

Quiz #1

**Go to gradescope and find the item
20 minutes in total**

Main purpose for quizzes

- **Help you review the course materials**
 - Repeating is helpful to form long-term memory
- **But the 3% points is not decisive**
 - We have about 10% for prog hw, 10% for written hw, 10% for exams and 10% for class participation, which is 40% in total
- **Of course, we don't require you to do them all (although it will be great if you do so)**
 - Mainly for deciding who will get A+
 - If you do all the mandatory work (a lot but not too hard), you will be fine

Midterm exam

- **Date: Oct 28**
- **Time: 6:30-8:30pm**
- **Location: likely MSE 116**
- **Will contain basic problems so if you attend all lectures and discussions and do all hws by yourselves, you should be fine**
- **Midterm challenge (tentative):**
 - Oct 28 9pm to Oct 31 (Sunday) 11:59pm
 - 4 programming problems, all require no additional knowledge from the lectures (both CS 10C and CS 141), but can be very challenging
 - Very small bonus points will be given (1pt per problem)
 - Optional to attend



Greedy Algorithms

Yan Gu



Optimization Problems & Greedy

- **Optimization Problems: A class of problems in which we are asked to**
 - Find a **set** (or a **sequence**) of “**items**”
 - That satisfy some **constraints** and simultaneously optimize (i.e., **maximize** or **minimize**) some **objective function**

Being Greedy

- **Only care about the immediate reward!**
 - When making a decision, always choose the “best” based on a certain criterion
- **May lose the overall earnings in a long-term...**
 - Greedy solution is not necessary to be optimal!
- **Sometimes greedy may also be good enough?**
 - When you can prove it!

Buying gifts: a greedy algorithm

- If possible, buy the cheapest available candy // **greedy choice**
- Repeat until no candies left or run out of money // **optimal substructure**



Prove the optimality of a greedy algorithm

- To prove optimality of a greedy strategy, we have to prove the following two properties
 1. **Greedy Choice:** The greedy choice is part of the optimal answer
 2. **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem
 - After making the first choice,
 - The final best solution is first choice + best solution for the rest of input

Buying gifts: revisit the greedy choice

- The cheapest candy is always in ONE OF the optimal solutions
 - Look at an optimal solution
 - If the cheapest candy is in, we are good
 - If not, I can always substitute any chosen candy with the cheapest one, and this is still an optimal solution!



Prove the optimality of a greedy algorithm

- To prove optimality of a greedy strategy, we have to prove the following two properties

1.  **Greedy Choice:** The greedy choice is part of the answer

2. **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem

- After making the first choice,
 - The final best solution is first choice + best solution for the rest of (compatible) input
 - We can solve the same optimization problem recursively!

Buying gifts: optimal substructure

- Global optimal solution is the cheapest candy + the optimal solutions for the subproblem ($n-1$ candies, lower budget)
 - Assume to the contrary that the optimal solution is \$1 candy + another solution



Buying gifts: optimal substructure


- **Global optimal solution is the cheapest candy + the optimal solutions for the subproblem (n-1 candies, lower budget)**
 - Assume to the contrary that the optimal solution is \$1 candy + another solution
 - Then \$1 candy + optimal solution for the rest is better!



Prove the optimality of a greedy algorithm

- To prove optimality of a greedy strategy, we have to prove the following two properties

1.  **Greedy Choice:** The greedy choice is part of the answer

2.  **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem

- After making the first choice,
 - The final best solution is first choice + best solution for the rest of input
 - We can solve the same optimization problem recursively!



Example: Kayaking!



Kayaking

- n 141 students go kayaking
- Each kayak can take 1 or 2 person(s)
 - Same price
 - Same weight limit w
- Given the weight of all students $a[i]$
- What's the smallest number of kayaks needed?



Kayaking

- $a[] = 3, 4, 6, 8, 10, 12, 15, 18$

- $w = 20$

- **Solution 1:**

- Start with the lightest student $a[1]$, pair it with the heaviest s.t. they can be in one kayak
- (3, 15)
- (4, 12)
- (6, 10)
- (8)
- (18)
- 5 in total

Kayaking

- $a[] = 3, 4, 6, 8, 10, 12, 15, 18$

- $w = 20$

- **Solution 2:**

- Start with the heaviest student $a[n]$, pair it with the heavies s.t. they can be in one kayak
- (18)
- (15, 4)
- (12, 8)
- (10, 6)
- (3)
- 5 in total

Kayaking

- **Actually, both will give you an optimal solution!**
- **Why????**
- **Take solution 1 as an example**
 - Start with the lightest student $a[1]$, pair it with the heaviest s.t. they can be in one kayak

Prove the optimality of a greedy algorithm

- To prove optimality of a greedy strategy, we have to prove the following two properties
 1. **Greedy Choice:** The greedy choice is part of the answer
 2. **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem
 - After making the first choice,
 - The final best solution is first choice + best solution for the rest of (compatible) input
 - We can solve the same optimization problem recursively!

Kayaking: The greedy choice is part of the answer

- Pair the lightest student A_1 with the heaviest student (say X) s.t. they can be in one kayak
- Is (A_1, X) always part of SOME optimal solutions?
- $a[] = 3, 4, 6, 8, 10, 12, 15, 18$ $w = 20$
 - Should we pair $(3, 15)$? Is it contained in SOME optimal solution?
 - Look at some optimal solution
 - If $(3, 15)$ is in the solution, we are good!
 - If not, 3 can be paired with someone else and 15 is paired with someone else?
 - $(3, 12), (15, 4), (6, 8), (18), (10)$
 - We can always **switch 3's partner with 15**! $(3, 15), (12, 4)$ must also be valid! (why?)
 - 3 or 15 can also be in a single kayak, but that's an easy case

Kayaking: The greedy choice is part of the answer

- Pair the lightest student A_1 with the heaviest student (say X) s.t. they can be in one kayak
- Is (A_1, X) always part of SOME optimal solutions?
- Look at some optimal solution
 - If (A_1, X) is in the solution, we are good!
 - If not, A_1 can be paired with someone else and X is paired with someone else?
 - $(A_1, Y), (X, Z)$
 - We can always switch A_1 's partner with X ! $(A_1, X), (Y, Z)$ must also be valid!
 - Why? **We already know $A_1 + X \leq w$**
 - $Y \leq X$ because X is the “heaviest one that can be in the same kayak as A_1 ”
 - **$Y + Z \leq X + Z \leq w$, so pairing Y and Z is valid!**
 - X and A_1 can also be in a single kayak but that's an easy case
- **So it's OK to pair A_1 with X ! Even if not, we just change them to the same kayak – it does not affect the optimality of the solution**

Kayaking: greedy choice

- The greedy choice is part of the answer
- Or ..., if we cannot find anyone to pair with A_1 , then the kayak (A_1) must be in the optimal solution

Prove the optimality of a greedy algorithm

- To prove optimality of a greedy strategy, we have to prove the following two properties

1.  **Greedy Choice:** The greedy choice is part of the answer

2. **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem

- After making the first choice,
 - The final best solution is first choice + best solution for the rest of (compatible) input
 - We can solve the same optimization problem recursively!


Kayaking: optimal substructure

- **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem
- After we pair A_1 with X , what happens?
- **Claim:** the optimal solution of the problem with choosing (A_1, X) is
 - (A_1, X) + optimal solution of assigning kayaks to the rest $n-2$ students
 - Why? Assume to the contrary it is not, it is (A_1, X) + solution B , where B is worse than the “optimal solution of assigning boats to the rest $n-2$ students”.
 - Why don't we substitute B with the “optimal solution of assigning boats to the rest $n-2$ students”?
- **Similar for the case when we cannot pair A_1 with anyone**

Prove the optimality of a greedy algorithm

- To prove optimality of a greedy strategy, we have to prove the following two properties

1.  **Greedy Choice:** The greedy choice is part of the answer

2.  **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem

- After making the first choice,
 - The final best solution is first choice + best solution for the rest of (compatible) input
 - We can solve the same optimization problem recursively!

Solution 1 will always be optimal

- How about Solution 2?
- You'll prove in your homework 2!

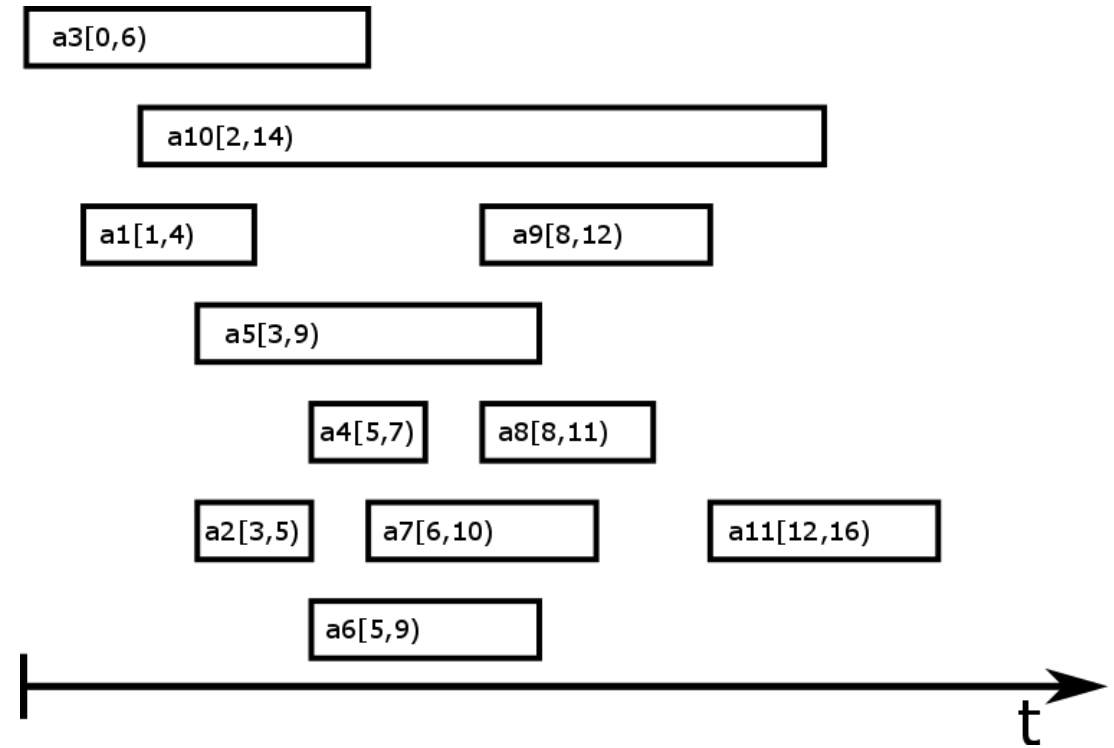
Activity Selection

a.k.a. Task Scheduling

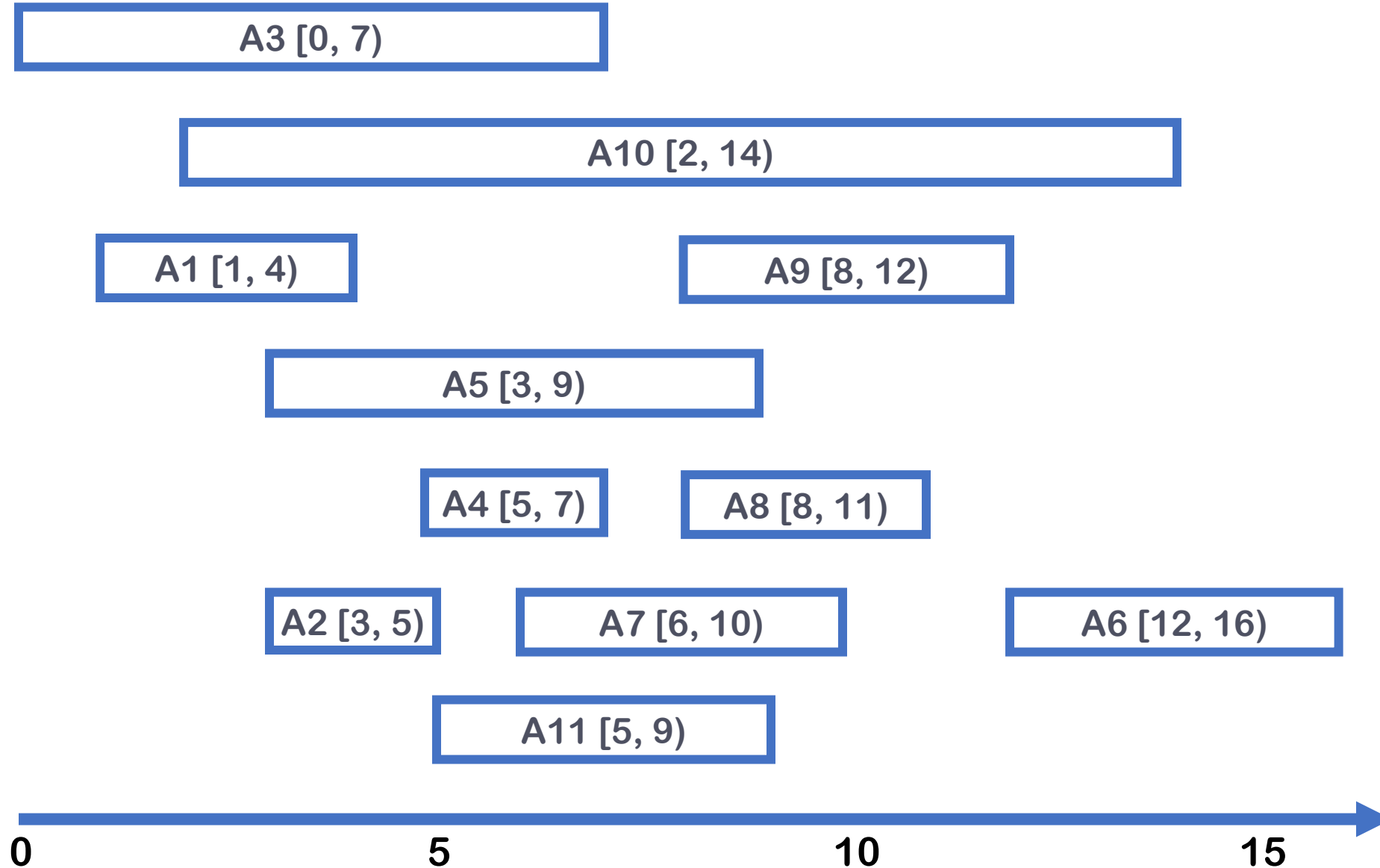
CLRS 16.1

Activity Selection Problem

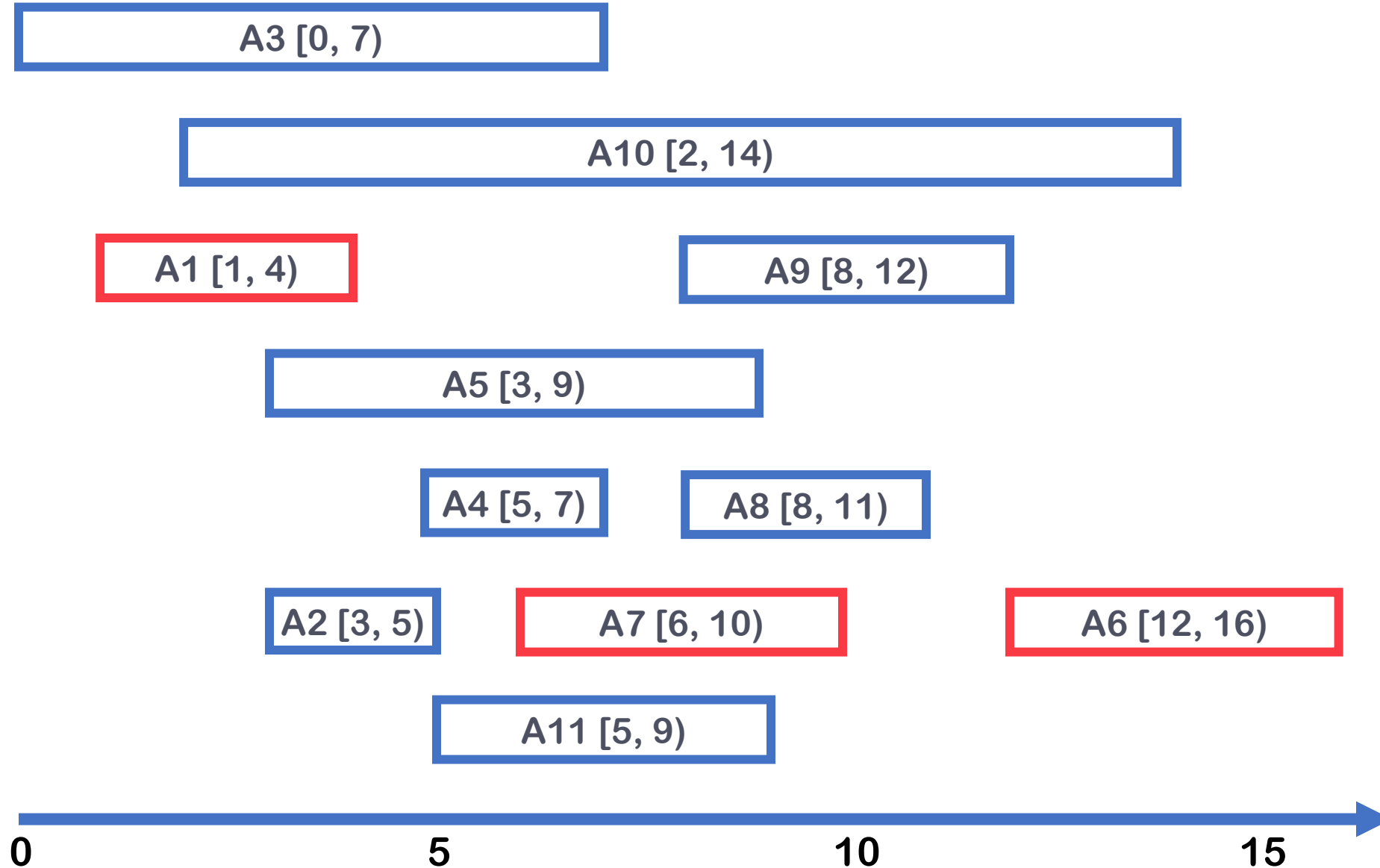
- Given a set of activities $S = \{a_1, a_2, \dots, a_n\}$ where
 - Each activity i has a start time s_i and a finish time f_i , where $0 \leq s_i < f_i < \infty$.
 - An activity a_i happens in the half-open time interval $[s_i, f_i)$.
 - Two activities are said to be **compatible** if they **do not overlap**.
- The problem is to find a maximum-size compatible subset, i.e., a one with the **maximum number of activities**.



Example of Activity Selection

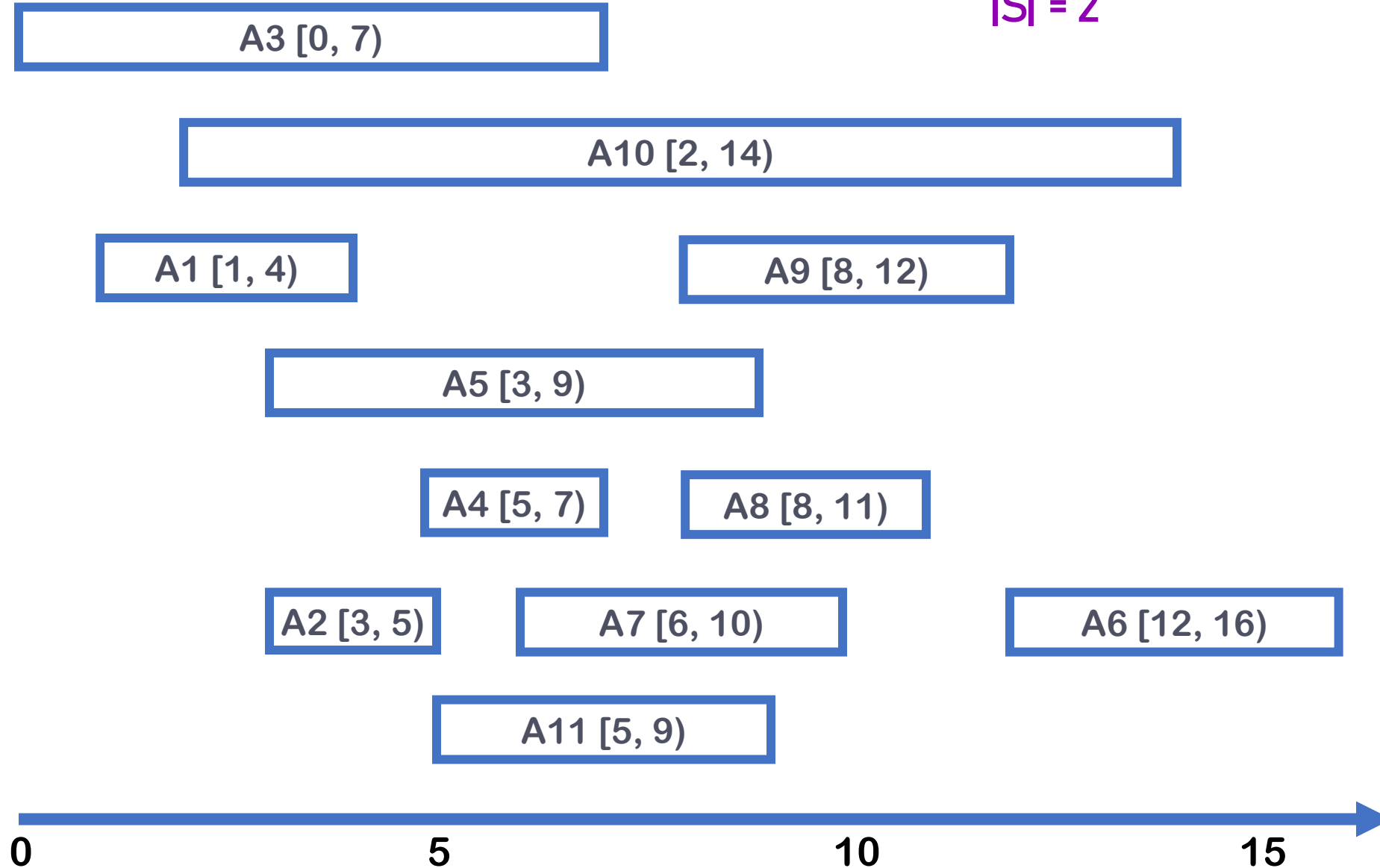


One solution: 3 activities



Strategy 1: random pick

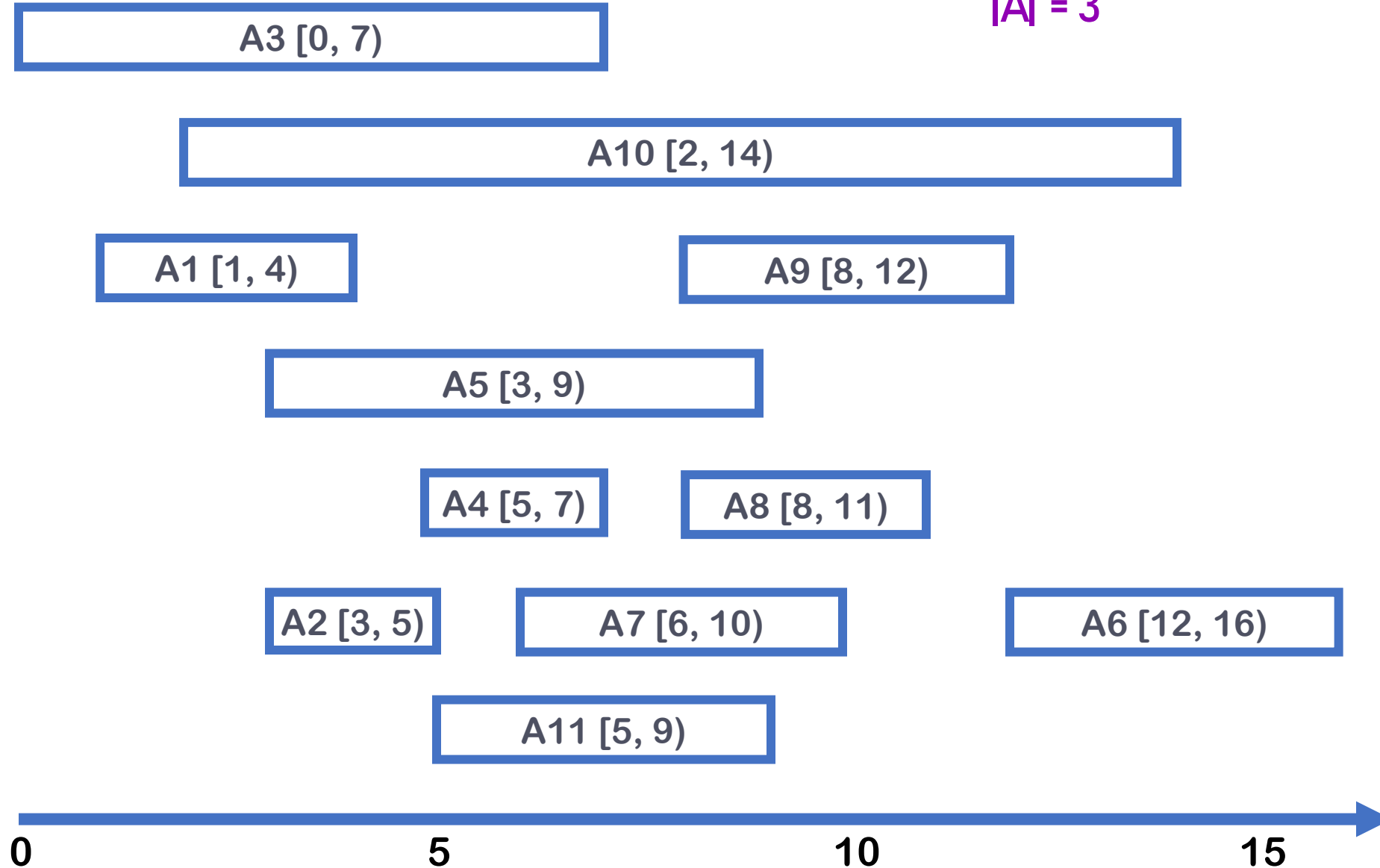
Solution: $S = \{A5, A6\}$
 $|S| = 2$



Strategy 2: earliest start first

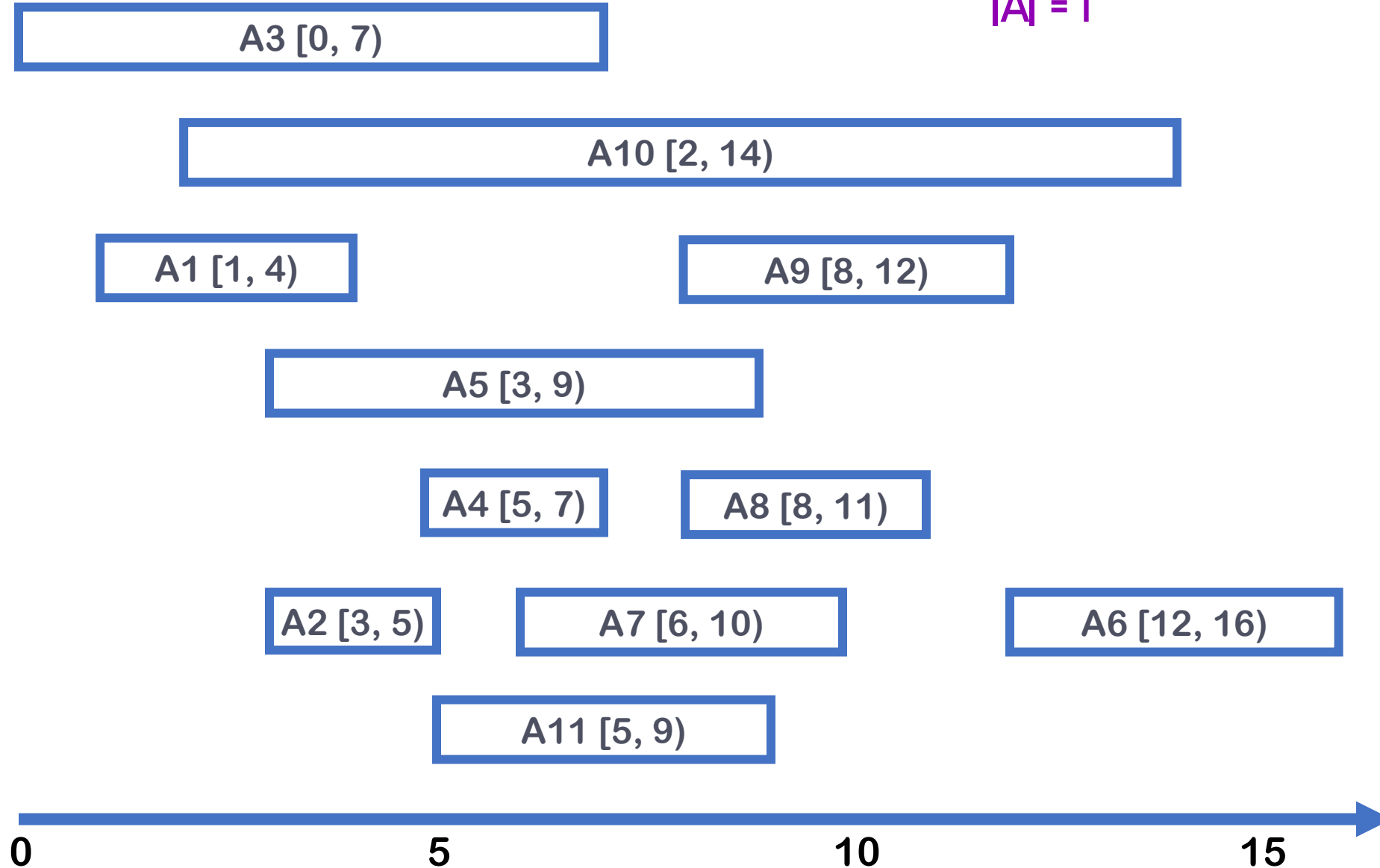
Solution: $A = \{A3, A9, A6\}$

$|A| = 3$



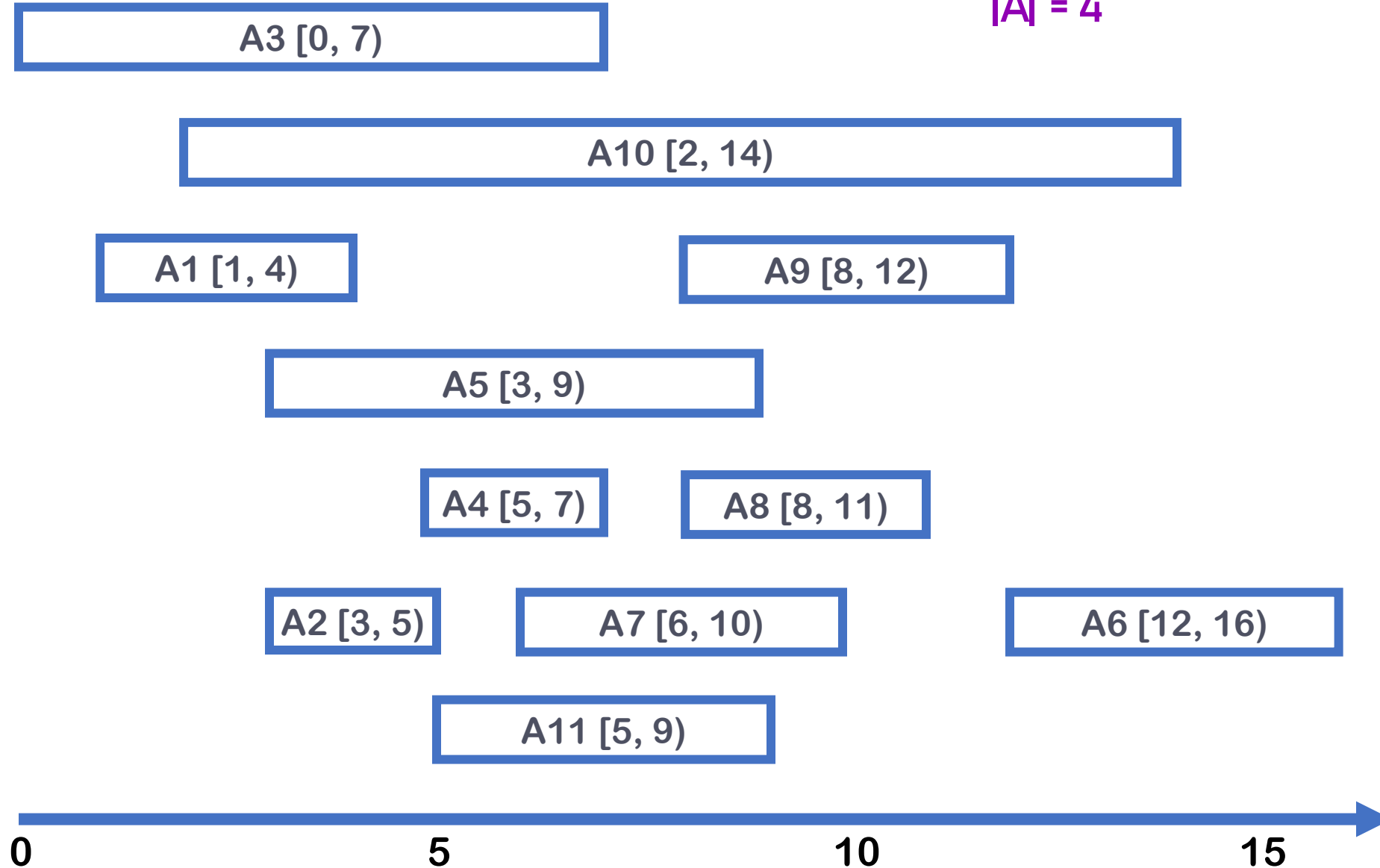
Strategy 3: largest first

Solution: $A = \{A_{10}\}$
 $|A| = 1$



Strategy 4: smallest first

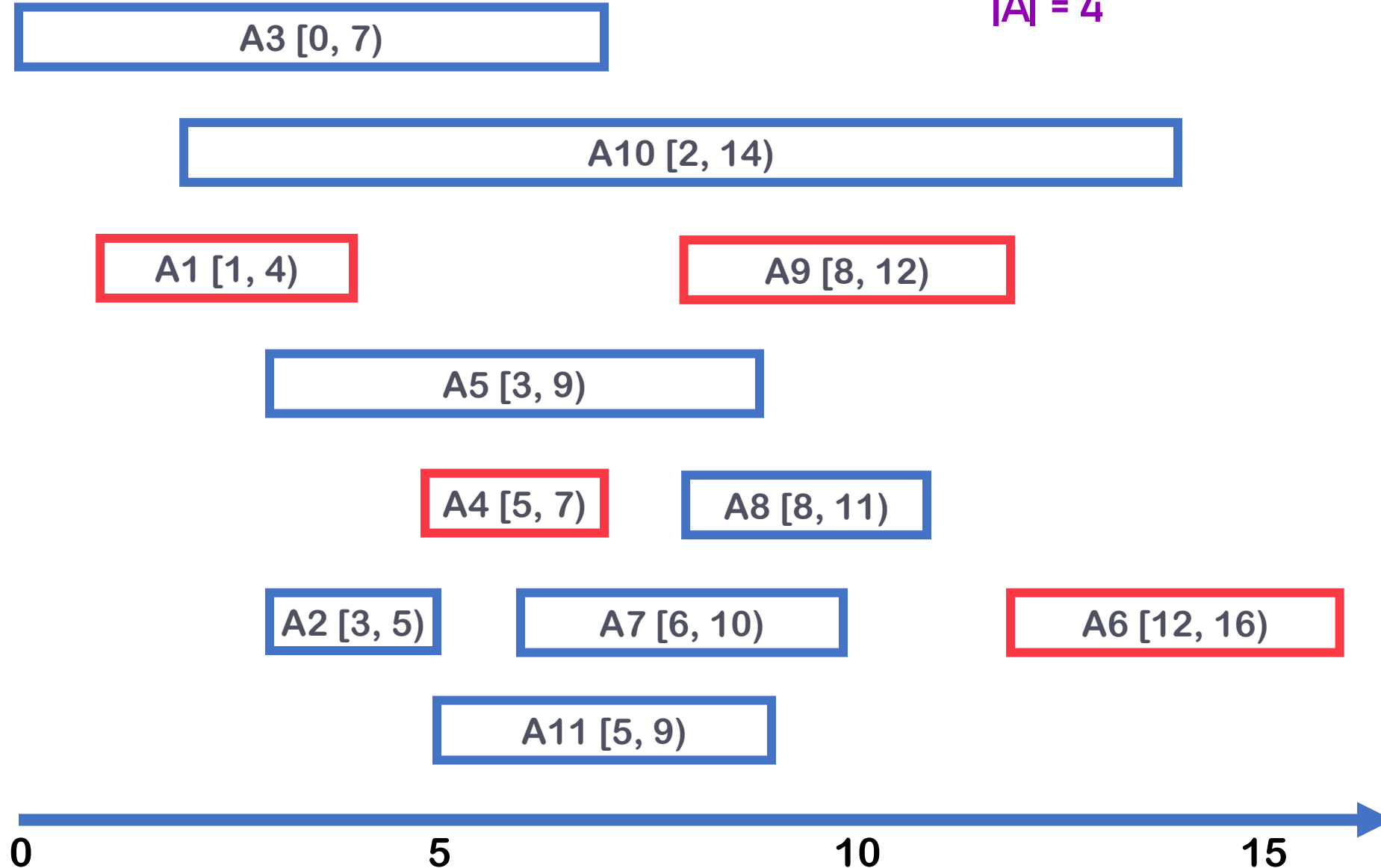
Solution: $A = \{A_4, A_2, A_8, A_6\}$
 $|A| = 4$



Another solution: 4 activities

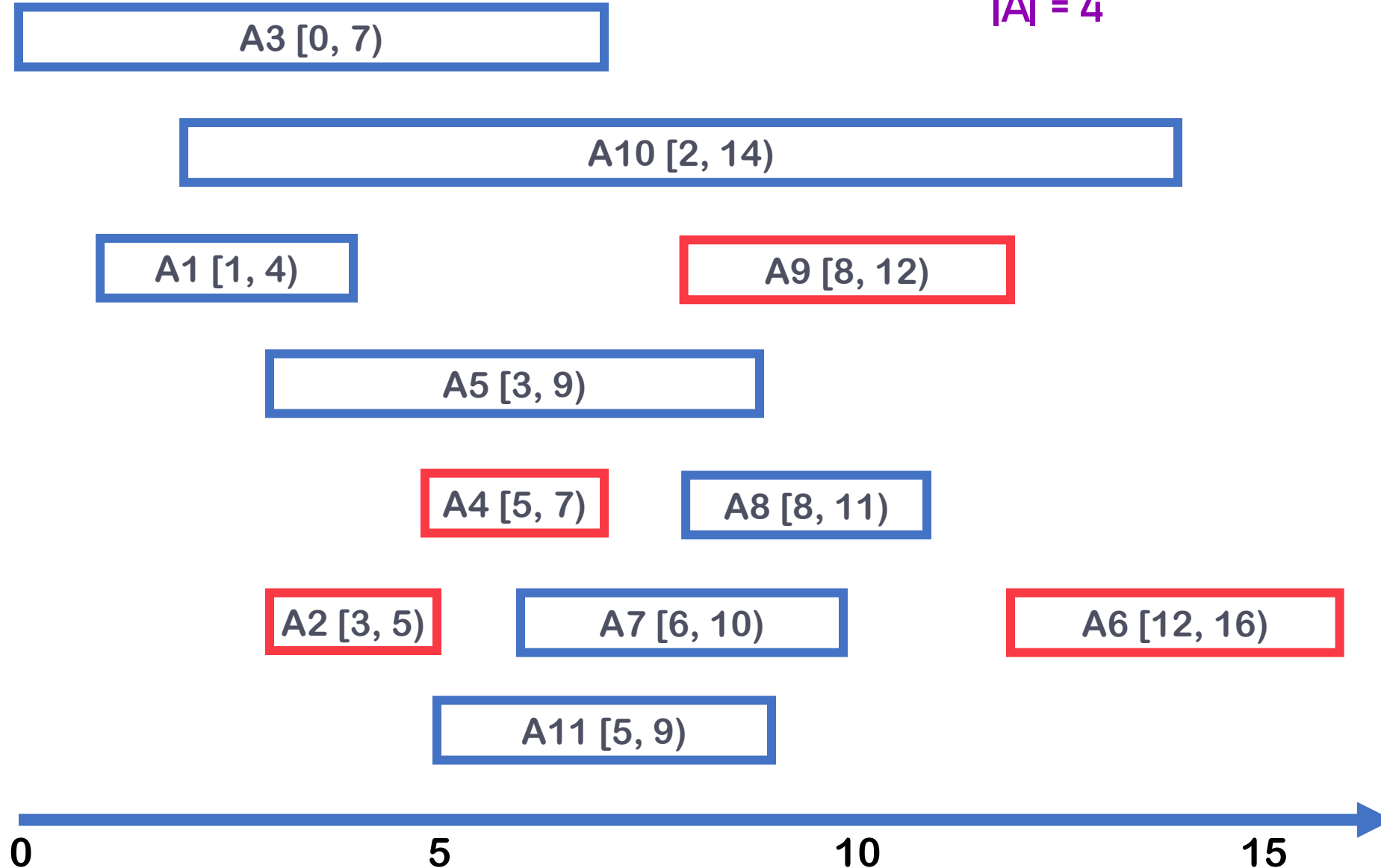
Solution: $A = \{A1, A4, A9, A6\}$

$|A| = 4$



Another solution: 4 activities

Solution: $A = \{A_2, A_4, A_8, A_6\}$
 $|A| = 4$



Example of Activity Selection

- In the previous example, actually we cannot pick more than 4 activities
- Does that mean smallest first can get the optimal solution?

A1 [2, 6)

A3 [7, 13)

A2 [5, 8)

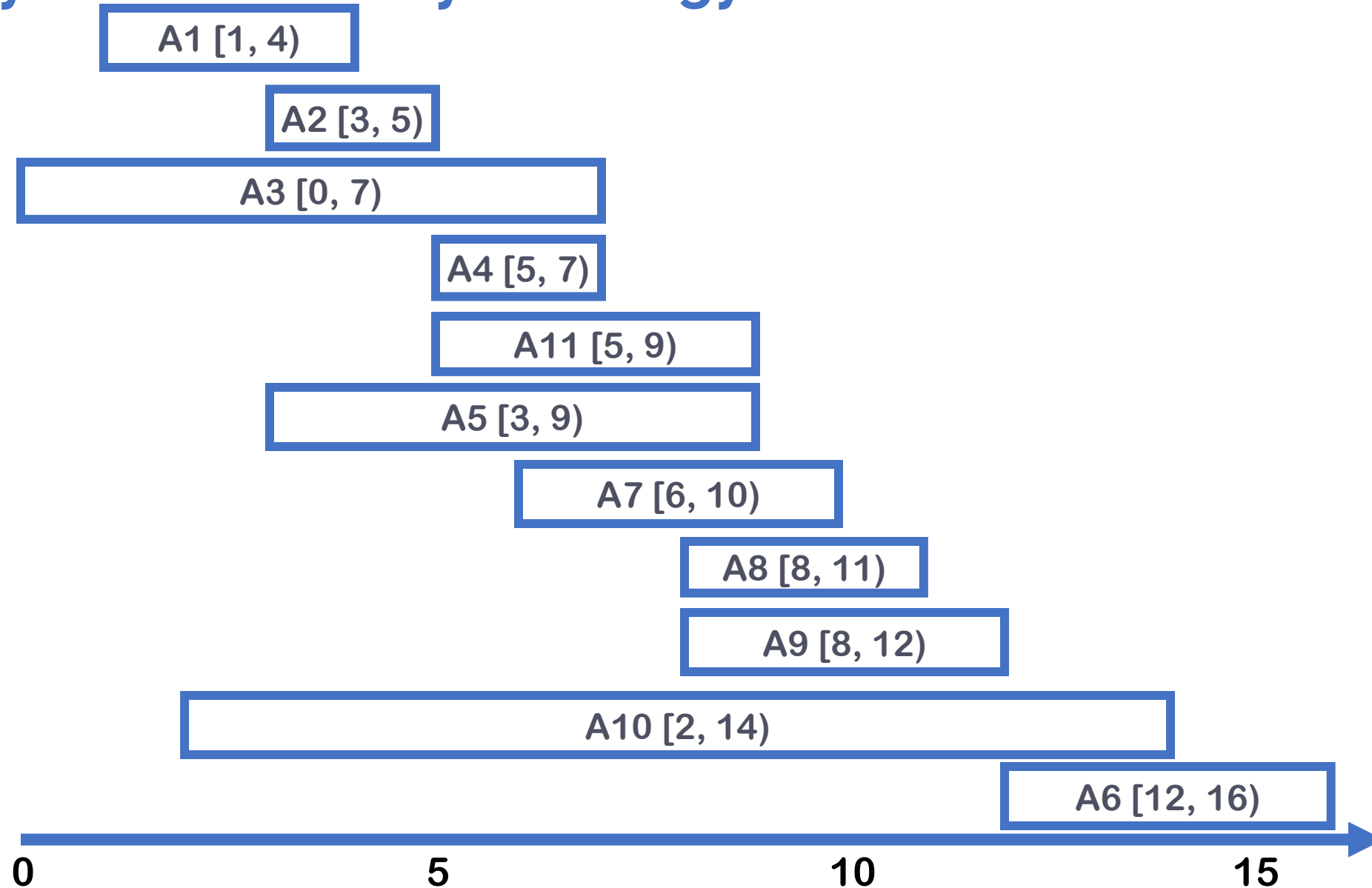
Smallest first: 1 activity
Optimal solution: 2 activities

Early Finish Greedy Strategy – Iterative solution

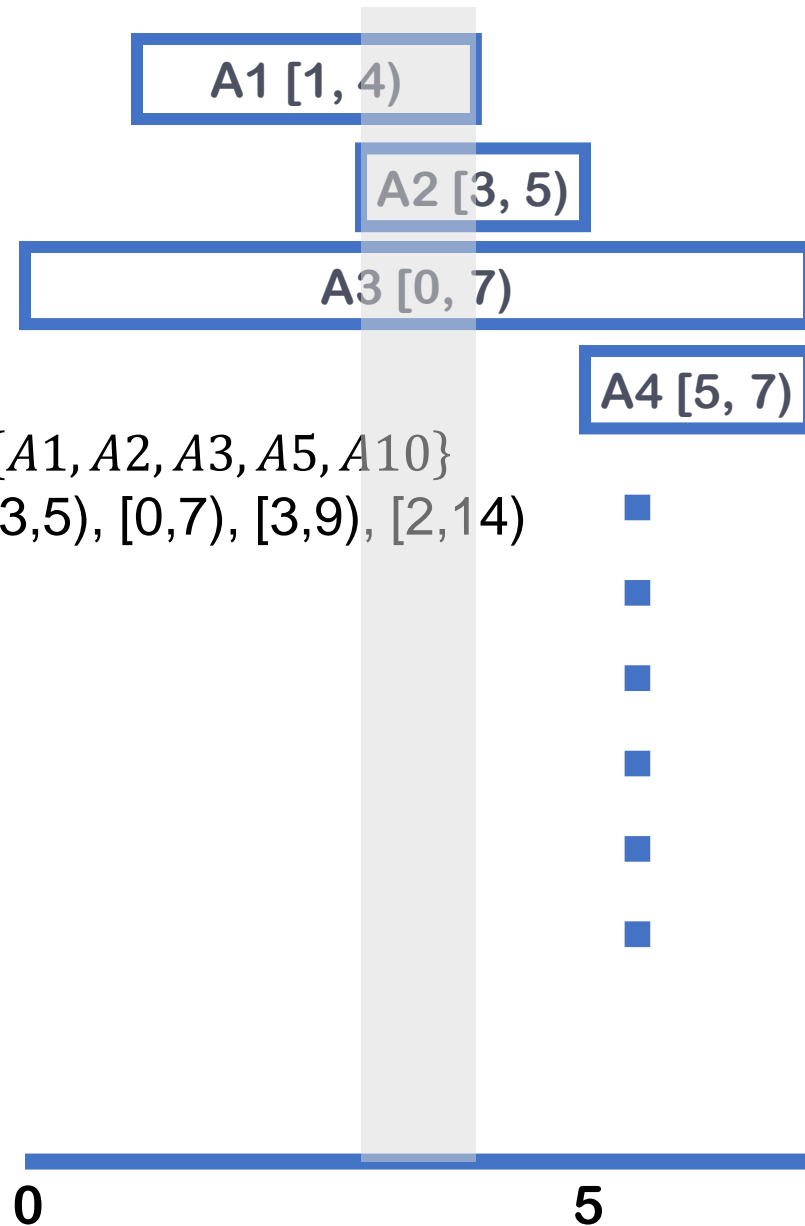
activity_selection (S: a set of activities)

- 1. Schedule the earliest-finish activity $a_1 \in S$**
- 2. Remove all incompatible activities from S**
- 3. If there are more activities, repeat 1-2**

Early Finish Greedy Strategy



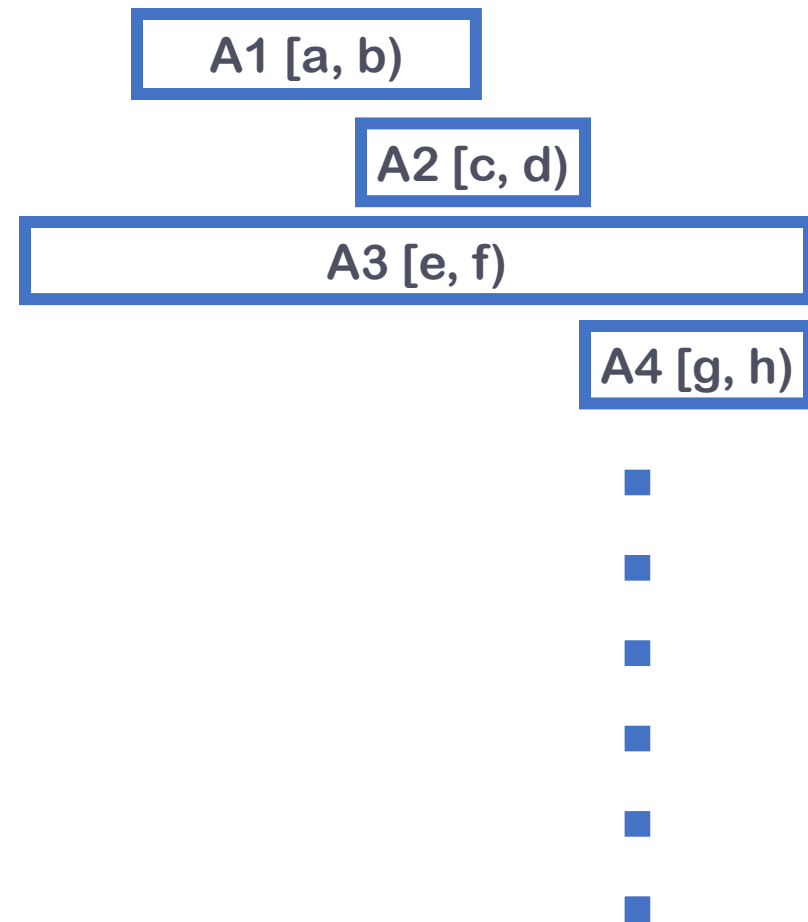
Why it's optimal?



Let's take a look at the **earliest finish activity [1, 4)**:

- It overlaps with some other activities
- Let T be the set of all activities overlapping with [1, 4), they **must all contain** interval [3, 4) (why?)
 - All of them ends ≥ 4 : otherwise [1, 4) is not the earliest finish
 - All of them must start ≤ 3 : otherwise they don't overlap with [1, 4)
- So **at most one of them** could be selected!
 - Then why don't **select [1, 4)**: disables no more other activities
- If we don't select any of them, [1, 4) will always be vacant, so **we can afford select [1, 4) at last**
- **So choosing [1, 4) is always good!**
- For the rest steps, the same claim holds – we can inductively prove the optimality

Why it's optimal?



Let's take a look at the **earliest finish activity** [a, b):

- It overlaps with some other activities
- Let T be the set of all activities overlapping with [a, b), they **must all contain** interval [b-1, b) (why?)
 - All of them ends $\geq b$: otherwise [a, b) is not the earliest finish
 - All of them must start $\leq b-1$: otherwise they don't overlap with [a, b)
- So **at most one of them** could be selected!
 - Then why don't select [a, b): disables no more other activities
- If we don't select any of them, [a, b) will always be vacant, so **we can afford select [a, b) at last**
- **So choosing [a, b) is always good!**
- For the rest steps, the same claim holds – we can use induction to prove the optimality

How to prove the optimality of a greedy algorithm in general?

CLRS 16.2

Prove the optimality of a greedy algorithm

- To prove optimality of a greedy strategy, we have to prove the following two properties
 1. **Greedy Choice:** The greedy choice is part of the answer
 2. **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem
 - After making the first choice,
 - The final best solution is first choice + best solution for the rest of (compatible) input

Prove the optimality of a greedy algorithm: activity selection

1. **Greedy Choice:** The greedy choice is part of the answer

- The earliest finish activity t is part of some optimal solution

2. **Optimal Substructure:** The optimal solution to the big problem contains the optimal solution to the sub-problem

- Best solution with a is $\{a\} \cup$ “the best solution of Input — {activities incompatible with a }”

Greedy Choice

The earliest finish task t is part of some optimal solution

- Assume the earliest finish task in input is t
- We want to prove that t is part of an optimal solution
 - Choosing t is **always** “good”!
- Look at an optimal solution $A = \{a_1, \dots, a_k\}$ sorted by end time
- (case 1) If $a_1 = t$ then we are done
- (case 2) Otherwise, we prove that there **exists** another optimal solution $A' = A - \{a_1\} \cup \{t\}$
 - Although t may not be in A , we can **construct** another valid solution with t that is **as good as** A !

Greedy Choice

The earliest finish task a_m is part of some optimal solution

- Look at an optimal solution $A = \{a_1, \dots, a_k\}$ sorted by end time
- Let t be the earliest finish in input
 - Input = $[1,4), [3,5), [0,7), [5,7), [5,9), [3,9), [6,10)\dots$
 - $A = [1,4), [5,7), [8,12), [12,16)$ Then $a_1 = [1,4), t = [1,4)$
 - $A = [3,5), [5,7), [8,11), [12,16)$ Then $a_1 = [3,5), t = [1,4)$
- We want to prove that t is part of an optimal solution
- (case 1) If $t = a_1$ then we are done

Greedy Choice

The earliest finish task a_m is part of some optimal solution

- Look at an optimal solution $A = \{a_1, \dots, a_k\}$ sorted by end time
- Let t be the earliest finish in input
 - Input = $[1,4), [3,5), [0,7), [5,7), [5,9), [3,9), [6,10)\dots$
 - $A = [1,4), [5,7), [8,12), [12,16)$ Then $a_1 = [1,4), t = [1,4)$
 - $A = [3,5), [5,7), [8,11), [12,16)$ Then $a_1 = [3,5), t = [1,4)$
- We want to prove that t is part of an optimal solution
- (case 1) If $t = a_1$ then we are done
- (case 2) If $a_1 \neq t$, we prove that there is another optimal solution $A' = A - \{a_1\} \cup \{t\}$

Greedy Choice

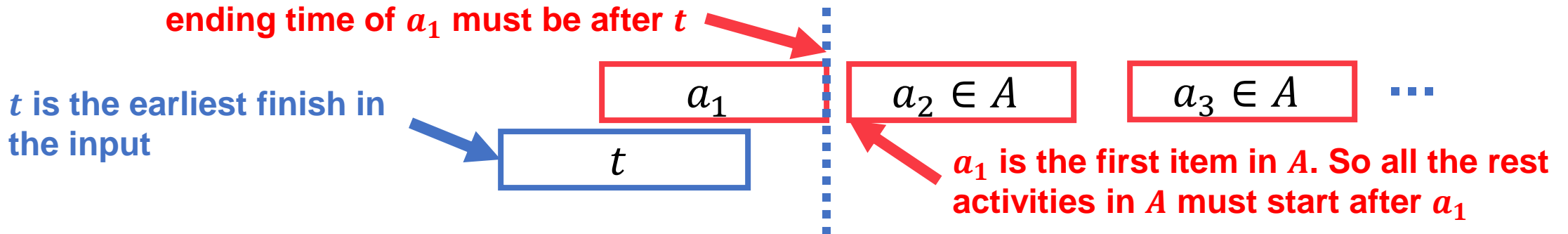
The earliest finish task a_m is part of some optimal solution

- Look at an optimal solution $A = \{a_1, \dots, a_k\}$ sorted by end time
- Let t be the earliest finish in input
 - Input = $[1,4), [3,5), [0,7), [5,7), [5,9), [3,9), [6,10)\dots$
 - $A = [3,5), [5,7), [8,11), [12,16)$ Then $a_1 = [3,5), t = [1,4)$
- We want to prove that t is part of an optimal solution
- (case 2) If $a_1 \neq t$, we prove that there is another optimal solution $A' = A - \{a_1\} \cup \{t\}$
 - $A = [3,5), [5,7), [8,11), [12, 16)$ Then $a_j = [3,5), a_m = [1,4)$
 - Replace $[3,5)$ with $[1,4)$: $A' = [1,4), [5,7), [8,11), [12, 16)$
 - We need to show: It's also valid, and it's also optimal

Greedy Choice

The earliest finish task a_m is part of some optimal solution

- **(case 2) If $a_1 \neq t$, we prove that there is another optimal solution $A' = A - \{a_1\} \cup \{t\}$**
 - $A = [3,5), [5,7), [8,11), [12, 16)$ Then $a_j = [3,5), a_m = [1,4)$
 - Replace $[3,5)$ with $[1,4)$: $A' = [1,4), [5,7), [8,11), [12, 16)$
 - Is A' a solution? Yes! All other tasks in A are still compatible with t !
 - t is the earliest finish in the input
 - So a_1 ends after t
 - a_1 is the earliest finish in A , so all other tasks in A start and finish after a_1 finish
 - **So all other tasks also don't overlap with t . It is safe to replace a_1 with t**



Greedy Choice

The earliest finish task a_m is part of some optimal solution

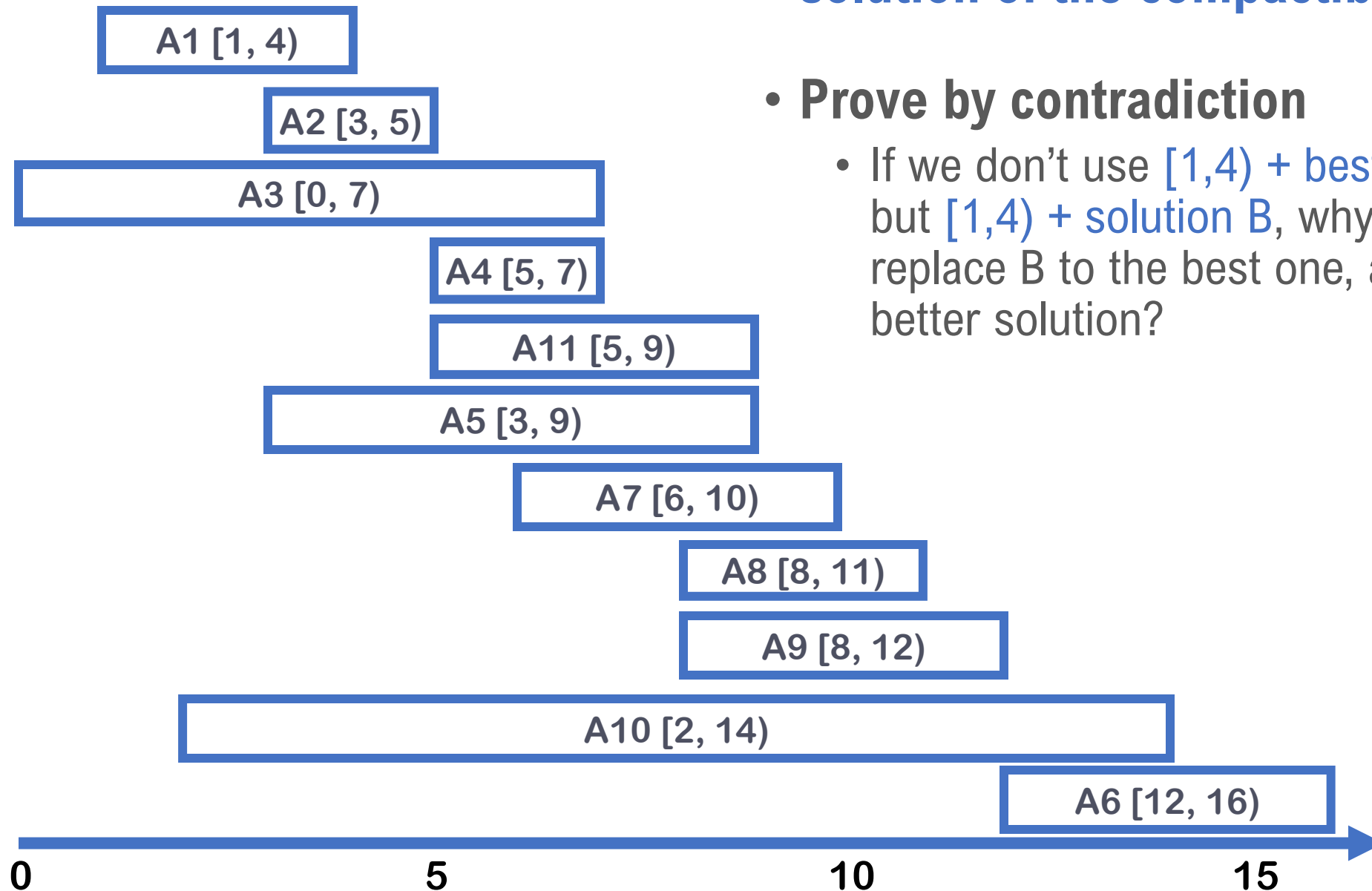
- Assume the earliest finish task in input is t
- We want to prove that t is part of an optimal solution
 - Choosing t is **always “good”!**
- Look at an optimal solution $A = \{a_1, \dots, a_k\}$ sorted by end time
- (case 1) If $a_1 = t$ then we are done
- (case 2) Otherwise, we prove that there **exists** another optimal solution $A' = A - \{a_1\} \cup \{t\}$
 - Is A' a solution? **Yes! t is compatible with all the other activities in A**
 - Is A' optimal? **Yes! The size is the same with A**

Prove the optimality of a greedy algorithm: activity selection

 1. **Greedy Choice:** The earliest finish task t is part of some optimal solution

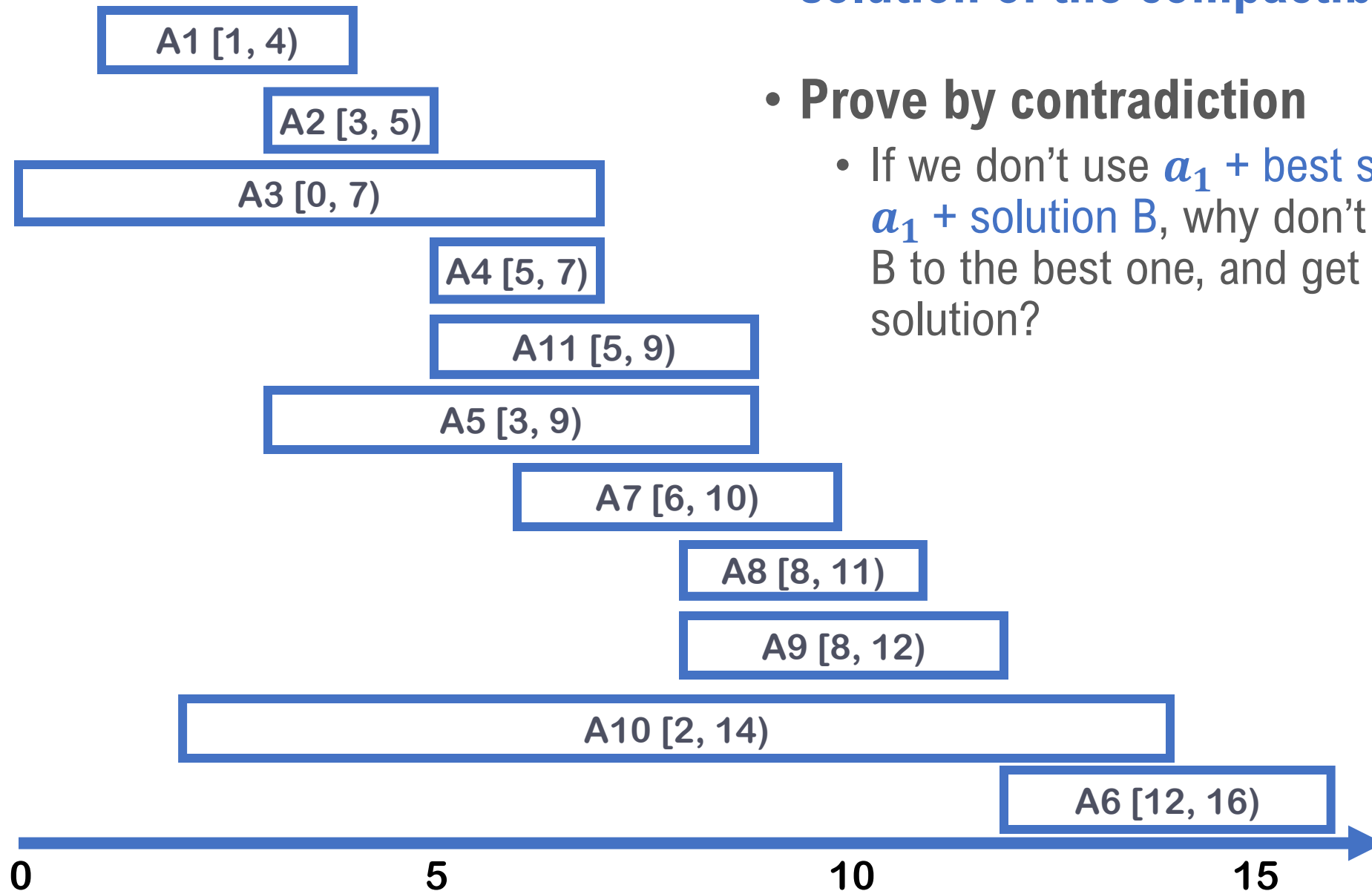
2. **Optimal Substructure:** optimal solution $A = \{a, \dots\}$ without a is “the best solution of input – {activities incompatible with a }”

Optimal Substructure





- The best solution is **[1,4) + best solution of the compactible ones**
- **Prove by contradiction**
 - If we don't use **[1,4) + best solution** but **[1,4) + solution B**, why don't we replace B to the best one, and get a better solution?

Optimal Substructure



- The best solution is a_1 + best solution of the compactible ones
- Prove by contradiction
 - If we don't use a_1 + best solution but a_1 + solution B, why don't we replace B to the best one, and get a better solution?

Prove the optimality of a greedy algorithm: activity selection

-  1. **Greedy Choice:** The earliest finish task t is part of some optimal solution
-  2. **Optimal Substructure:** optimal solution $A = \{a, \dots\}$ without a is “the best solution of input – {activities incompatible with a }”

Optimal substructure

Optimal Substructure: The optimal solution to the big problem contains the optimal solution to the sub-problem

- After choosing the greedy choice, we just call the same greedy algorithm on the rest of the (compatible) input and repeat

(not just for greedy algorithms, we'll see the concept again in dynamic programming)

- After we make the first decision, the best solution is to solve the same optimization problem on a smaller size

What do greedy choice and optimal substructure mean?

- **Greedy choice (intuitively):**

- The element t you greedily choose is not a bad idea! It appears in some optimal solution
- (for any optimal solution, if it doesn't contain t , we can modify it to contain t !)
- So just choose it!

- **Optimal substructure (intuitively):**

- After choosing some element t
- The final optimal solution is just to find the optimal solution for the rest of the (compatible) elements!
- Recursively solve it using the same approach

- **So we repeatedly choose the greedy choice!**

Greedy algorithms

- **What we have already covered**

- Buying gifts
- Kayaking
- Activity selection

- **The next lecture: Huffman code**

- **Designing greedy algorithms: Prog HW2 Prob B, C, D, Written HW2 Prob B**
- **Proving optimality: Written HW2 Problem A and B**