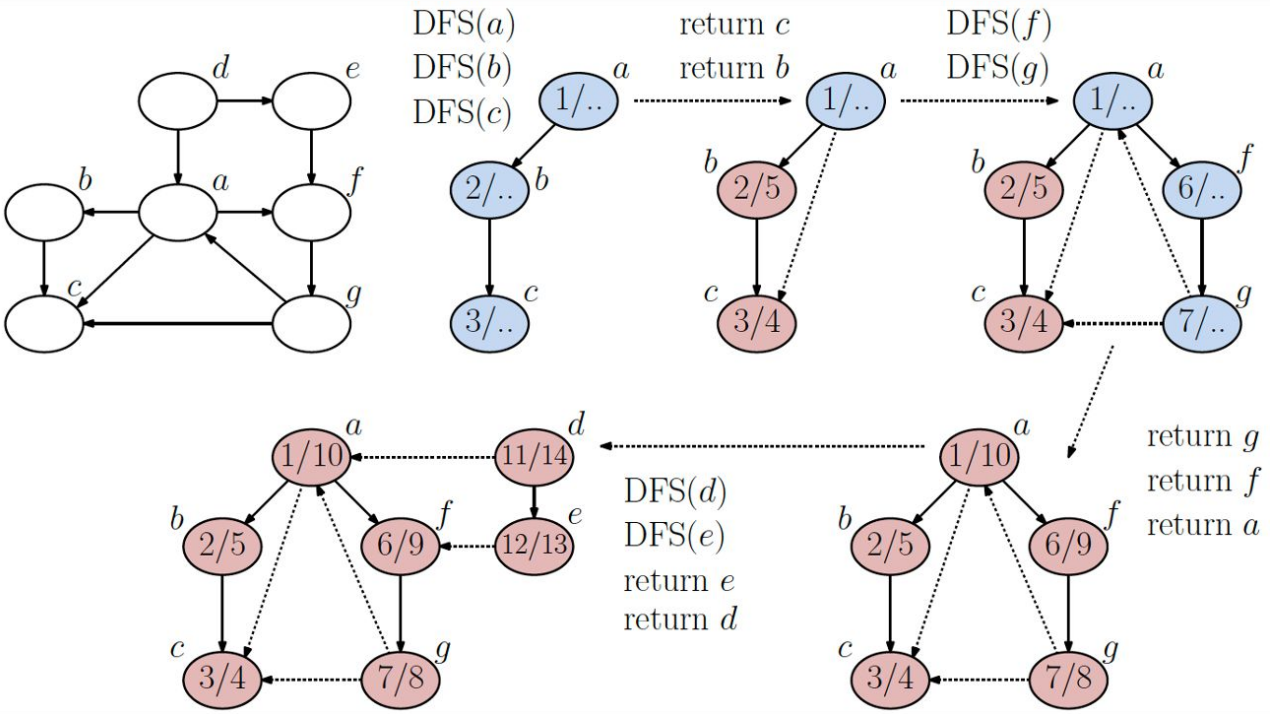
- start times array: `s`
- finish times array: `f`
- predecessors array: `pred`

```
DFSG(G) {
  time = 1
  for each (u in V) {
    mark[u] = unseen
  }
  for each (u in V) {
    if (mark[u] == unseen)
      DFS(u)
  }
}
```

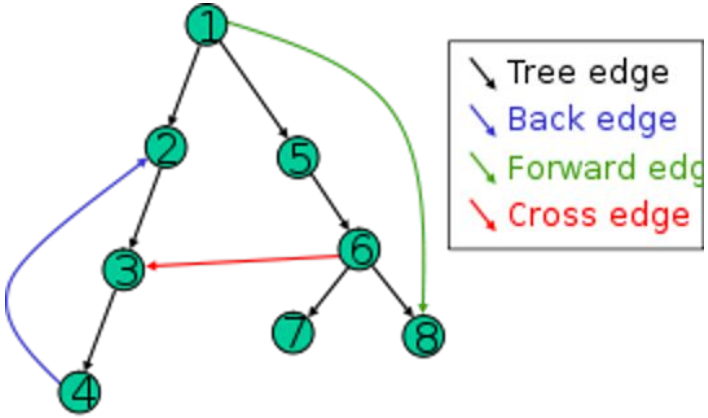



```
DFS(u) {
  mark[u] = seen
  s[u] = time++
  for each (v in Adj[u]) {
    if (mark[v] == unseen){
      pred[v] = u
      DFS(v)
    }
  }
  mark[u] = finished
  f[u] = time++
}
```

DFS with more record keeping



DFS Edge Classification



- If v is visited for the first time as we traverse (u, v), then (u, v) is a tree edge
- else, v has already been visited
  - if v is an ancestor of u, (u, v) is a back edge
  - if v is a descendent of u, then (u, v) is a forward edge
  - if v is neither, then (u, v) is a cross edge

Greedy Algorithms

Greedy Algorithms

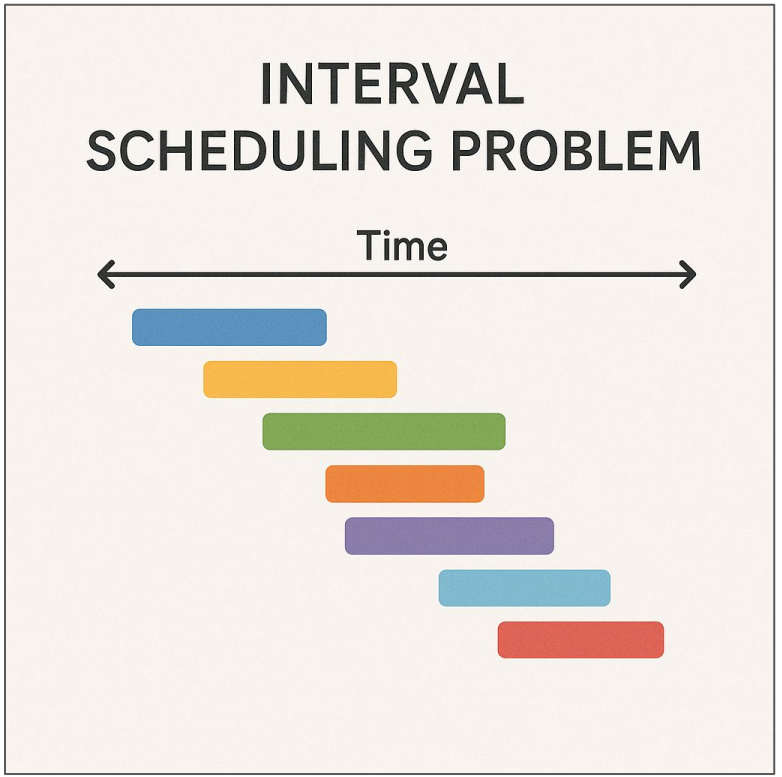
- An algorithm that builds up a solution by *myopically* selecting the best choice at the moment

- Greedy algorithms don’t always produce optimal solutions, but sometimes they do!

- We will investigate the pros and cons of short-sighted greed in the design of algorithms

Interval Scheduling

1. Explanation of the problem
2. A greedy algorithm for scheduling
3. Runtime analysis
4. Proof of correctness



Interval Scheduling

You have a resource (lecture room, supercomputer, pool, hockey rink, electron microscope...) that can be used by at most one person / group at a time

Many requests come in to use that resource for periods of time

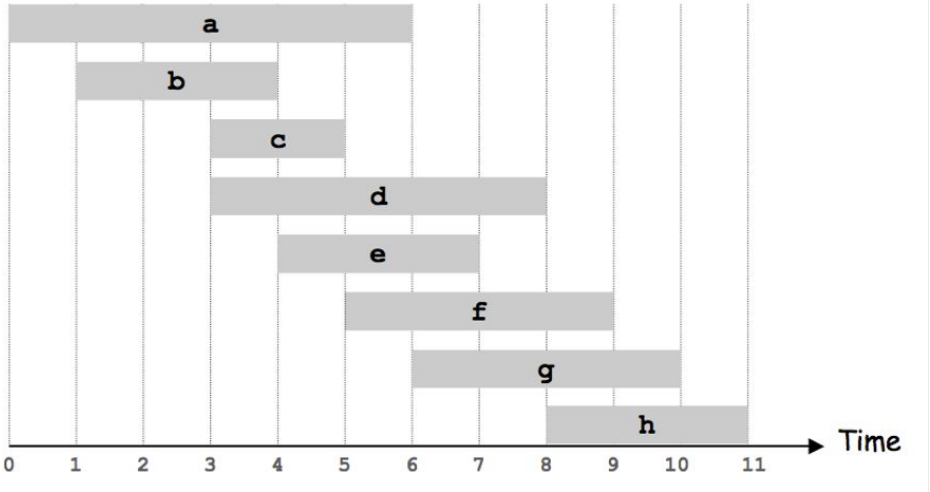
A request takes the form: *Can I reserve the resource starting at time **s** until time **f**?*

Goal: schedule as many requests as possible

Interval Scheduling

- Given a set R of n activities with start-finish times $[s_i, f_i], 1 \leq i \leq n$, determine a maximum subset of R consisting of compatible requests

What does compatible mean?



Interval Scheduling - compatibility

Two requests i and j are compatible if the requested intervals do not overlap:

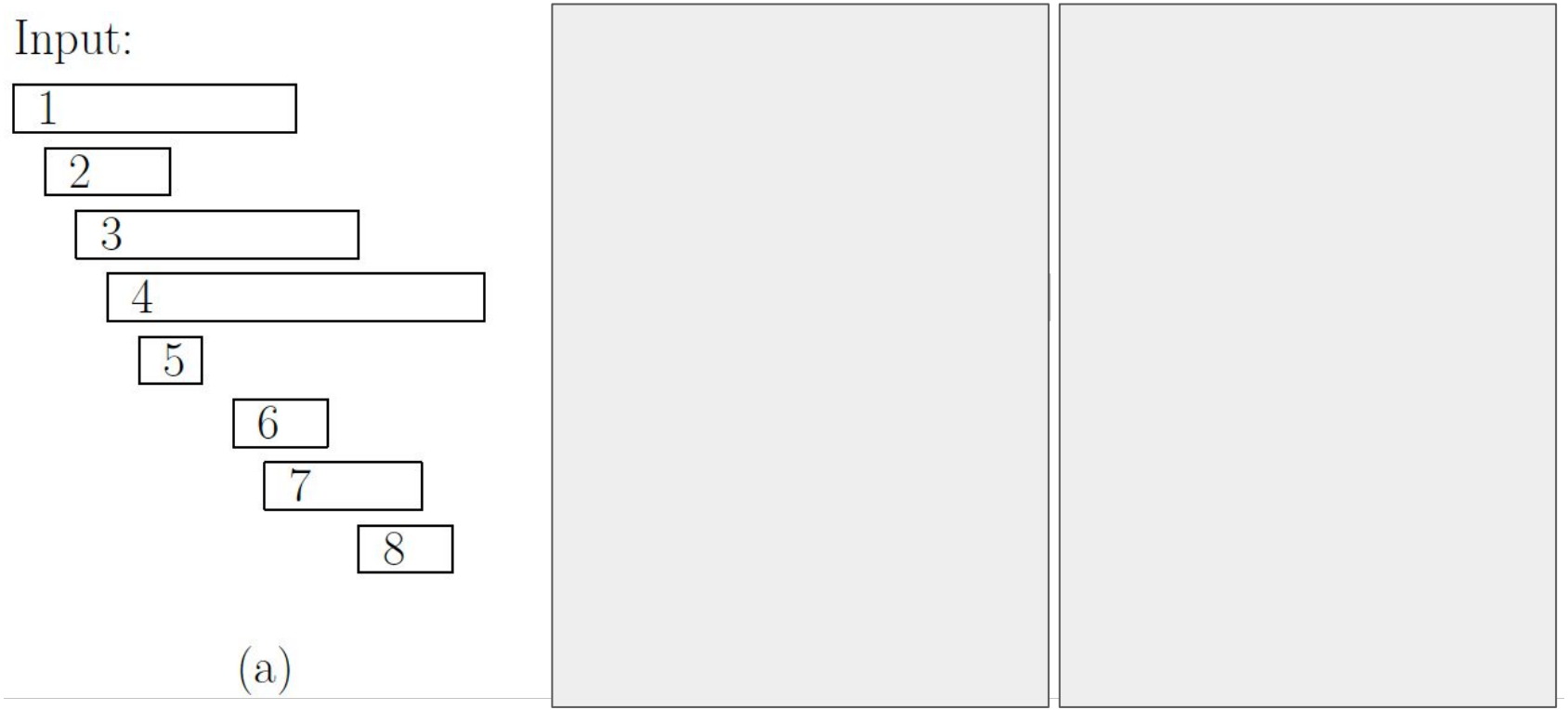
Either

(a) request i is for an earlier time interval than request j ($f_i \leq s_j$) OR
(b) Request i is for a later time than request j ($f_j \leq s_i$)

A subset A of requests is compatible if all pairs of requests i, j $\in$ A, i != j are compatible.

Goal of interval scheduling is to select a compatible subset of requests of **maximum size**

Interval Scheduling - Example

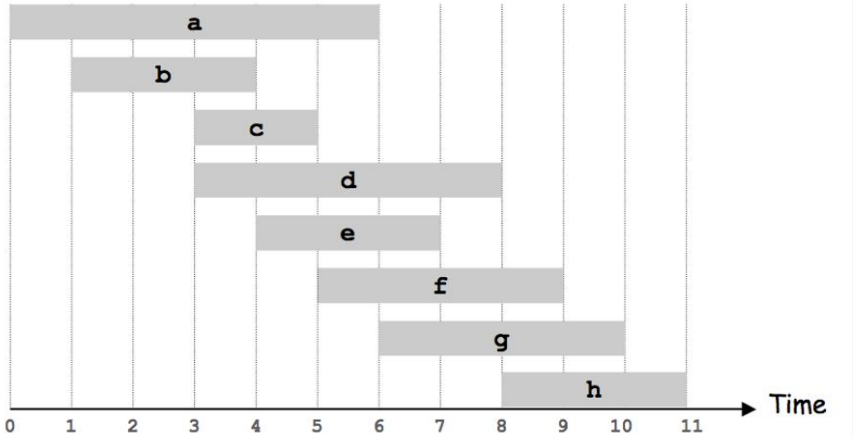


A Greedy Solution

- For each request, use simple criteria to decide if it should be accepted

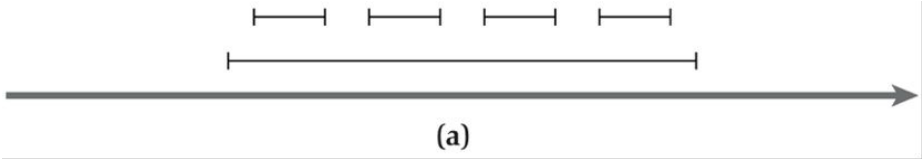
- Once accepted it can not be rescinded
  - Greedy algorithms do not backtrack

- What criteria should we use here?

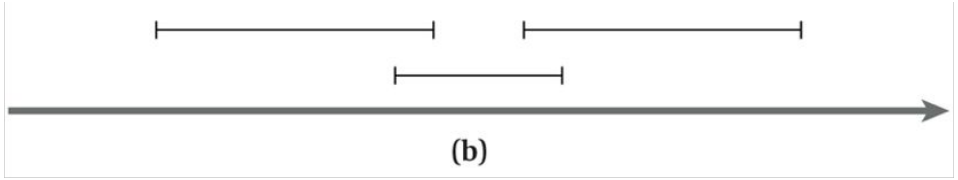


Greedy Criteria Ideas

1. Select the request that starts first

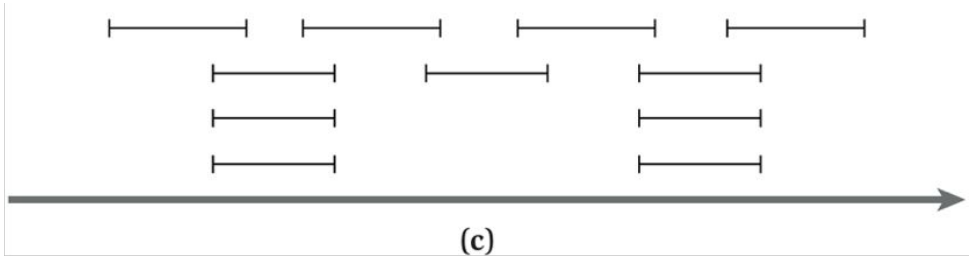


2. Select the interval that takes the least amount of time

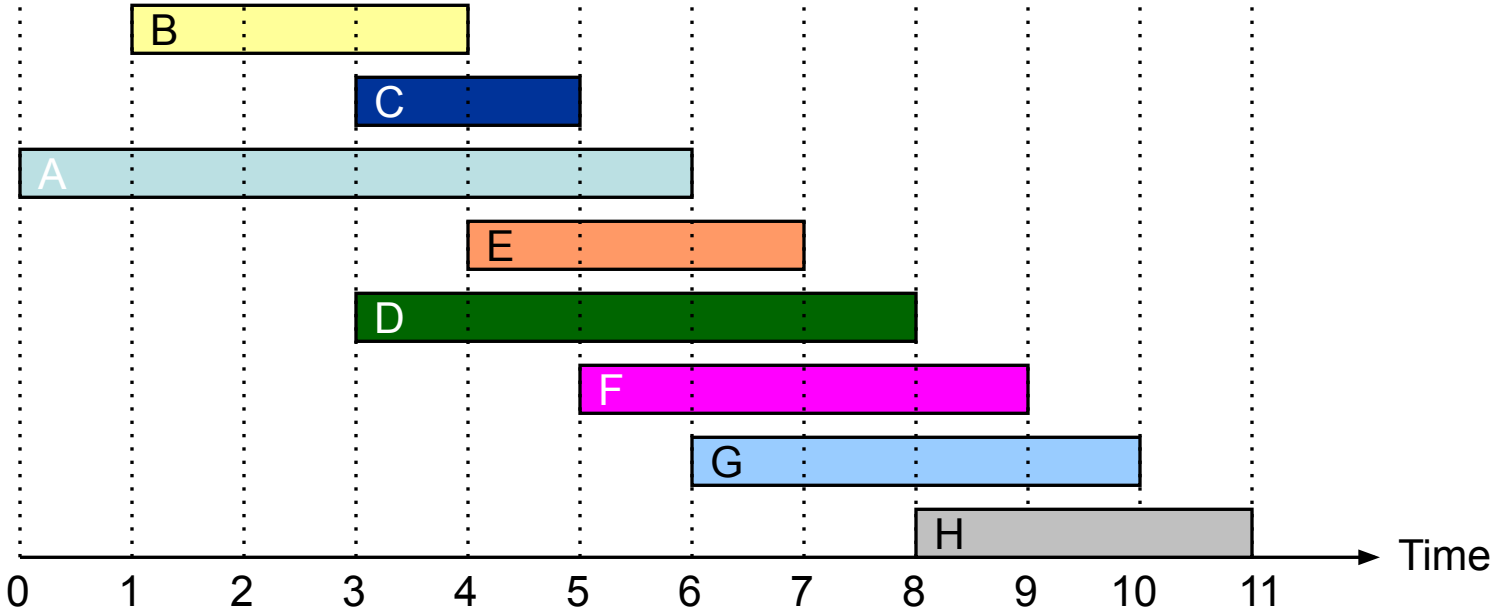


3. Select the interval with the fewest conflicts

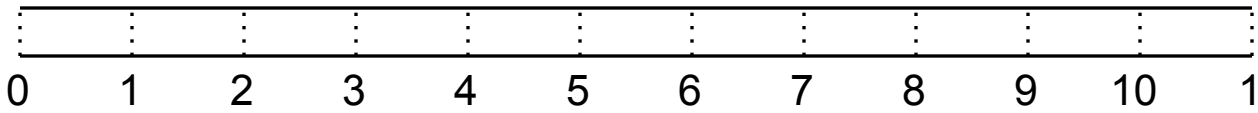
4. **Select the interval that finishes first**



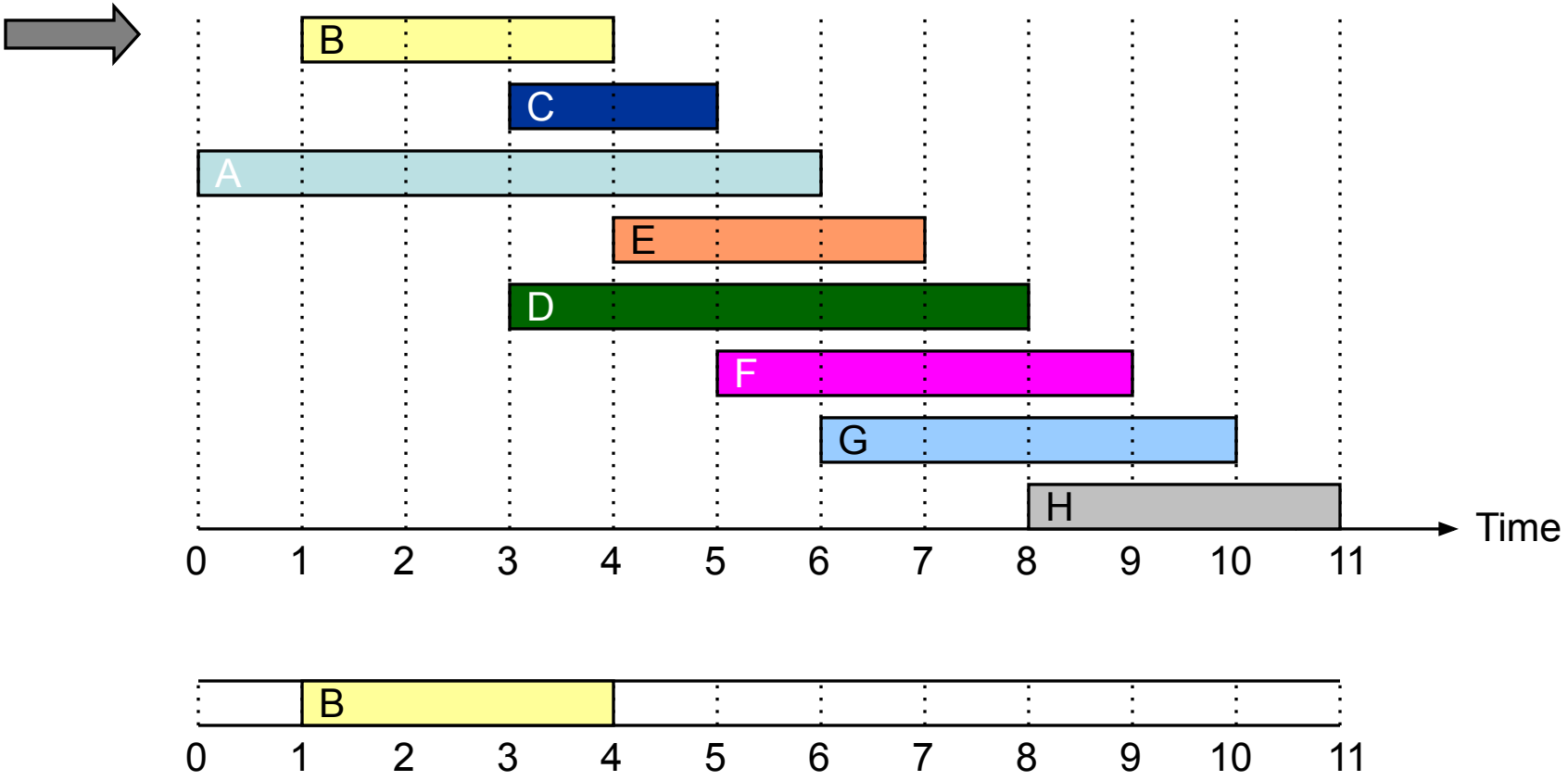
Interval Scheduling: Select Earliest Finish



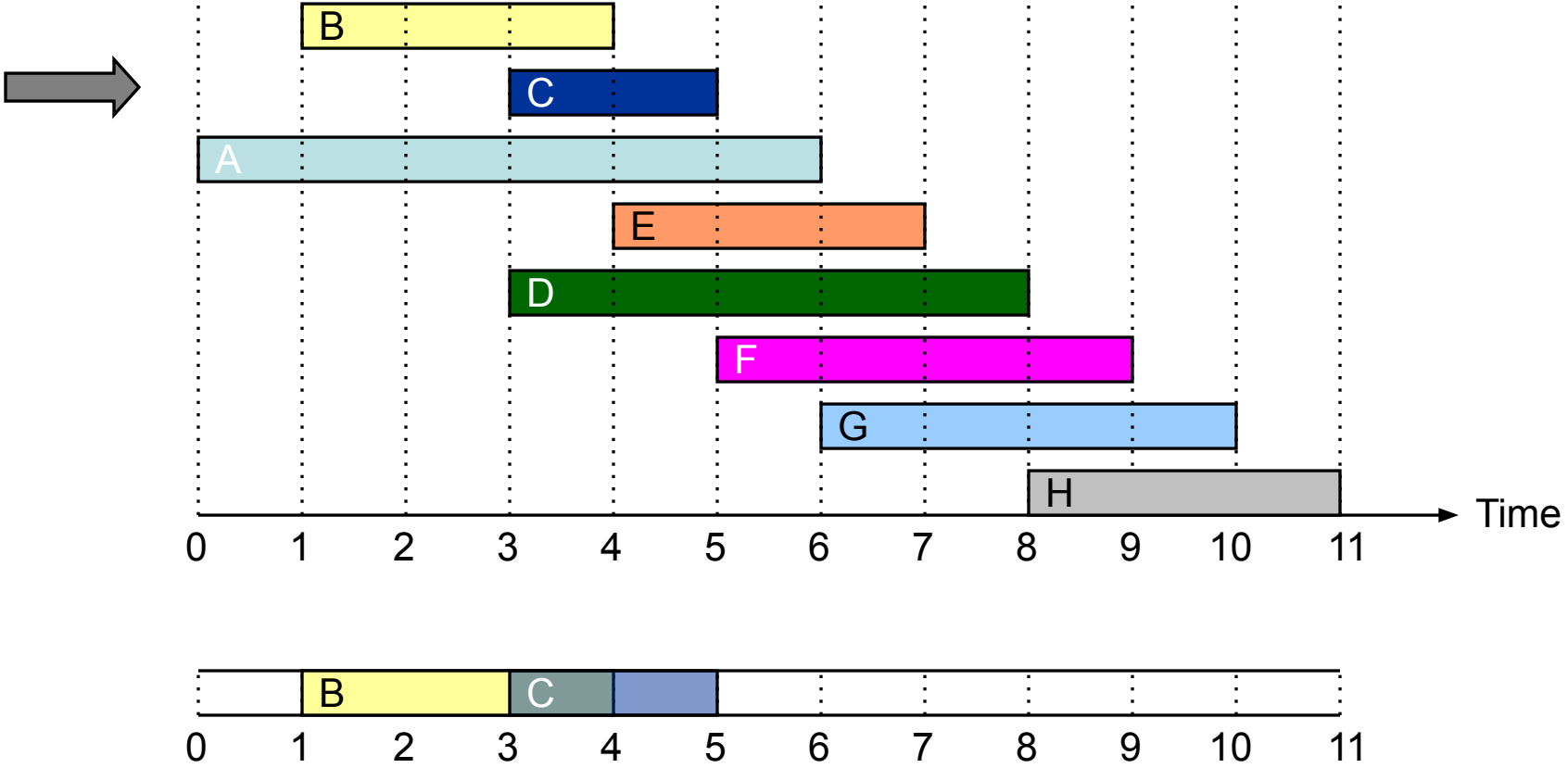
Fill in the schedule:



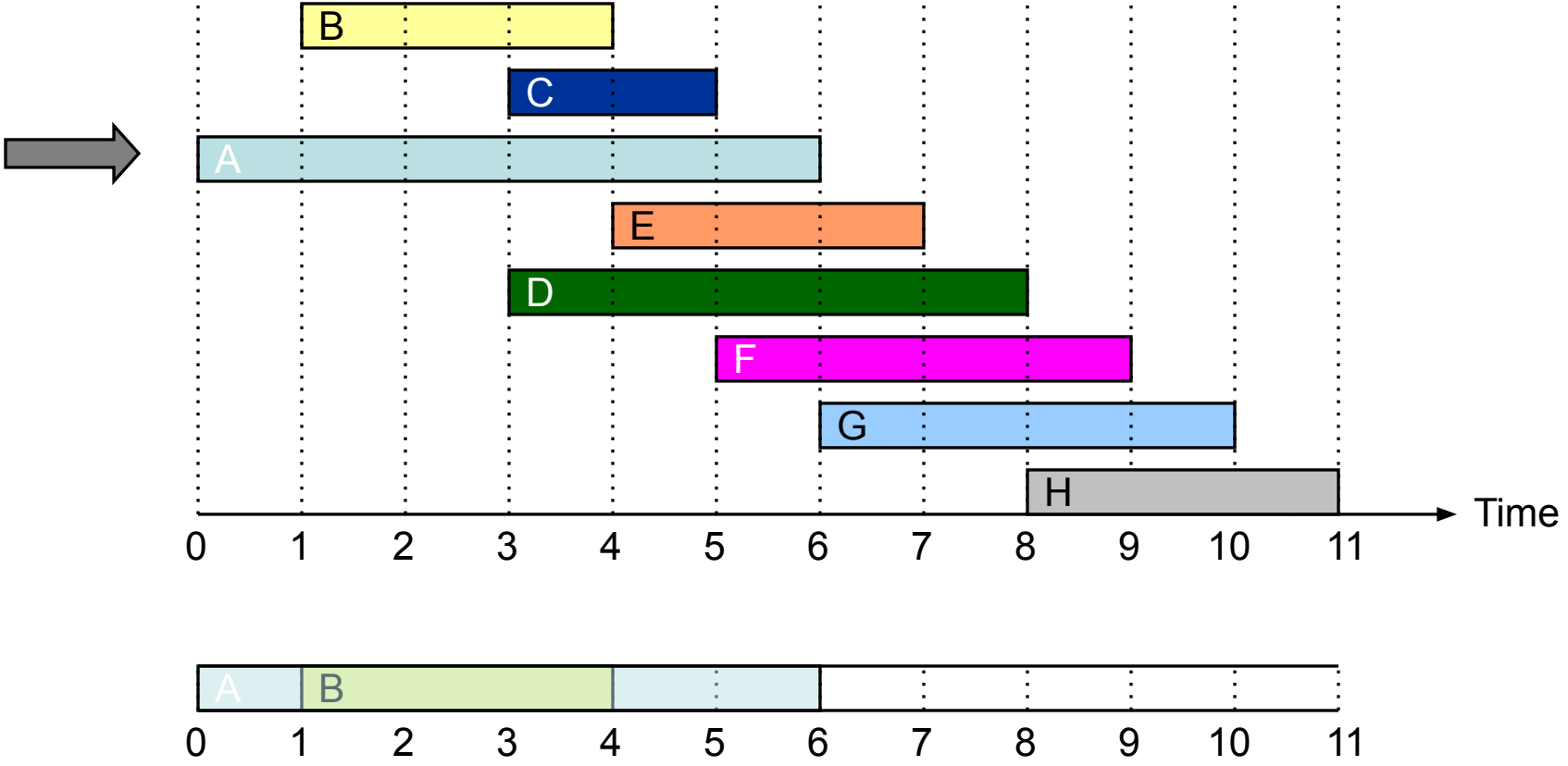
Interval Scheduling: Select Earliest Finish



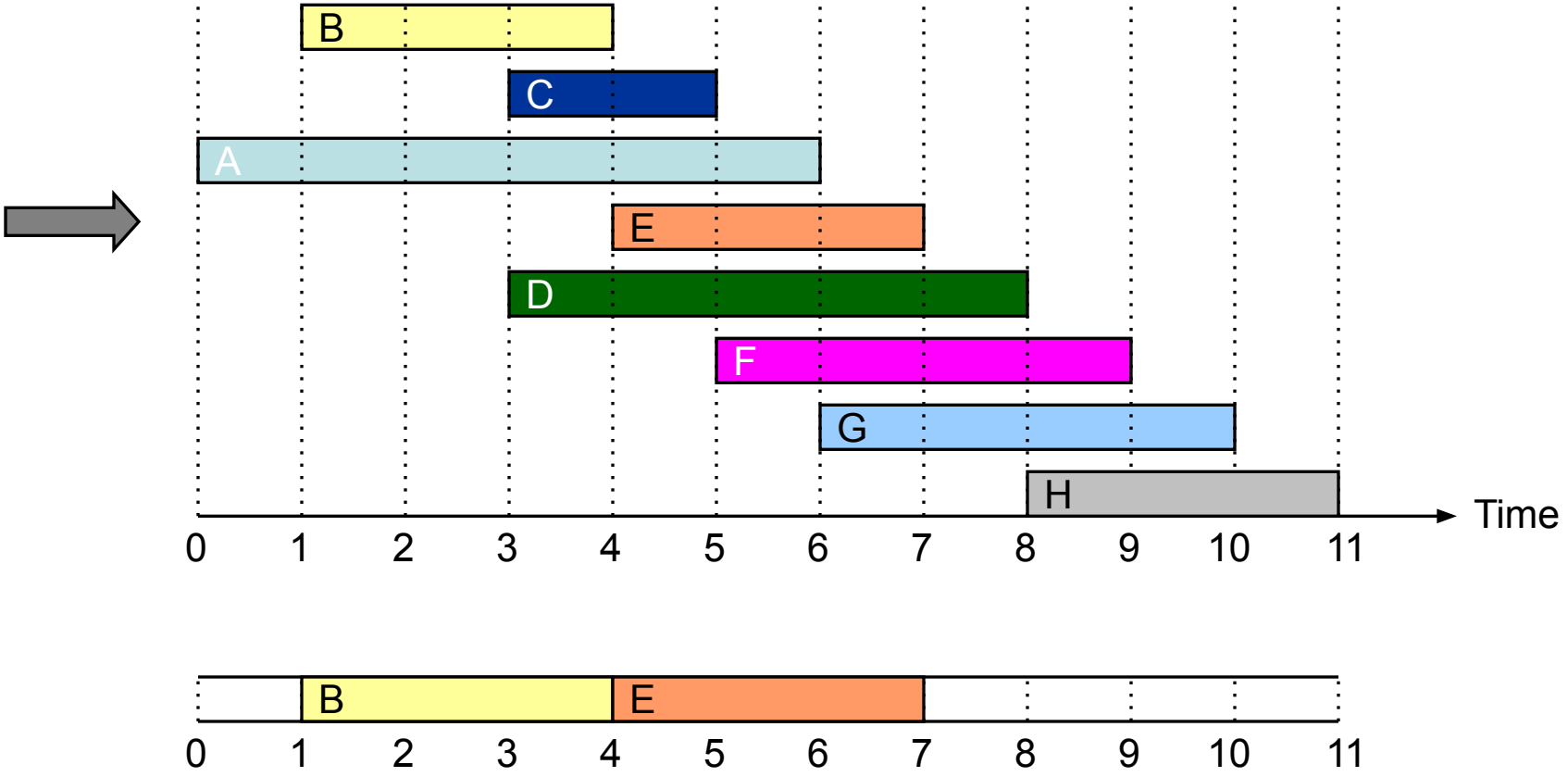
Interval Scheduling: Select Earliest Finish



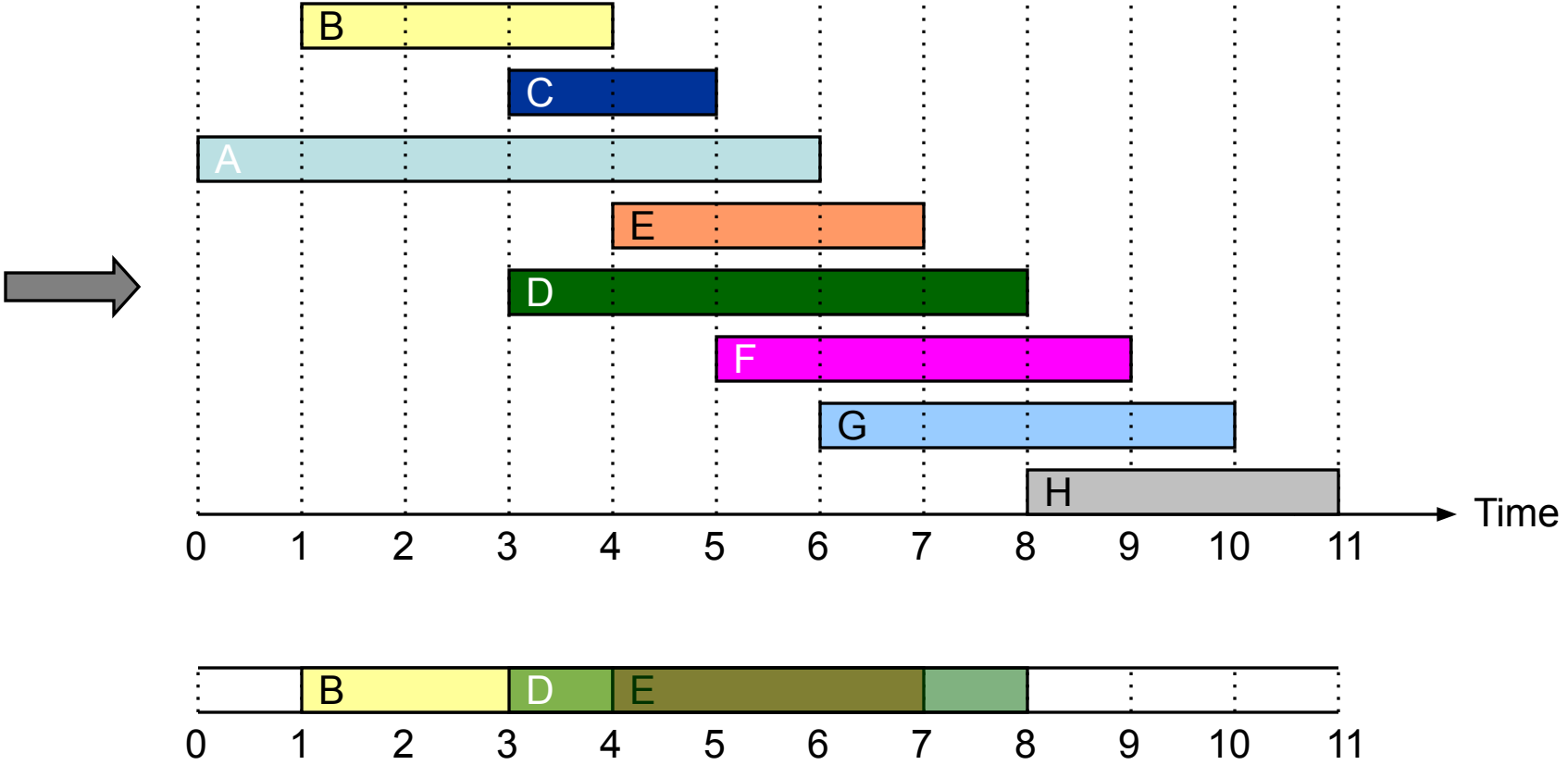
Interval Scheduling: Select Earliest Finish



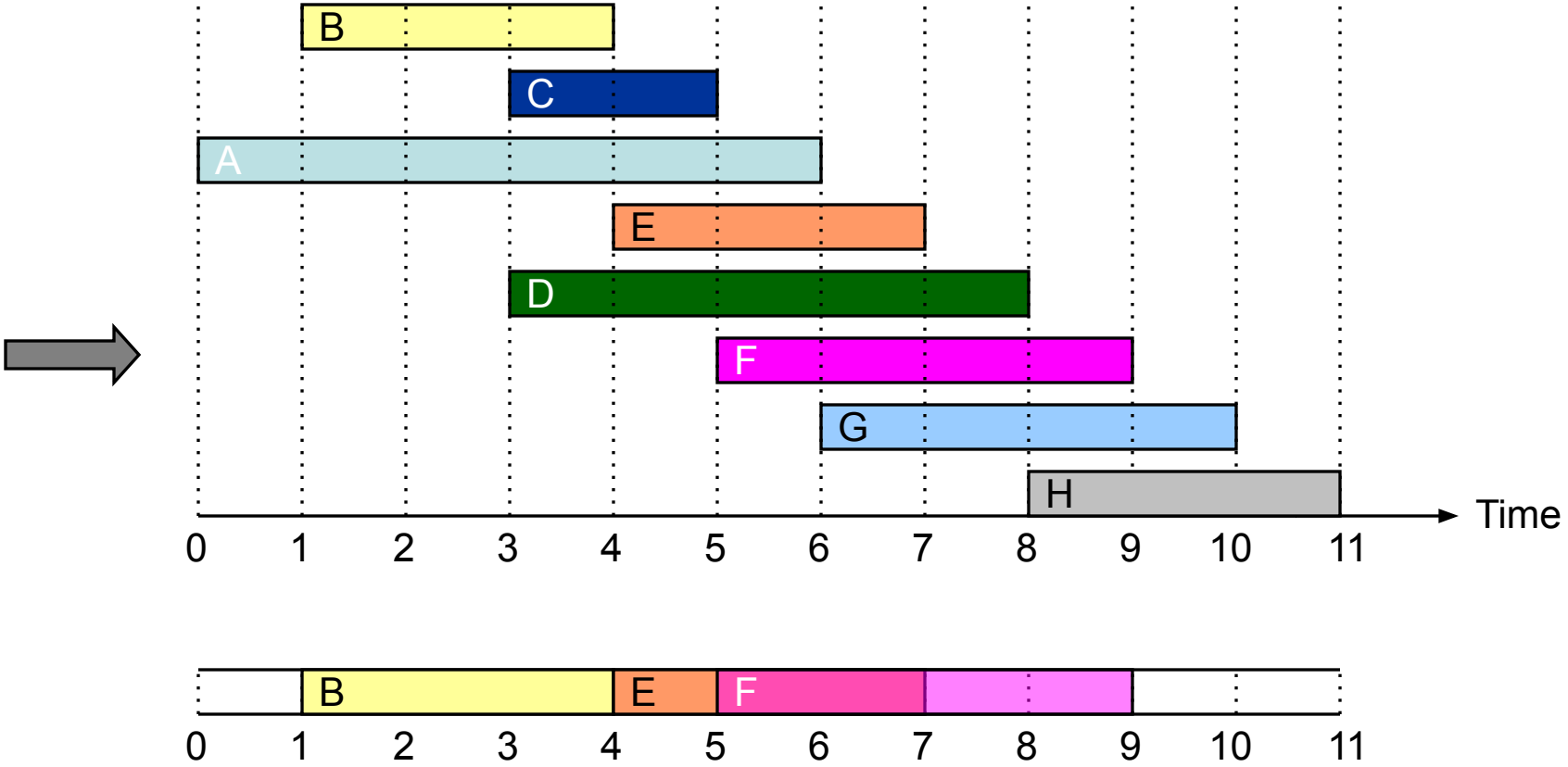
Interval Scheduling: Select Earliest Finish



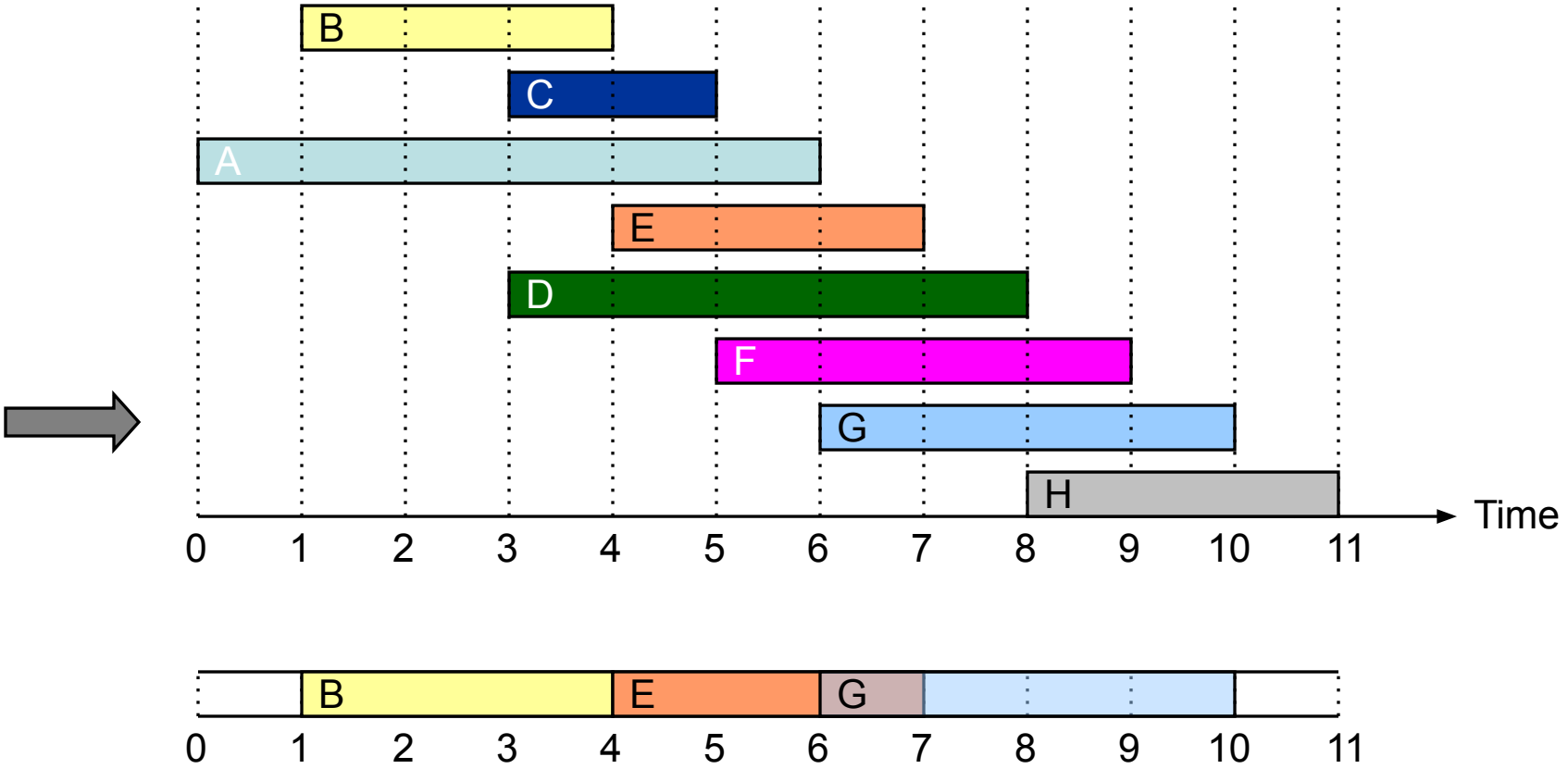
Interval Scheduling: Select Earliest Finish



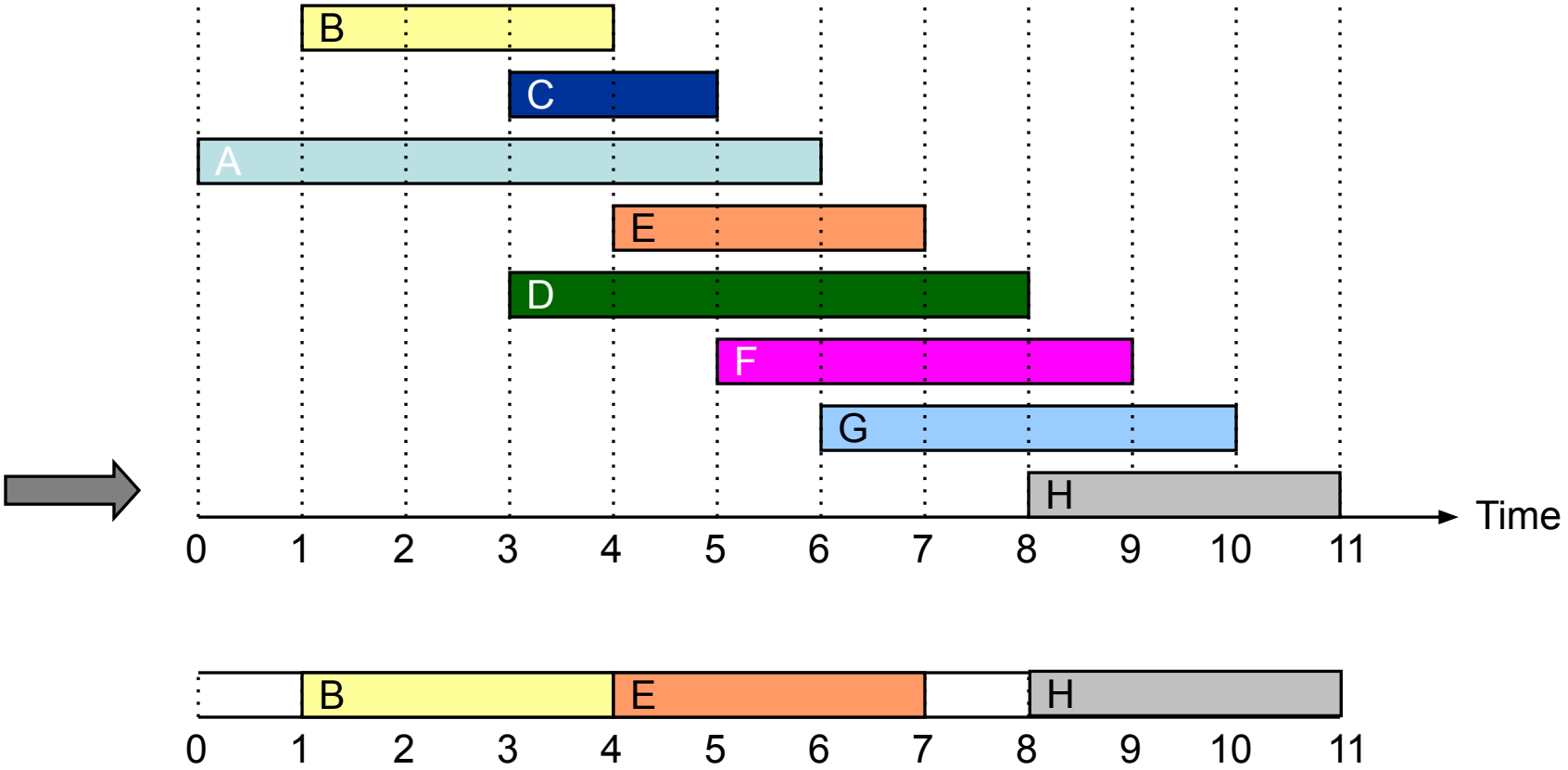
Interval Scheduling: Select Earliest Finish



Interval Scheduling: Select Earliest Finish



Interval Scheduling: Select Earliest Finish



Select Earliest Finish (Not Optimized) - Runtime Analysis

```
greedySchedule(R) { // R the set of requests
  A = empty // A the set of scheduled activities
  while (R is nonempty) {
    r = request in R with the smallest finish time
    append r to A
    delete from R all requests that overlap r
  }
  return A
}
```

|R| = n

- While loop runs how times?
  - n
  - O(n)
- Finding request with smallest finish time
  - O(n)

Total runtime of O(n*n) = **$O(n^2)$**

Q: Can we do better?

Select Earliest Finish (Optimized) - Runtime Analysis

```
greedySchedule(R) { // R the set of requests

     A = empty; // A the set of scheduled activities
     sort R by finish times
     prevA = null //last picked activity

     for each (r in R) {
          if (r does not conflict with prevA) {
               append r to A;
               prevA = r;
          }
     }
     return A;
}
```

|R| = n

- Sorting:
  - O(nlogn)
- For loop runs n times
  - O(n)
- Checking conflict:
  - O(1)

Total runtime of O(nlogn + n) = **O(nlogn)**

Proof of Correctness

Greedy solutions are very natural, but it is certainly not obvious they return an optimal solution.

Method for proving that a greedy algorithm establishes an optimal solution:

1. *The greedy algorithm stays ahead -*
   a. If we measure the greedy algorithm's progress in a step-by-step fashion, one sees that it does better than any other algorithm at each step
   b. It follows that it produces an optimal solution

2. *Exchange argument -*
   a. The optimal solution is equivalent to the greedy solution
   b. Consider any possible solution to the problem and gradually transform it into the solution found by the greedy algorithm without hurting its quality

Proof of Correctness

1. Termination:
    a. The algorithm terminates in n steps

2. Correctness: *The greedy algorithm stays ahead*
    a. We will prove **lemma:** our greedy criteria has earlier (or equal) finish times compared to any other solution
    b. Then, we will use this to show that our solution is optimal

Lemma: Our greedy criteria finishes first

We will show that our greedy algorithm has earlier (or equal) finish times compared to any other solution.

- Given $O = \langle [os_1, of_1], \dots, [os_i, of_i], \dots \rangle$ and $G = \langle [gs_1, gf_1], \dots, [gs_i, gf_i], \dots \rangle$,
- Claim: $gf_i \leq of_i, \forall i$
-

Lemma: Our greedy criteria finishes first

- Given $O = \langle [os_1, of_1], \ldots, [os_i, of_i], \ldots \rangle$ and $G = \langle [gs_1, gf_1], \ldots, [gs_i, gf_i], \ldots \rangle$,
- Claim: $gf_i \leq of_i, \forall i$
- Proof by induction
  - base case $i = 1$: by greedy construction
  - inductive hypothesis: $gf_{i-1} \leq of_{i-1}$

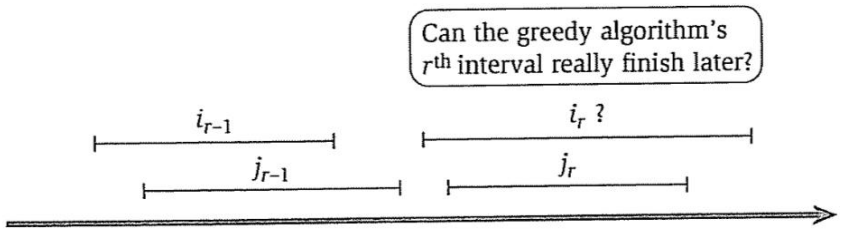


Figure 4.3 The inductive step in the proof that the greedy algorithm stays ahead.

Proof of Correctness

1. Termination:
    a. The algorithm terminates in n steps

2. Correctness: *The greedy algorithm stays ahead*
    a. We will prove **lemma:** our greedy criteria has earlier (or equal) finish times compared to any other solution
    b. Then, we will use this to show that our solution is optimal

Proof of Correctness

By induction, we showed: $gf_i \leq of_i, \forall i$

We will use this to show that our set of requests is optimal: |O| = |G|

Proof by contradiction:

- |O| > |G|
- There is some request k that is in O, but not in G
- By IH, our last interval finishes before O's last interval
- So, k must be compatible with our set G, and we would have picked it

Writeup of proof posted under resources tab

Summary

1. DFS:
    a. recursive
    b. O(m+n)
    c. Track pred to record the path

2. Greedy algorithms
    a. Prove by showing your solution “stays ahead” at each step

3. Upcoming deadlines:
    a. Lab 1 due tomorrow
    b. HW2 due Sep 22nd (Monday)

4. Next class: Dijkstra's Algorithm