GREEDY ALGORITHMS

Lecture 11, CMSC 142

Previous Topic(s)

- Minimum Edit Distance
- Longest Increasing Subsequence

Today's Topics

- Greedy Algorithms
- Activity Selection

Recall: Dynamic Programming

- Solves subproblems by combining solutions to subproblems that contain common sub-sub-problems

- Difference between DP and Divide-and-Conquer:
  - Using Divide and Conquer to solve these problems is inefficient as the same common sub-sub-problems have to be solved many times
  - DP will solve each of them once and their answers are stored in a table for future reference

Elements of Dynamic Programming

Optimal substructure

- An optimal solution to the problem contains within it optimal solutions to subproblems.

Overlapping subproblems

- There exist some places where we solve the same subproblem more than once

Steps to Designing a DP Algorithm

- Characterize optimal substructure
- Recursively define the value of an optimal solution
- Compute the value bottom-up

Design of Algorithms

- Brute-Force Approach
- Divide and Conquer
- Dynamic Programming
- Greedy Approach

Greedy Approach

Being greedy

- A greedy man takes as much as he can, as often as he can.
- At some point in our life, we have made greedy decisions.

Being greedy

- **Example:** When we go shopping or when we commute, we make choices that seem best for the moment.

- This myopic (or short-sighted) decision-making behavior can be applied to algorithms too.

Chess vs Scrabble

- A game like chess can only be won by **thinking ahead**. *(not greedy, strategized.)*
- But in Scrabble, you can do well simply by making whichever move seems **best at the moment** and not worry too much about future consequences *(greedy)*

Optimization Problems

- For many optimization problems, using Dynamic Programming to determine the best choice is **overkill**

- Simpler, more efficient algorithms using the greedy approach will do.

Greedy Algorithms

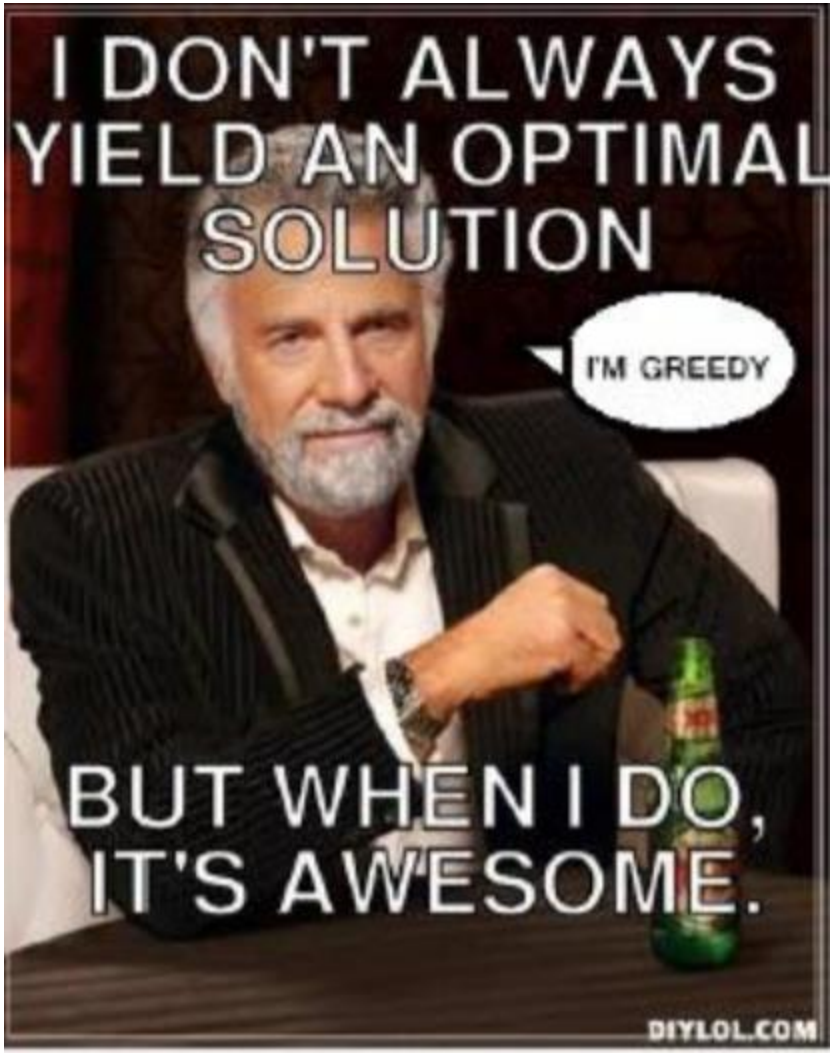
- build up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit

- It makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution

Will this approach **always** yield optimal solutions?

Greedy Algorithms

- Greedy algorithms do not always yield optimal solutions, but for many problems they do.


I DON'T ALWAYS YIELD AN OPTIMAL SOLUTION
I'M GREEDY
BUT WHEN I DO, IT'S AWESOME.
DIYLOL.COM

When does greedy algorithm work?

Basic Ingredients for a Greedy Algorithm

- Optimal Substructure Property
- Greedy Choice Property

Optimal Substructure Property

- An optimal solution contains **optimal solutions to subproblems**.
- This ensures that solving a smaller problem optimally leads to the best overall solution.

Greedy choice property

- One can always arrive at a global optimal solution by making a locally optimal choice

- At every step, we consider only what is best in the **current** problem,

- **Not considering the results of the subproblems.**

Greedy vs **DP**

- This is where greedy algorithms differ from dynamic programming

- In DP, we make a choice at each step, but the choice usually **depends on the solution to subproblems**.

- In greedy algorithms, we make whatever choice seems **best at the moment** but it doesn't depend on any future choices or solutions to subproblems

Analogy

- DP plays it safe → “sigurista”
- Greedy is a risk-taker → “YOLO”

Greedy vs DP

Can I make the best choice at each step without looking ahead or reconsidering previous choices?

✅ **Yes** → A greedy algorithm might work.

❌ **No** → You may need **dynamic programming (DP)**.

Does picking the local best option at each step always lead to the global best solution?

- ✅ **Yes** → A greedy algorithm will work.
- ❌ **No** → You need DP or another approach.

Greedy Algorithms

- Activity Selection
- Fractional Knapsack
- Set Cover
- Huffman Encoding

Algorithm for Greedy Algorithm

```
Greedy(A, n):
    //a[1..n] contains n inputs
    Solution = {}
    for i=1 to n:
        x = select(a)
        if feasible(Solution, x):
            Solution = union(Solution, x)
    return Solution;
```

Activity Selection

Introduction

- Imagine you're on an international developer's conference and most *imba* developers are present
- The conference has a list of activities that you can do for the whole day.
- Since there are a lot of activities, some of these activities are overlapping.
- Naturally, you want to maximize the number of activities that you do for a day

Introduction

- Given the list of activities and the start and finish times of each, which activities should you do to maximize the number of activities you do for a day?

Introduction

- Given the list of activities and the start and finish times of each, which activities should you do to maximize the number of activities you do for a day?

- Luckily, you took CMSC 142 so this problem will be easy to solve.

Activity Selection

- This is known as the **activity selection** problem

- You want to select a **maximum-size subset** of mutually *compatible activities* from a set of activities

Compatibility

Activities a_i and a_j are said to be compatible if the intervals

$$[s_i, f_i) \text{ and } [s_j, f_j)$$

do not overlap.

That is, $s_i \geq f_j \text{ or } s_j \geq f_i$

It is helpful to draw the activities on a **timeline** to immediately see which activities are conflicting

Activity Selection

Input: set $A = \{a_1, a_2, \ldots, a_n\}$ of activities

$S = \{s_1, s_2, \ldots, s_n\}$ (start times for each activity)

$F = \{f_1, f_2, \ldots, f_n\}$ (finish times for each activity)

where $0 \leq s_t < f_t$

Output: maximum-size subset of mutually-compatible activities

Applications

- Scheduling problems
- *Example: CPU Process Scheduling*

Example

- A = {x, y, z}
- S = {1, 3, 5}
- F = {4, 5, 7}

Example

$x(1, 4)$, $y(3, 5)$, $z(5\text{-}7)$

Check statements that are true:

x and z are compatible ______

y and z are compatible ______

x and y are compatible ______

Example

x(1, 4), y(3, 5), z(5-7)

Check statements that are true:

x and z are compatible ______ ✓

y and z are compatible ______ ✓

x and y are compatible ______

Greedy Idea

- Greedily pick an activity
- Add that activity to the answer
- Remove that activity and all conflicting activities from the set of activities
- Repeat until set of activities is empty

Efficient Greedy Heuristic

- Once you've identified a reasonable greedy **heuristic**, prove that it always gives the correct answer, then develop an efficient solution

Stays Ahead

- Show that no matter what other solution someone provides you, the solution provided by your greedy algorithm always "stays ahead", and no other choice could do better

How do we greedily pick an activity?

Possible Greedy Heuristics

- Select activity that starts the earliest
- Select the shortest activity
- Select the activity that ends the earliest
- Select the activity with minimum conflicts

Early start

- Does not yield optimal solution

- If the earliest activity is for a very long interval, then by doing the early activity we may have to reject a lot of other activities

Early start

Shortest activity

- Pick the activity with the **smallest duration** to leave more time for others.
- A short activity might **block** longer but more optimal choices.

Ends Early

- How about picking "Select the activity that ends the earliest" as our greedy heuristic?

- That is, the activity where f_i is as small as possible

Ends Early

- Will yield an **optimal** solution

- Idea: If we become free as soon as possible, we can maximize the time left to do other activities

Demo

Points to remember

- **Input:** list of activities with their starting time and finishing time
- Our goal is to select maximum number of non-conflicting activities that can be performed by a person or a machine, assuming that the person or machine involved can work on a single activity at a time
- Any two activities are said to be conflicting if starting time of one activity is greater than or equal to the finishing time of the other activity
- In order to solve this problem, we first sort the activities as per their finishing time in ascending order.
- Then we select non-conflicting activities

Example

Activity	a1	a2	a3	a4	a5	a6	a7	a8
start	1	0	1	4	2	5	3	4
finish	3	4	2	6	9	8	5	5

Steps

- Sort the activities as per finishing time in ascending order
- Select the first activity
- Select the new activity if it’s starting time is greater than or equal to the previously selected activity

Repeat step 3 till all activities are checked.

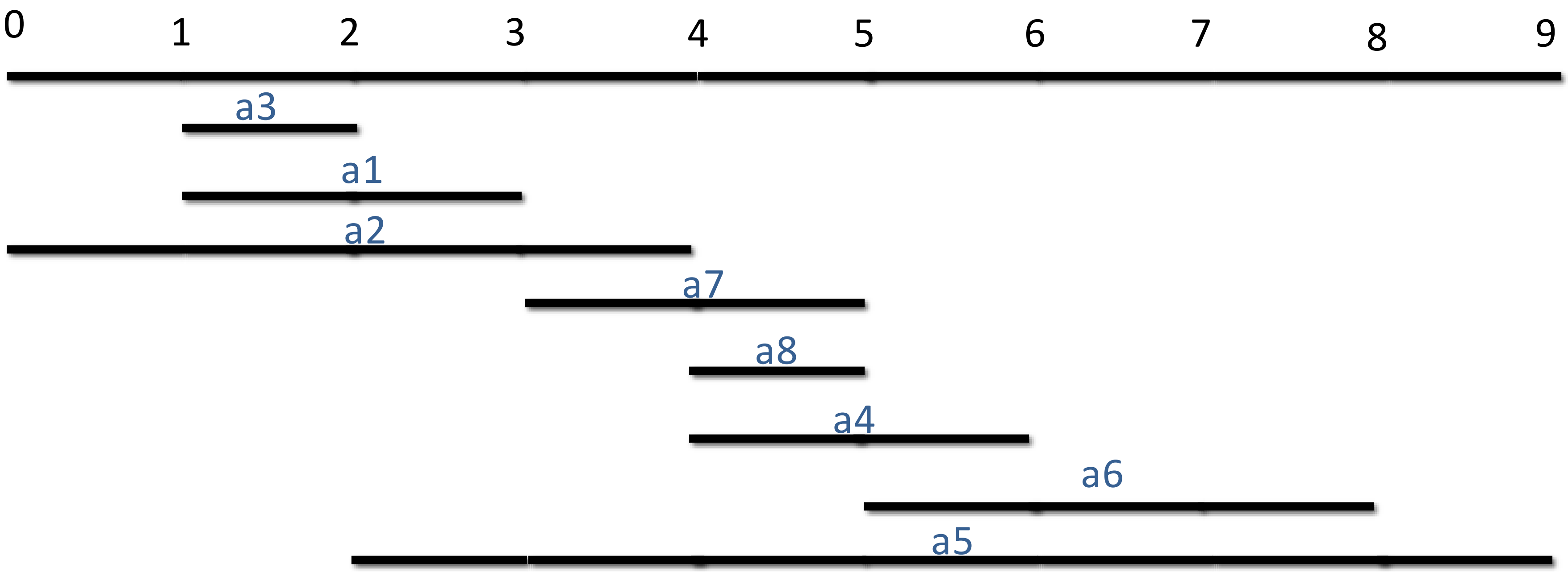
Step 1: Sort the activities as per finishing time in descending order

Activity	a1	a2	a3	a4	a5	a6	a7	a8
start	1	0	1	4	2	5	3	4
finish	3	4	2	6	9	8	5	5

Sorted Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

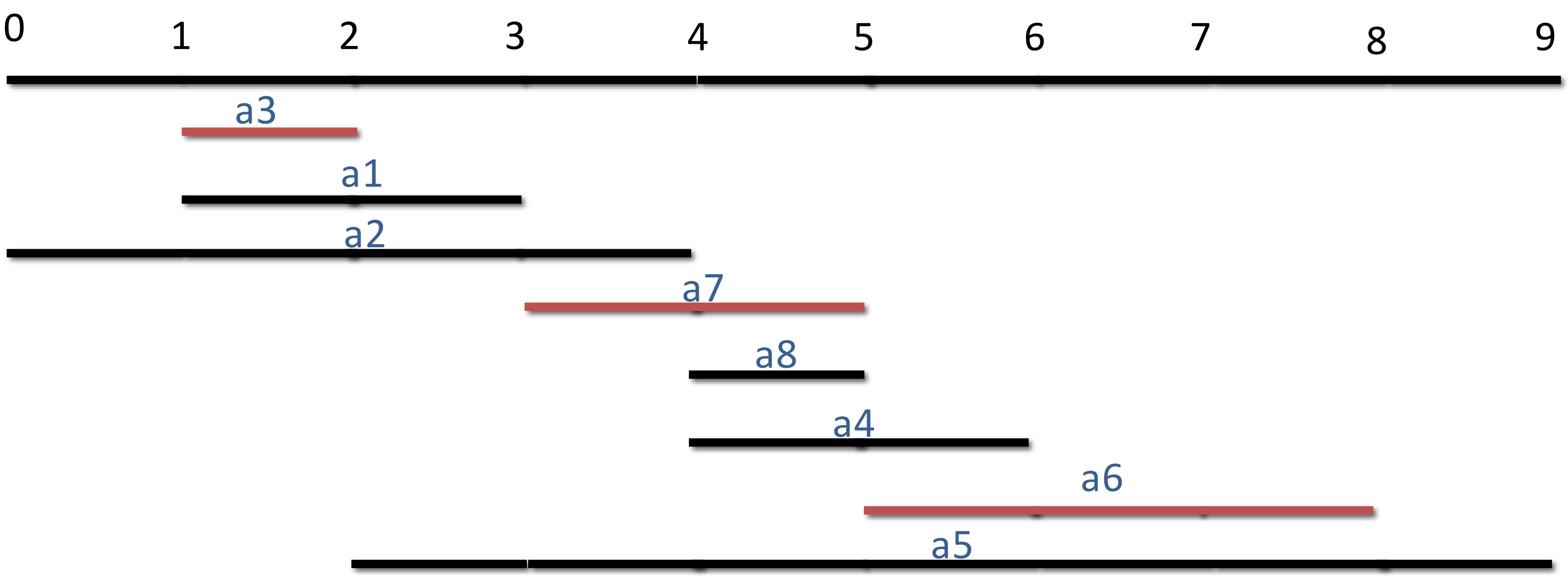
Check overlapping activities.

Sorted Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9



Check overlapping activities.

Sorted Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9



Step 2: Select the first activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

Selected activity	a3
start	1
finish	3

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

is a1 stime >= a2 ftime?

Selected activity	a3
start	1
finish	2

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

is a1 stime >= a2 ftime? **NO. Move on.**

Selected activity	a3
start	1
finish	2

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a2 stime >= a1 ftime?

Selected activity	a3
start	1
finish	2

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a2 stime >= a1 ftime? **NO. Move on.**

Selected activity	a3
start	1
finish	2

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a7 stime >= a1 ftime?

Selected activity	a3
start	1
finish	2

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a7 stime >= a1 ftime? **YES.**

Selected activity	a3
start	1
finish	2

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a7 stime >= a1 ftime? **YES.**

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

Point i to newly selected activity.

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

Move to next activity.

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a8 stime >= a7 ftime?

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

is a8 stime >= a7 ftime?

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

is a8 stime >= a7 ftime? **NO. Move on.**

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a4 stime >= a7 ftime?

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a4 stime >= a7 ftime? **NO. Move on.**

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a6 stime >= a7 ftime?

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a6 stime >= a7 ftime? **YES.**

Selected activity	a3	a7
start	1	3
finish	2	5

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i

j

Previously selected activity: i
Activity: j

is a6 stime >= a7 ftime? **YES.**

Selected activity	a3	a7	a6
start	1	3	5
finish	2	5	8

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

Point i to newly selected activity.

Selected activity	a3	a7	a6
start	1	3	5
finish	2	5	8

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

Move to the next activity.

Selected activity	a3	a7	a6
start	1	3	5
finish	2	5	8

Step 3: Select next activity whose start time is greater than or equal to the finish time of the previously selected activity

Activity	a3	a1	a2	a7	a8	a4	a6	a5
start	1	1	0	3	4	4	5	2
finish	2	3	4	5	5	6	8	9

i j

Previously selected activity: i
Activity: j

is a6 stime >= a7 ftime?

Selected activity	a3	a7	a6
start	1	3	5
finish	2	5	8

Finally! We have the required activity:

Selected activity	a3	a7	a6
start	1	3	5
finish	2	5	8

Algorithm

SGREEDY-ACTIVITY-SELECTOR (s, f)

```
    //sort S in order of increasing finishing time
    i ← 0
    X ← {A_i}
    for j ← 1 to n
        if s_j ≥ f_i
            X ← X ∪ {A_j}
            i ← j
return X
```

Analysis

- Sorting: O(n log n), For-Loop: O(n)
- Running Time: O(n log n)

Quiz/HW

No need to write your solution down.

- A = {a,b,c,d,e,f,g,h)
- S = {1,2,2,3,4,7,8,10)
- F = {6,5,8,4,8,9,10,12)

End of Lecture