CS218: design and analysis of algorithms

Greedy Algorithms

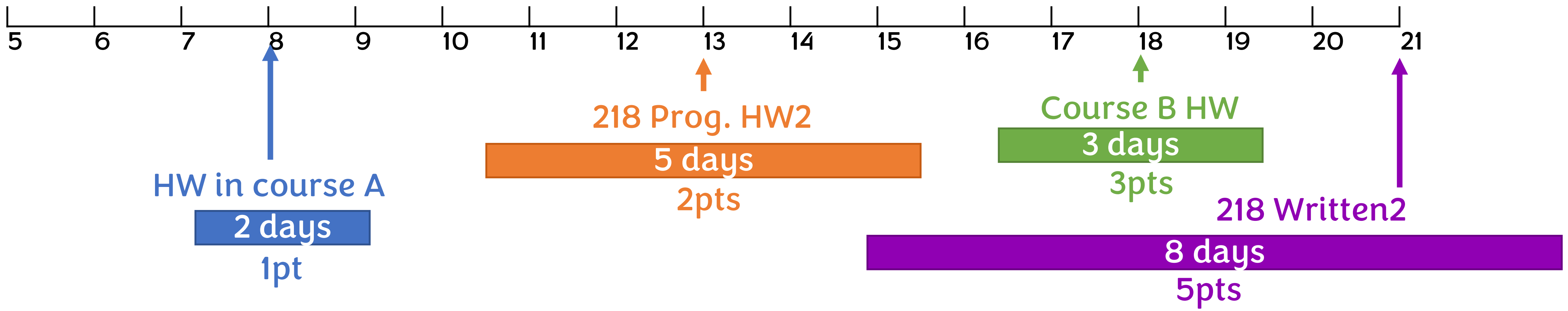
Yan Gu

How to be greedy?

- **Only care about the immediate reward for any decision make!**
- **I have a few homework assignments to do, which one should I start first?**
  - (For simplicity, we assume you can always get full score using a certain time)
  - A. Work on the one with the earliest deadline!
  - B. Work on the one that worth the highest points!
  - C. Work on the easiest one that requires the least time!
  - D. Work on the hardest one that requires the most time!
  - E. I roll the dice

- **Which one do you like most?**

A. Deadline First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

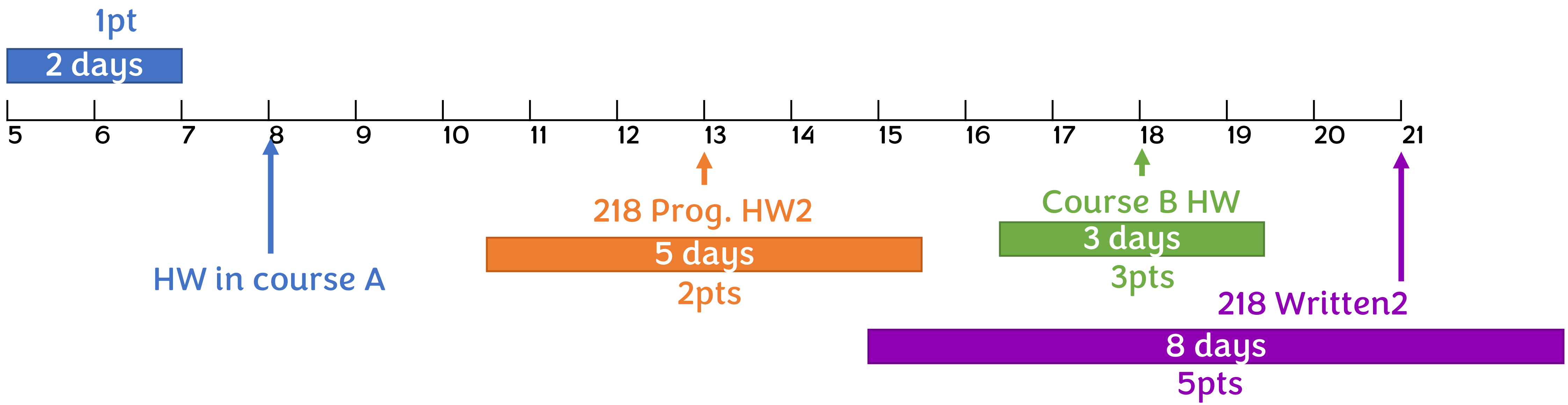
218 written II
Due 21st
5 points
Takes 8 days

HW in course A
Due 8th
1 point
Takes 2 days

HW in course B
Due 18th
3 points
Takes 3 days

A. Deadline first

A. Deadline First Strategy



218 programming II
Due 13^{th}
2 points
Takes 5 days

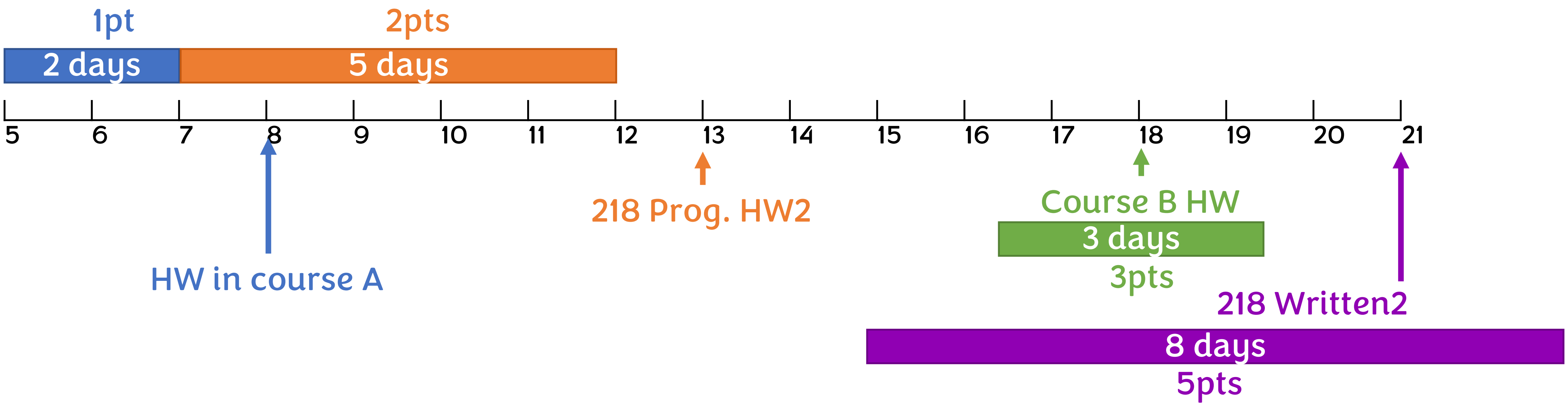
218 written II
Due 21^{st}
5 points
Takes 8 days

HW in course A
Due 8^{th}
1 point
Takes 2 days

HW in course B
Due 18^{th}
3 points
Takes 3 days

A. Deadline first

A. Deadline First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

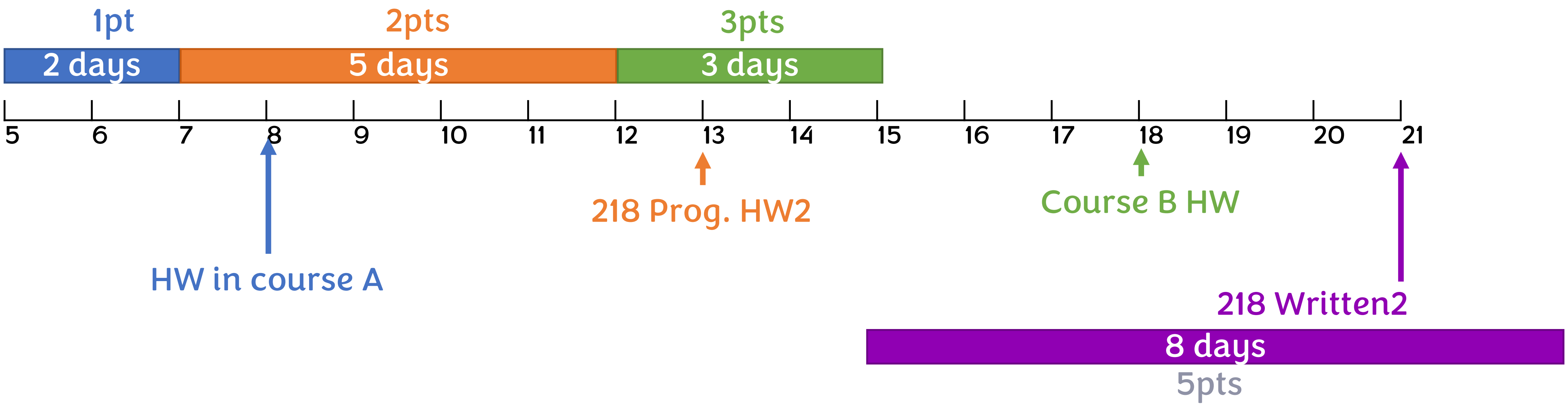
218 written II
Due 21st
5 points
Takes 8 days

HW in course A
Due 8th
1 point
Takes 2 days

HW in course B
Due 18th
3 points
Takes 3 days

A. Deadline first

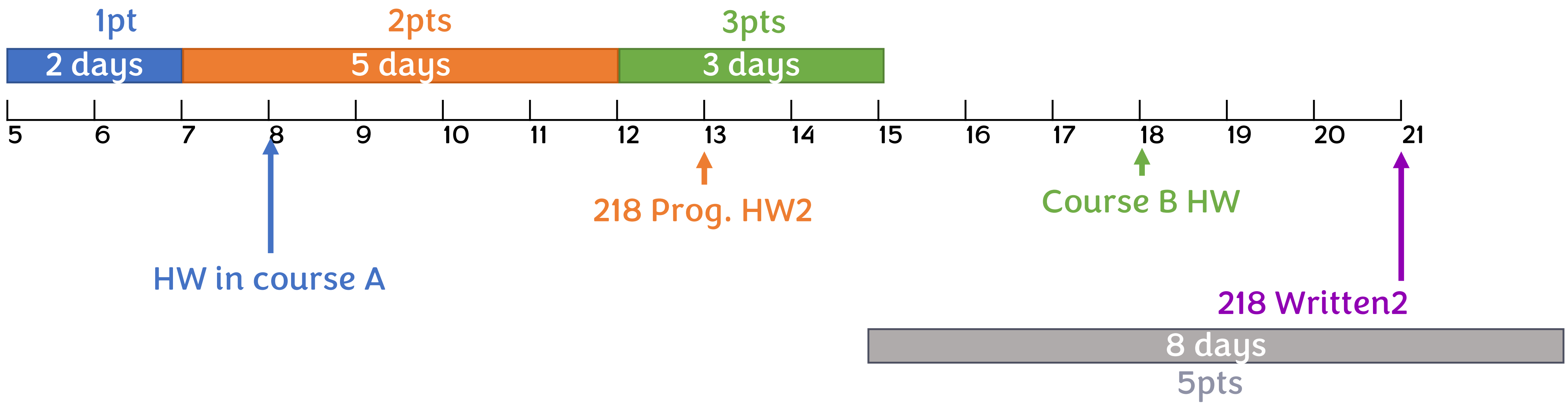
A. Deadline First Strategy



218 programming II Due 13th 2 points Takes 5 days	218 written II Due 21st 5 points Takes 8 days	HW in course A Due 8th 1 point Takes 2 days	HW in course B Due 18th 3 points Takes 3 days

A. Deadline first

A. Deadline First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

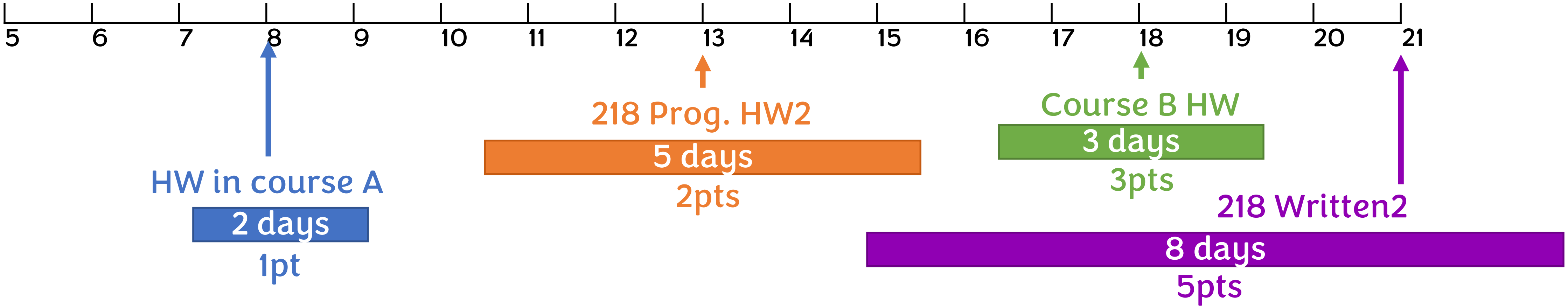
218 written II
Due 21st
5 points
Takes 8 days

HW in course A
Due 8th
1 point
Takes 2 days

HW in course B
Due 18th
3 points
Takes 3 days

A. Deadline first **6pts**

B. Highest Score First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

218 written II
Due 21st
5 points
Takes 8 days

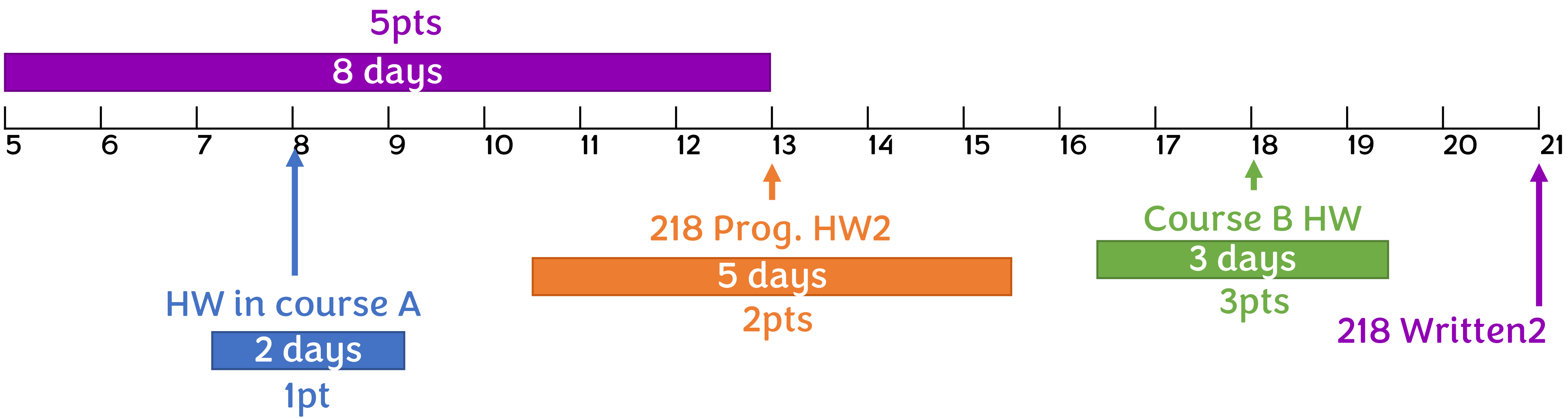
HW in course A
Due 8th
1 point
Takes 2 days

HW in course B
Due 18th
3 points
Takes 3 days

A. Deadline first **6pts**

B. Highest score first

B. Highest Score First Strategy

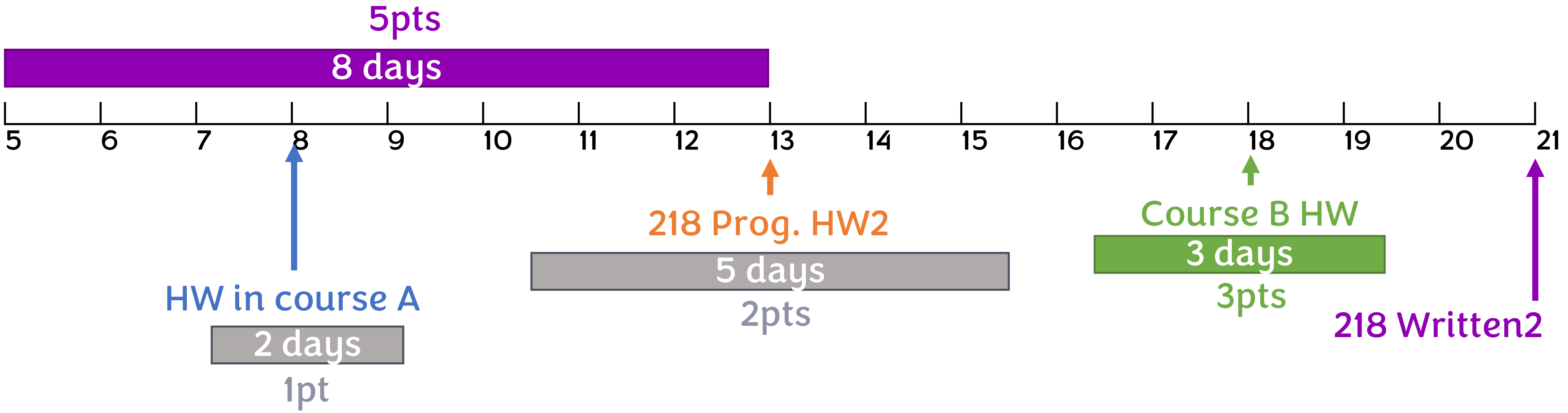


218 programming II Due 13th 2 points Takes 5 days	218 written II Due 21st 5 points Takes 8 days	HW in course A Due 8th 1 point Takes 2 days	HW in course B Due 18th 3 points Takes 3 days

A. Deadline first **6pts**

B. Highest score first

B. Highest Score First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

218 written II
Due 21st
5 points
Takes 8 days

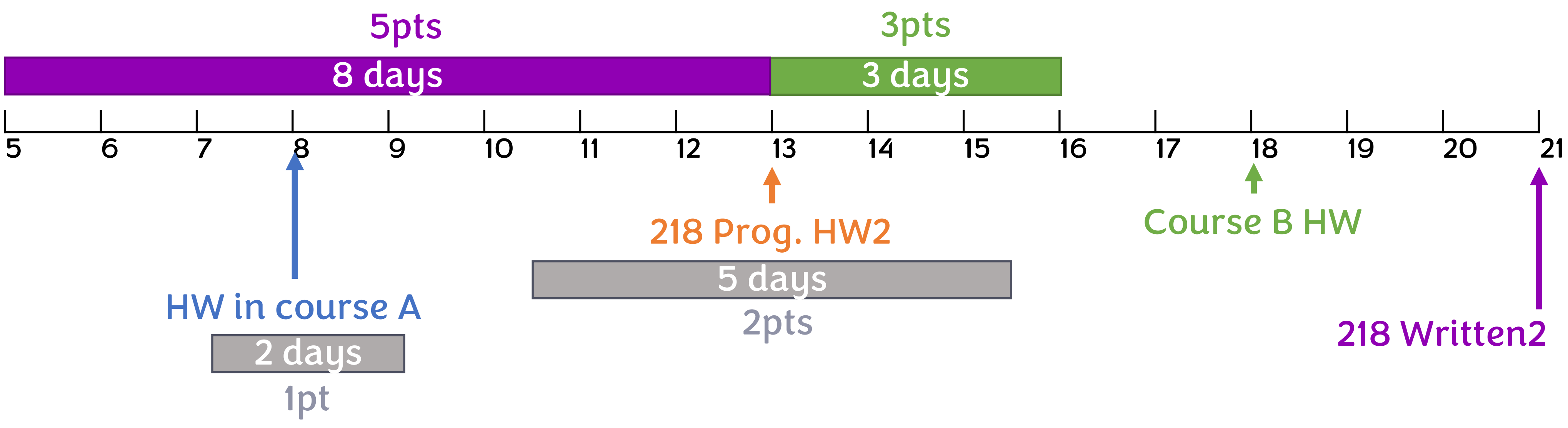
HW in course A
Due 8th
1 point
Takes 2 days

HW in course B
Due 18th
3 points
Takes 3 days

A. Deadline first **6pts**

B. Highest score first

B. Highest Score First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

218 written II
Due 21st
5 points
Takes 8 days

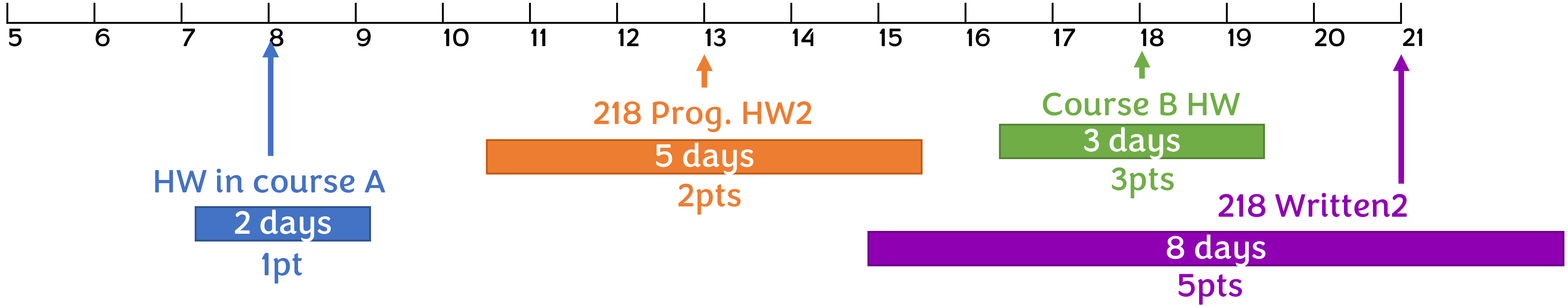
HW in course A
Due 8th
1 point
Takes 2 days

HW in course B
Due 18th
3 points
Takes 3 days

A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest First Strategy



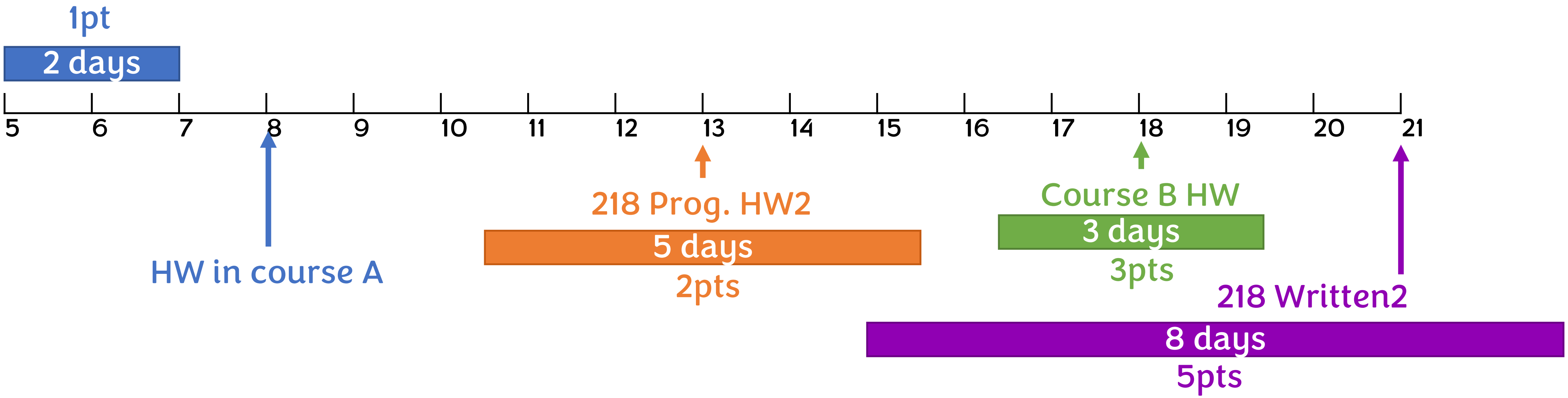
218 programming II	218 written II	HW in course A	HW in course B
Due 13th	Due 21st	Due 8th	Due 18th
2 points	5 points	1 point	3 points
Takes 5 days	Takes 8 days	Takes 2 days	Takes 3 days

A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest first

C. Shortest First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

218 written II
Due 21st
5 points
Takes 8 days

HW in course A
Due 8th
1 point
Takes 2 days

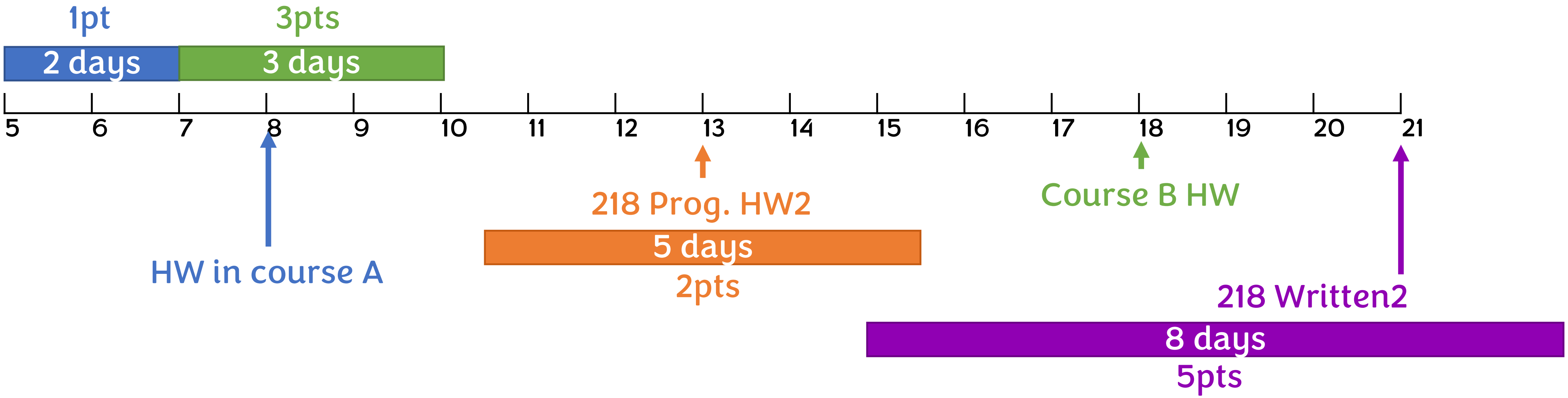
HW in course B
Due 18th
3 points
Takes 3 days

A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest first

C. Shortest First Strategy



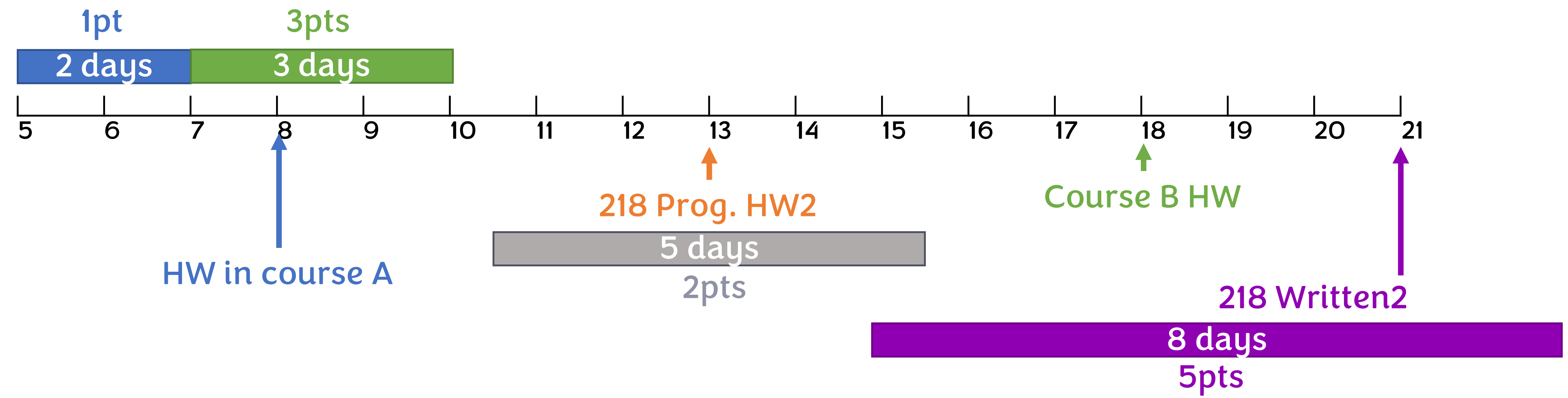
218 programming II Due 13th 2 points Takes 5 days	218 written II Due 21st 5 points Takes 8 days	HW in course A Due 8th 1 point Takes 2 days	HW in course B Due 18th 3 points Takes 3 days

A. Deadline first **6pts**

C. Shortest first

B. Highest score first **8pts**

C. Shortest First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

218 written II
Due 21st
5 points
Takes 8 days

HW in course A
Due 8th
1 point
Takes 2 days

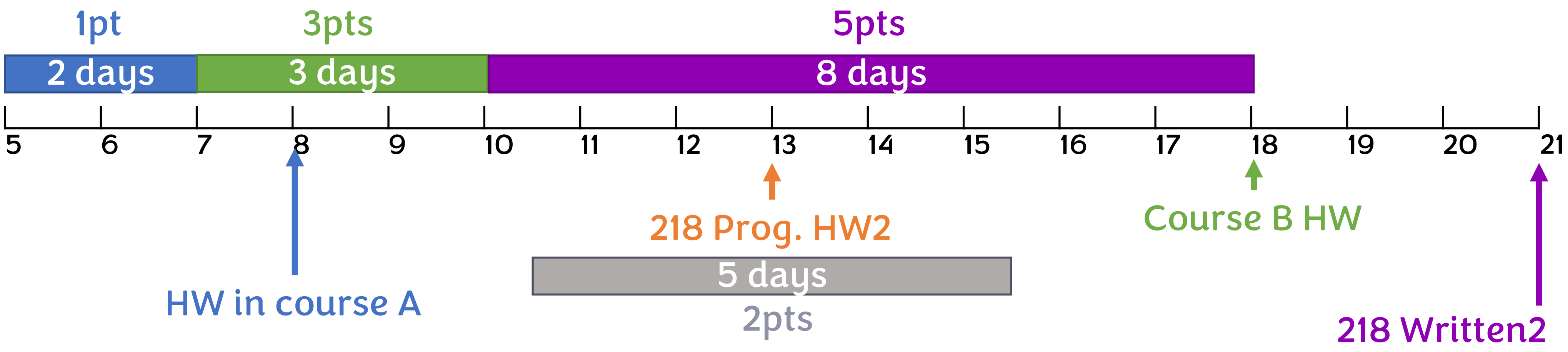
HW in course B
Due 18th
3 points
Takes 3 days

A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest first

C. Shortest First Strategy



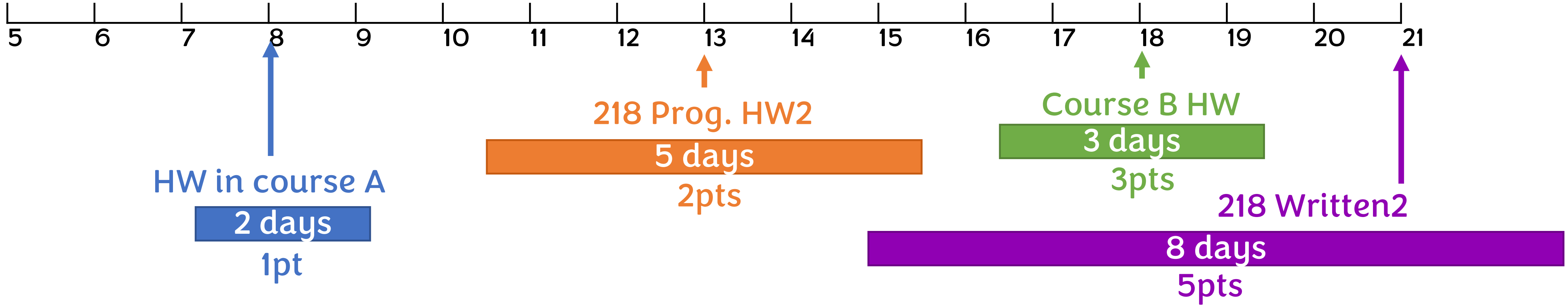
218 programming II Due 13^{th} 2 points Takes 5 days	218 written II Due 21^{st} 5 points Takes 8 days	HW in course A Due 8^{th} 1 point Takes 2 days	HW in course B Due 18^{th} 3 points Takes 3 days

A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest first **9pts**

D. Longest First Strategy



218 programming II	218 written II	HW in course A	HW in course B
Due 13th	Due 21st	Due 8th	Due 18th
2 points	5 points	1 point	3 points
Takes 5 days	Takes 8 days	Takes 2 days	Takes 3 days

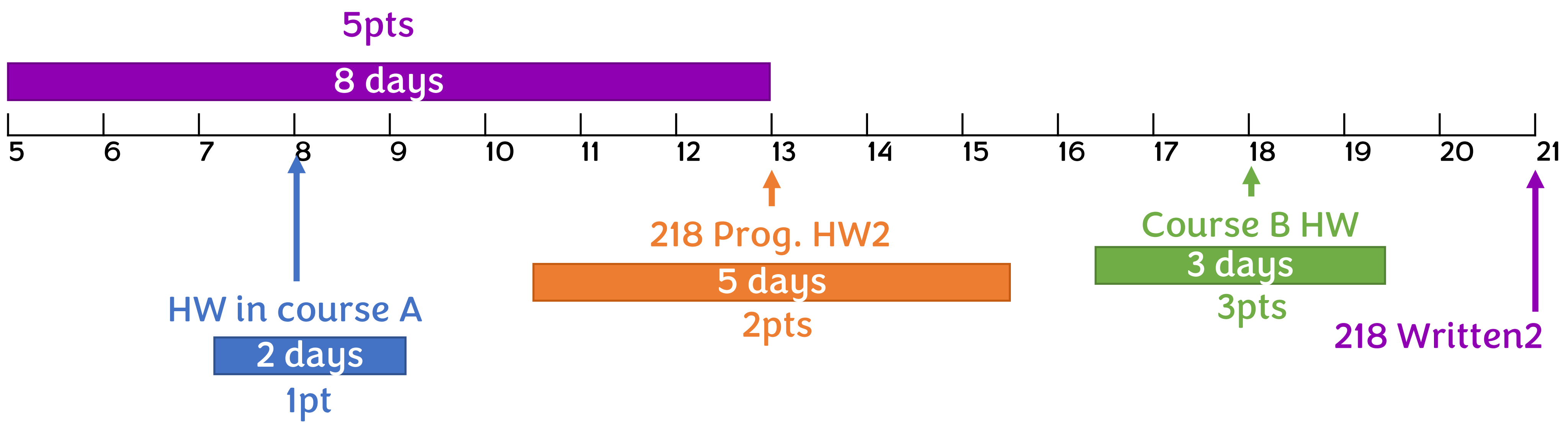
A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest first **9pts**

D. Longest first

D. Longest First Strategy



218 programming II Due 13th 2 points Takes 5 days	218 written II Due 21st 5 points Takes 8 days	HW in course A Due 8th 1 point Takes 2 days	HW in course B Due 18th 3 points Takes 3 days

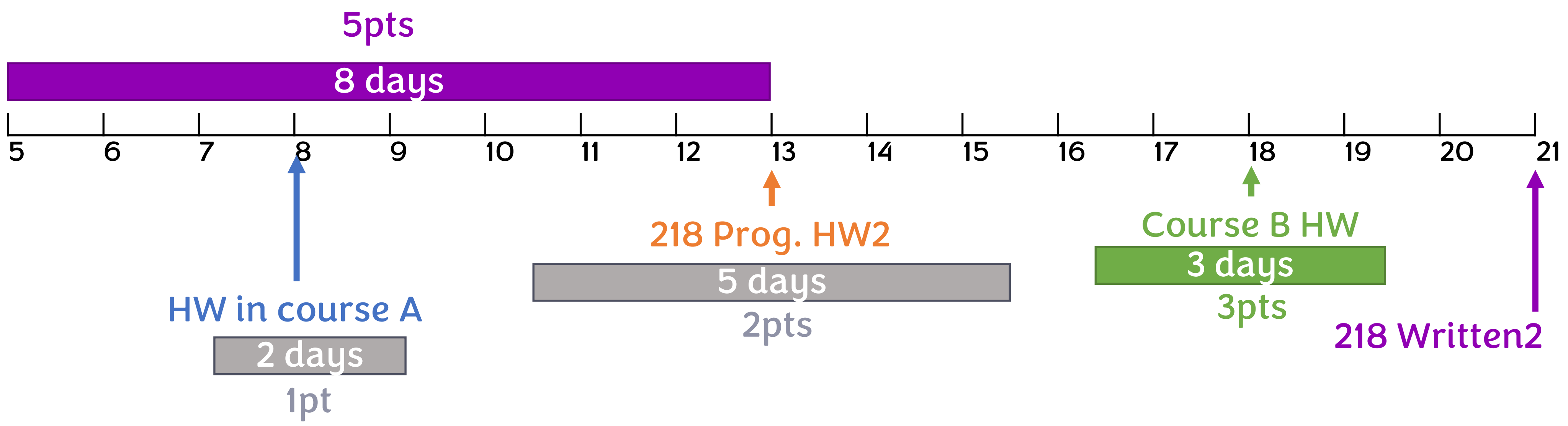
A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest first **9pts**

D. Longest first

D. Longest First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

218 written II
Due 21st
5 points
Takes 8 days

HW in course A
Due 8th
1 point
Takes 2 days

HW in course B
Due 18th
3 points
Takes 3 days

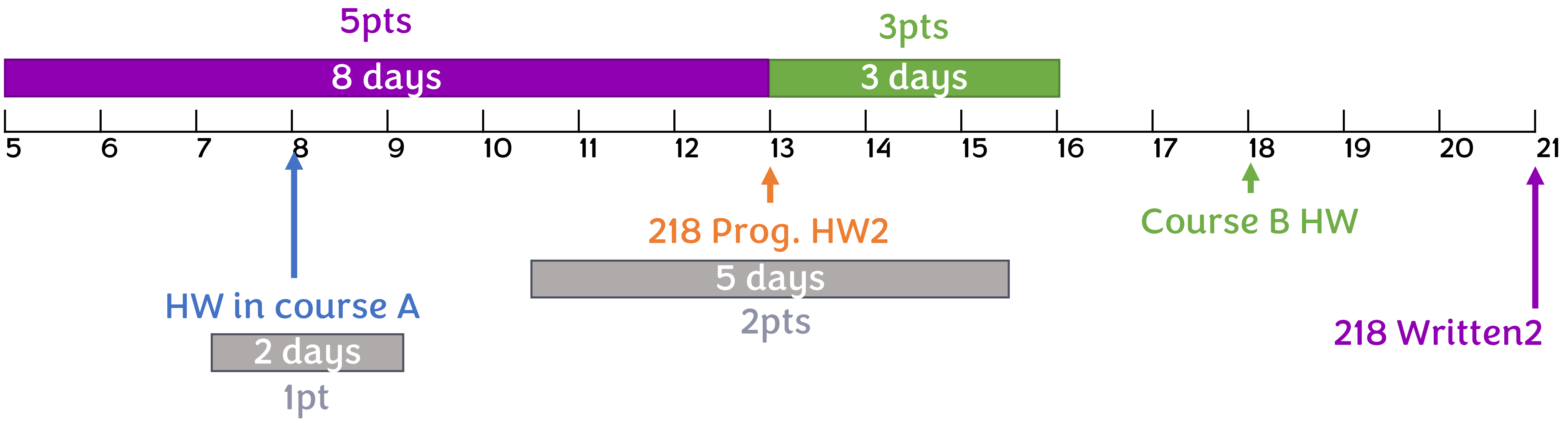
A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest first **9pts**

D. Longest first

D. Longest First Strategy



218 programming II
Due 13th
2 points
Takes 5 days

218 written II
Due 21st
5 points
Takes 8 days

HW in course A
Due 8th
1 point
Takes 2 days

HW in course B
Due 18th
3 points
Takes 3 days

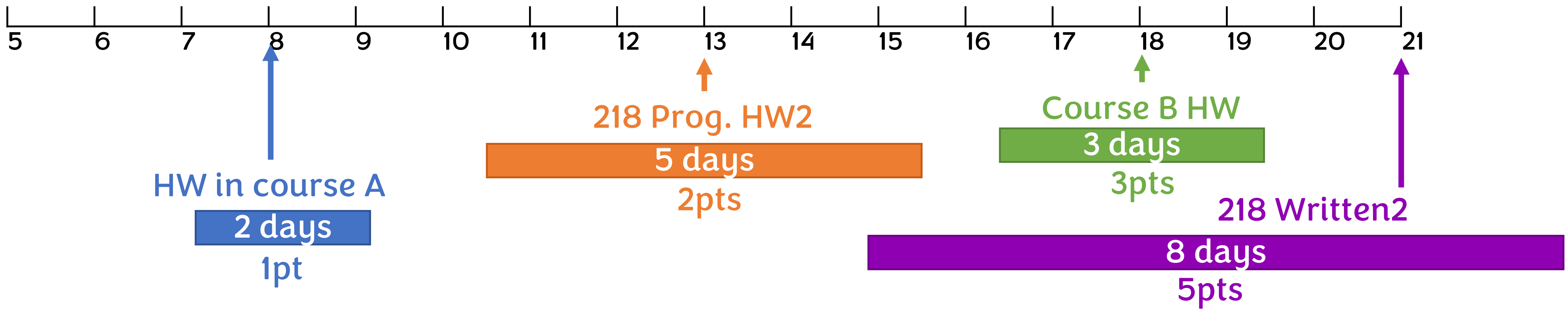
A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest first **9pts**

D. Longest first **8pts**

A Better Strategy!



218 programming II	218 written II	HW in course A	HW in course B
Due 13th	Due 21st	Due 8th	Due 18th
2 points	5 points	1 point	3 points
Takes 5 days	Takes 8 days	Takes 2 days	Takes 3 days

A. Deadline first **6pts**

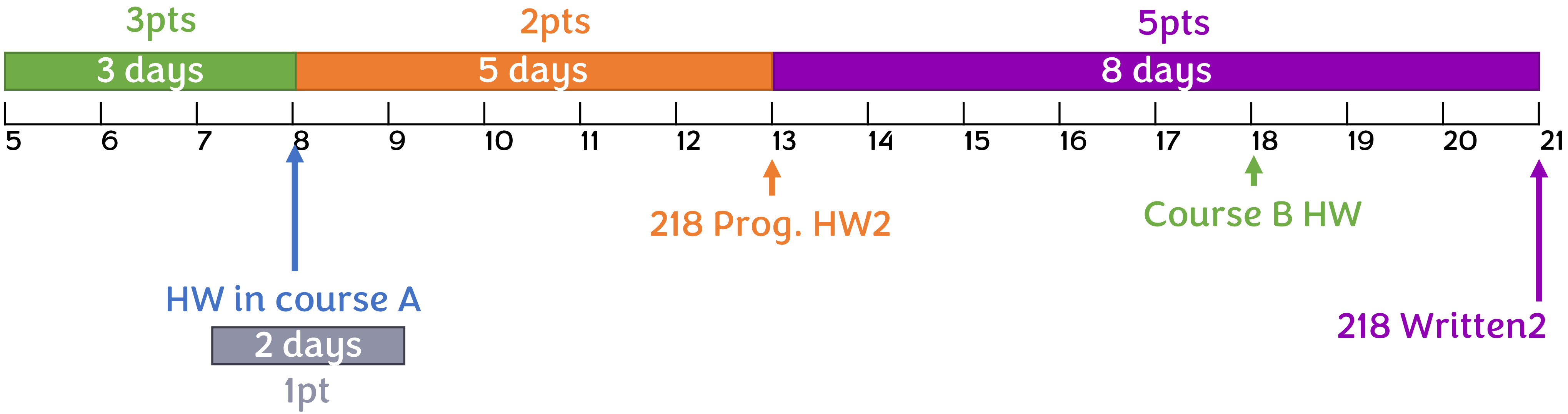
B. Highest score first **8pts**

C. Shortest first **9pts**

D. Longest first **8pts**

A Better Strategy!

10pts in total!



218 programming II	218 written II	HW in course A	HW in course B
Due 13th	Due 21st	Due 8th	Due 18th
2 points	5 points	1 point	3 points
Takes 5 days	Takes 8 days	Takes 2 days	Takes 3 days

A. Deadline first **6pts**

B. Highest score first **8pts**

C. Shortest first **9pts**

D. Longest first **8pts**

How to be greedy?

- **Only care about the immediate reward for any decision make!**
- **I have a few homework assignments to do, which one should I start first?**
  - (For simplicity, we assume you can always get full score using a certain time)
  - A. Work on the one with the earliest deadline!
  - B. Work on the one that worth the highest points!
  - C. Work on the easiest one that requires the least time!
  - D. Work on the hardest one that requires the most time!

- **There can be different greedy strategies based on different criteria**
- **They give you different solutions**
- **Not necessarily optimal**

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**

- **A sequence of tasks with workload/deadline/reward, maximize reward while finish before deadline**
  - Items: tasks; constraints: finish before deadline; optimize: total reward
- **A set of products with weight/value, put into a bag of weight limit x and maximize value**
  - Items: products; constraints: weight limit; optimize: total value
- **A file in computer, encode/compress it to minimize the length**
  - Items: codewords for each character; constrains: original file recoverable; optimize: code length
- **Shortest-paths, minimum spanning tree, …**

- **Not an optimization problem: sorting**

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…**
  - Conclusion: Plan ahead when you work on homework assignments!
  - (and don’t give up any assignments of 218)
  - Greedy solution is not necessary to be optimal!
- **Sometimes greedy may also be good enough?**
  - When you can prove it!

Example: Buying Gifts

Buying gifts

- Yihan is going to buy candies for 218 students
- Her budget is s dollars
- There are n candies in store, with price $p[i]$ each
- She doesn't want to buy the same candy twice
- She wants to buy as many candies as possible

Buying gifts

- Lowest price first!
- Consider the budget $s = 30$
- Can buy 6 items in total

Buying gifts

- Other solutions with 6 candies?

Buying gifts

- Other solutions with 6 candies?

Buying gifts: the first decision

- **Buying the $1 candy is never a bad idea**
- **If you don't buy it in an optimal solution, you can always substitute any chosen candy with the cheapest one!**
  - So why don't we buy the cheapest candy? It never hurts!

The first decision: Greedy choice

Greedy Choice: The greedy choice is part of the optimal answer

- Claim: There exists an optimal solution that chooses KitKat
- Assume we have an optimal solution $\{c_1, c_2, \ldots, c_t\}$
- If KitKat is in the set, problem solved!
- If KitKat is not in it, then, $\{\text{KitKat}, c_2, \ldots, c_t\}$ is also an optimal solution (same size, even lower budget used)! Problem solved!

The first decision: Greedy choice

Greedy Choice: The greedy choice is part of the optimal answer

- Buying the cheapest candy is part of an optimal answer
- If not, we can construct another optimal solution with the cheapest candy!
- So choosing it is good!

Buying gifts: What to do next?

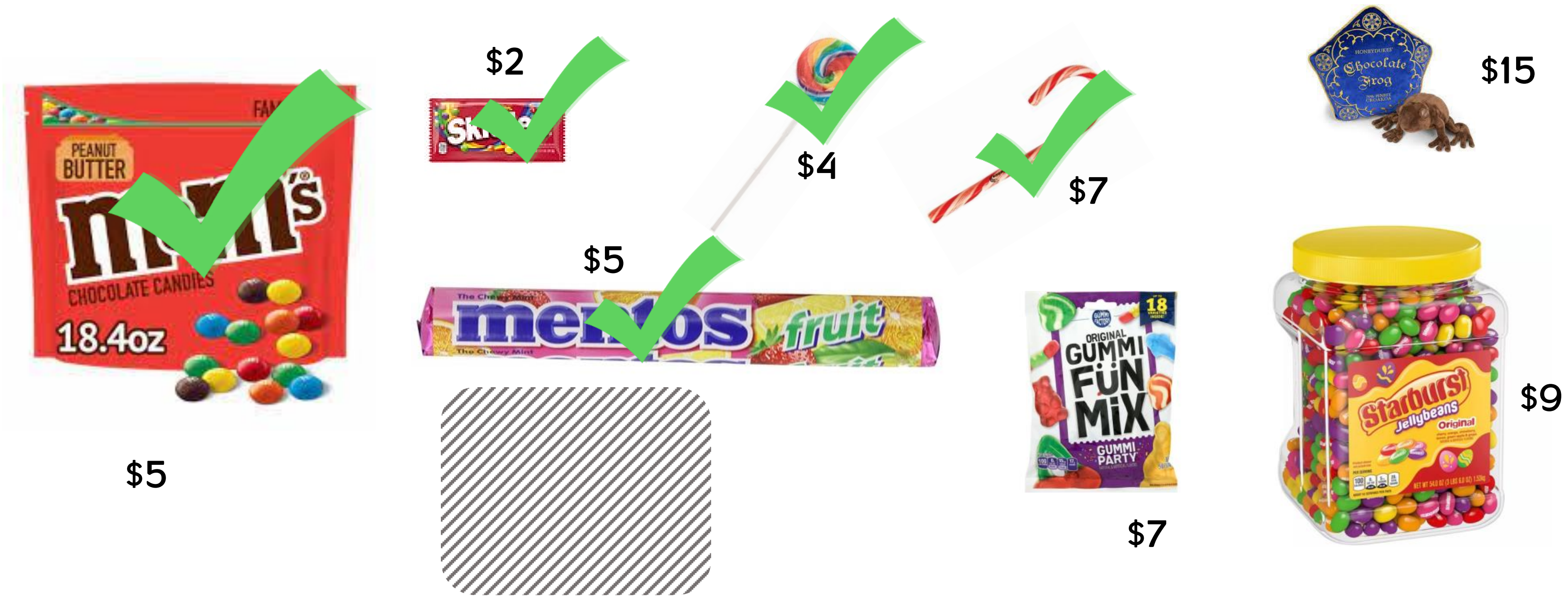
- **Choose from the rest 8 candies using 29 dollars! (The same optimization problem!)**
- **Wait... What if we should not use the optimal solution for 29 dollars?**
  - Assume to the contrary that the optimal solution is $1 candy + another solution

Buying gifts: What to do next?

- **Choose from the rest 8 candies using 29 dollars! (The same optimization problem!)**
- **Wait... What if we should not use the optimal solution for 29 dollars?**
  - Assume to the contrary that the optimal solution is $1 candy + another solution
  - Then $1 candy + optimal solution for the rest is better

Buying gifts: What to do next?

- **Choose from the rest 8 candies using 29 dollars! (The same optimization problem!)**
- **Wait... What if we should not use the optimal solution for 29 dollars?**
  - Assume to the contrary that the optimal solution is $1 candy + another solution
  - Then $1 candy + optimal solution for the rest is better



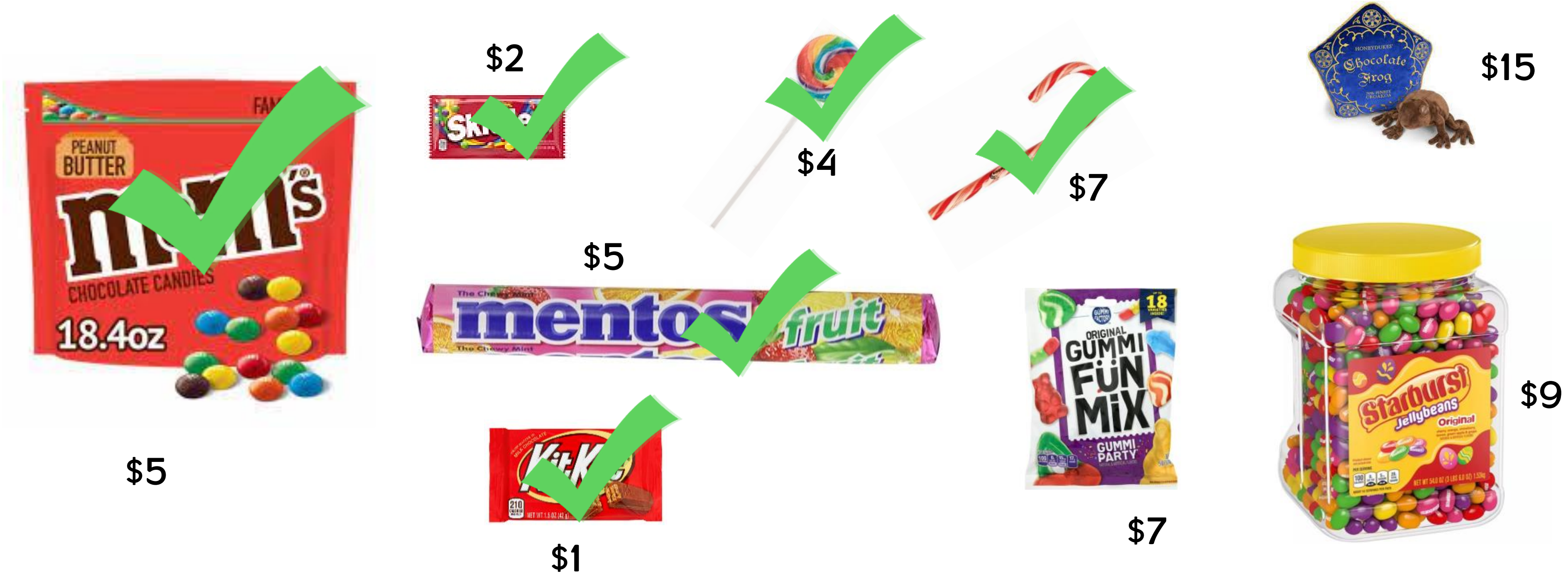
Buying gifts: What to do next?

- The same optimization problem with $n-1$ candies and $s-p_1$ budget!
- Use the same algorithm again: repeat to find the cheapest candy!

What to do next: Optimal substructure

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: 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!

Is this the only way to prove the optimality of greedy algorithms?

- We may have simpler proofs for each specific problem
- 6 is the optimal solution – why? Because if you want to buy 7, you need to have at least 1+2+4+5+5+7+7=31 dollars
- But using “greedy choice” and “optimal substructure” is a general way that works for multiple problems!

Activity Selection

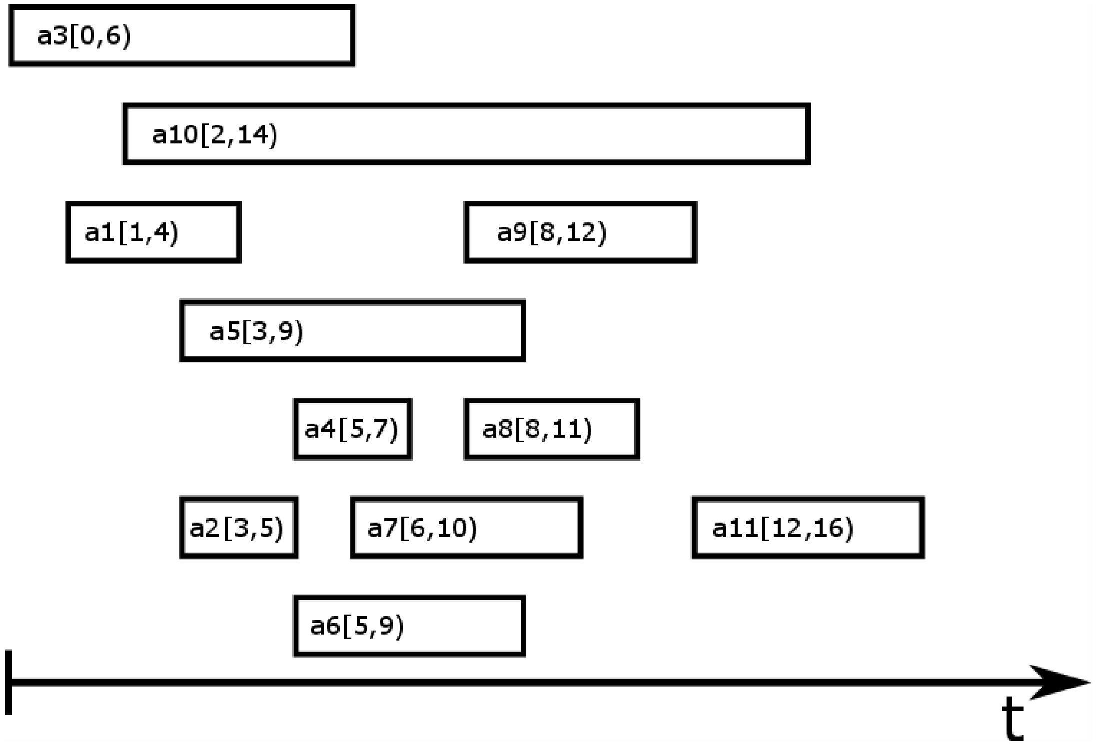
a.k.a. Task Scheduling

CLRS 16.1

Activity Selection Problem

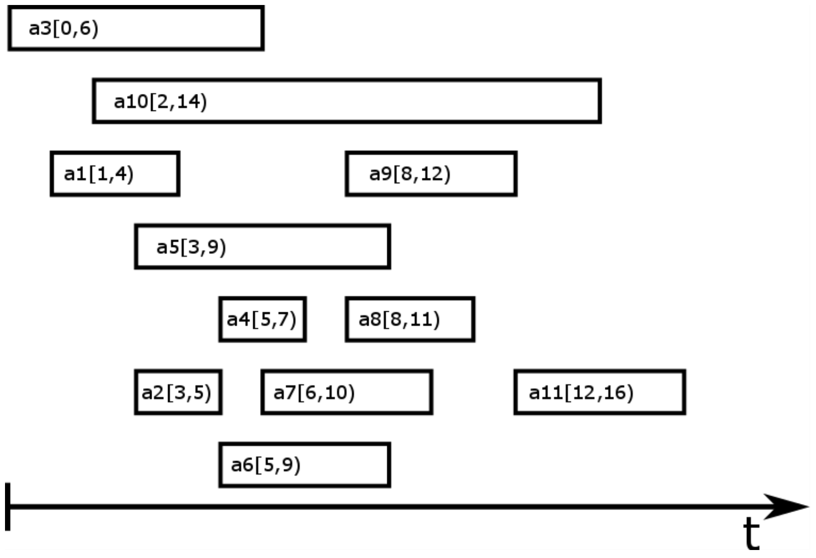
- You received a lot of party invitations!
- Each party is a time interval $[s_i, f_i)$
- You want to join as many as possible! However…
  - No two parties at the same time!
  - For any party you go to, you have to arrive when the party starts, and stay until the end
- Assume you don't need anytime for transportation

- How many parties can you join at most?

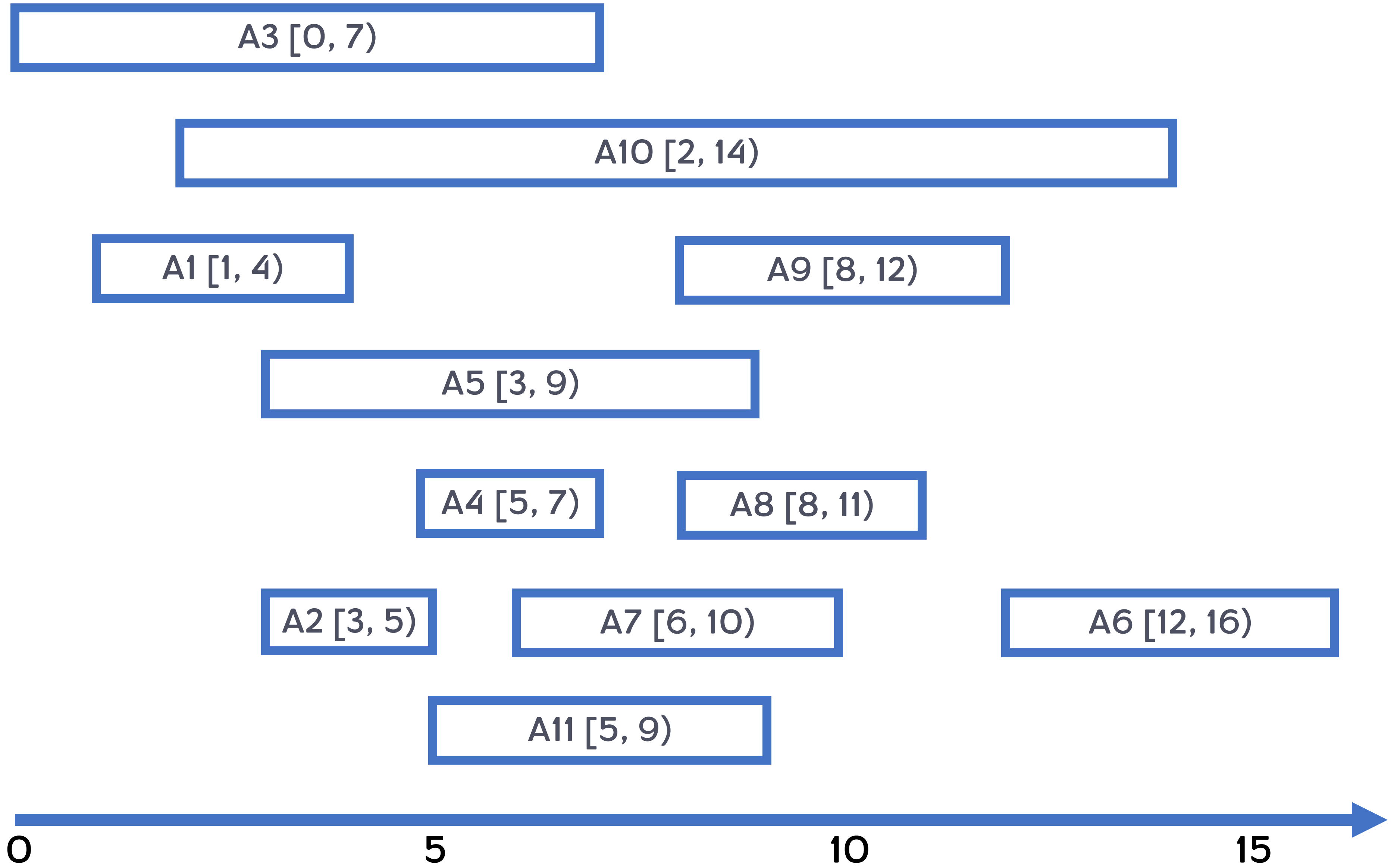


Activity Selection Problem

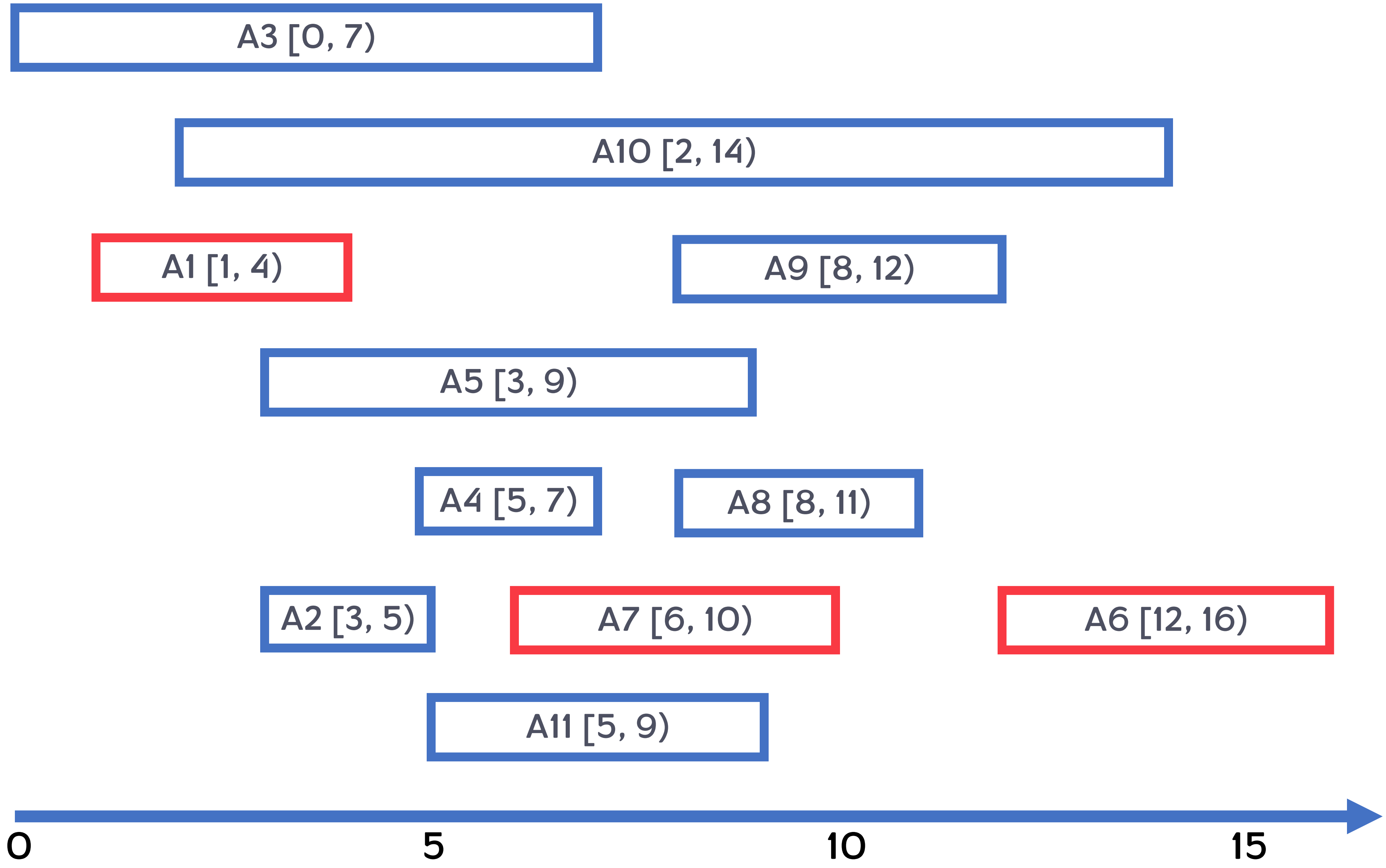
- Given a set of activities $S = \{a_1, a_2, \dots, a_n\}$
- Activity i has a start time s_i and a finish time f_i, taking half-open time interval $[s_i, f_i)$.
- Two activities are said to be compatible if they do not overlap.
- The problem is to find a **maximum-size compatible subset**, i.e., a one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



Example of Activity Selection

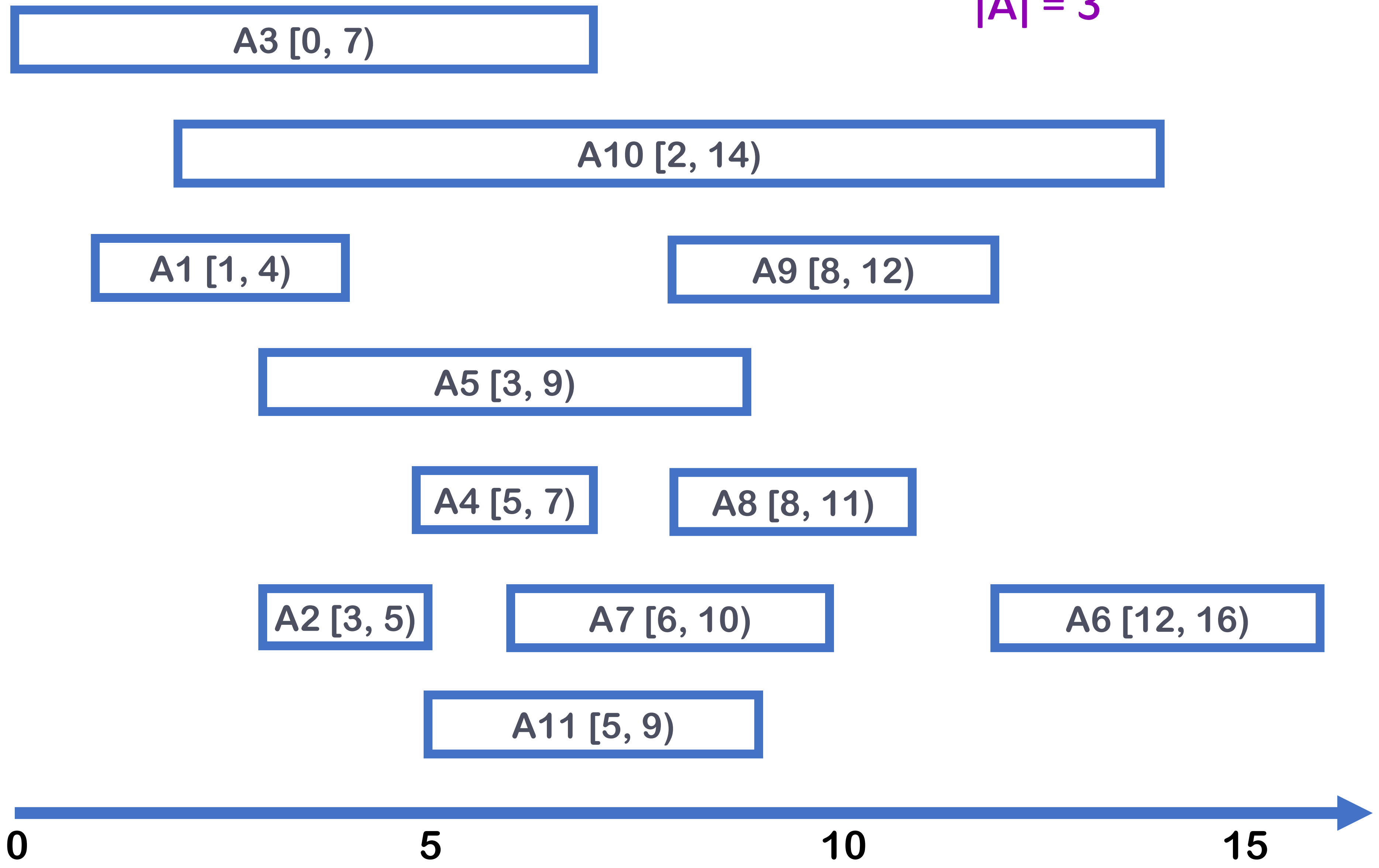


One solution: 3 activities



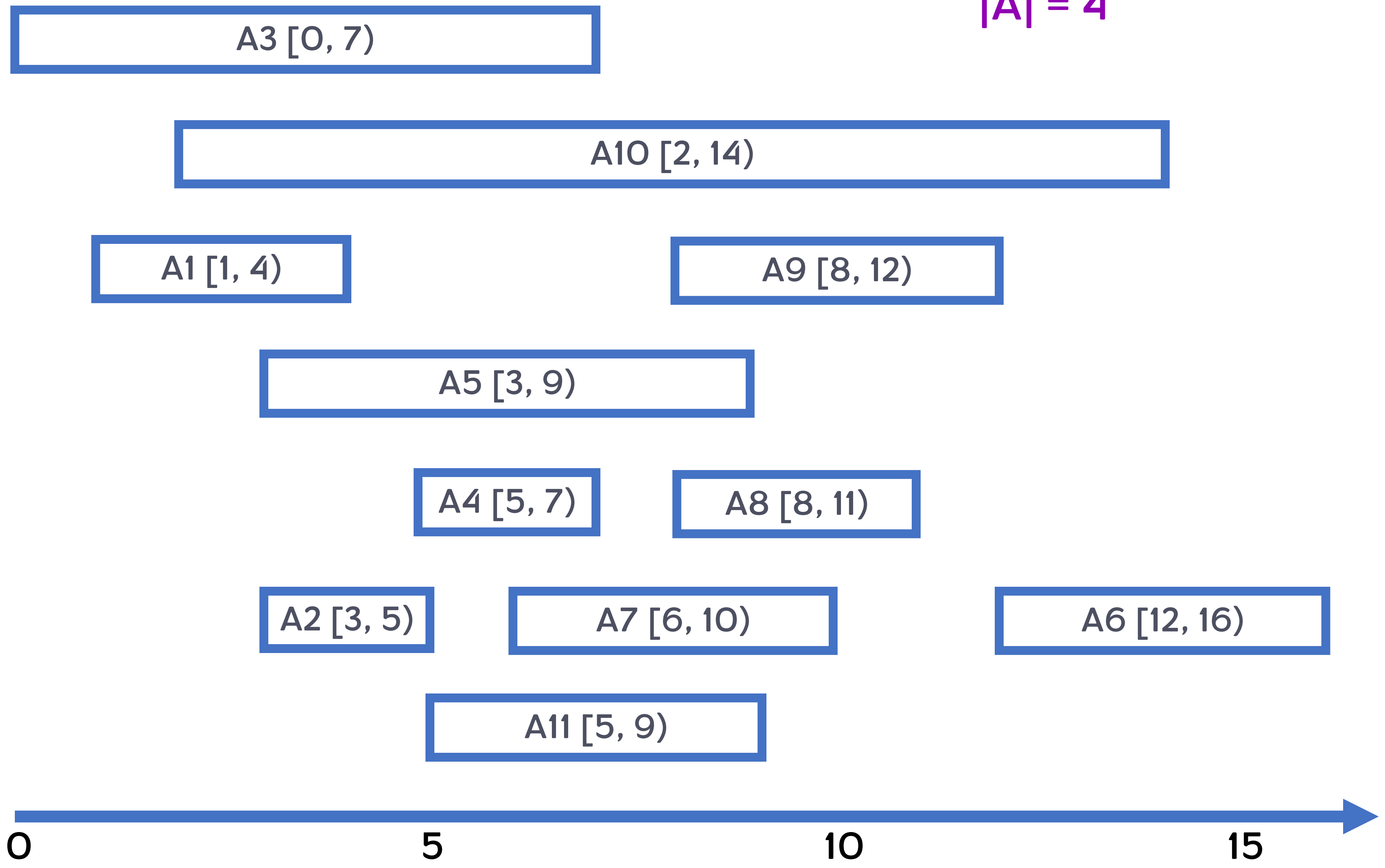
Strategy 1: earliest start first

Solution: A = {A3, A9, A6}
|A| = 3



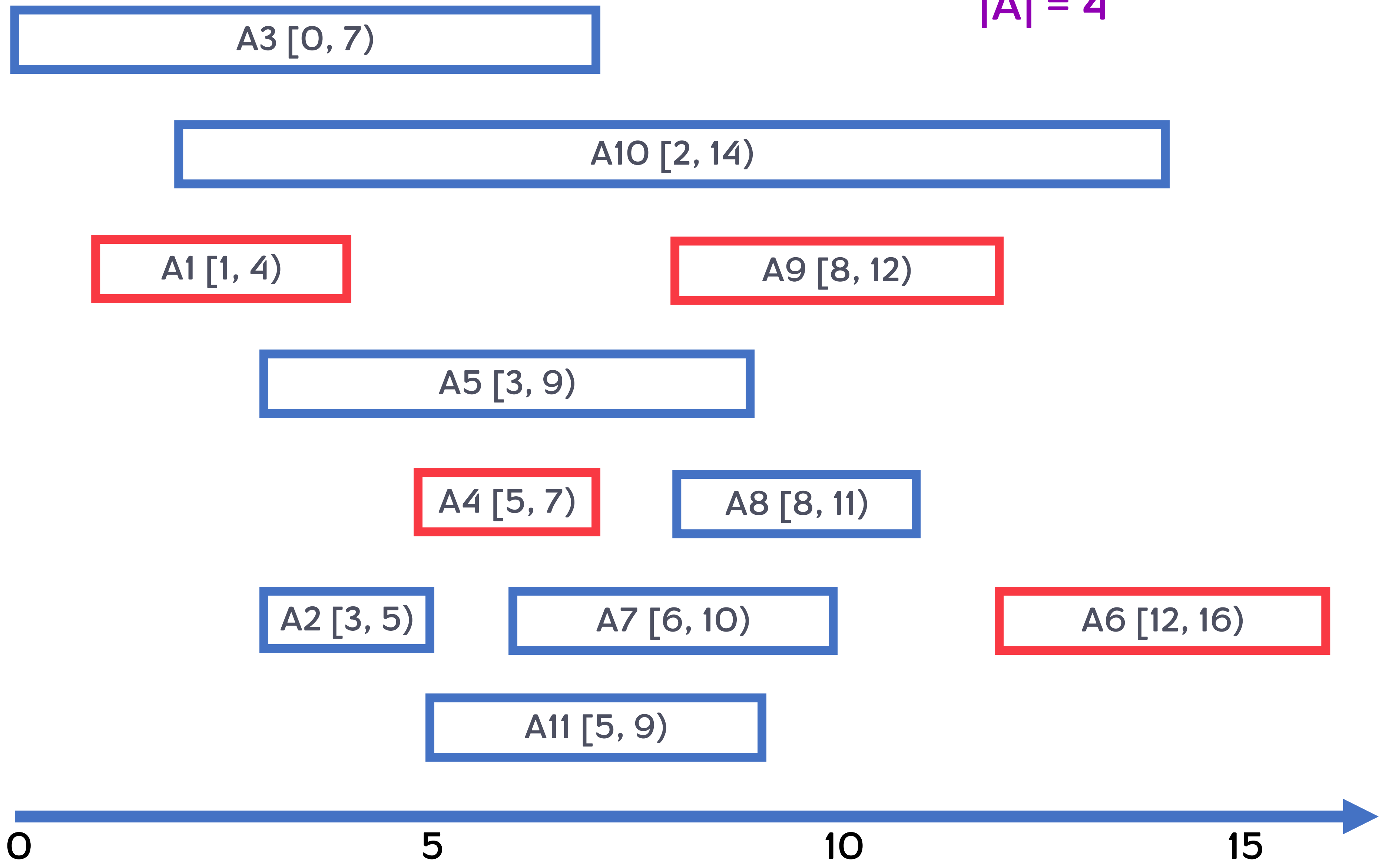
Strategy 2: smallest first

Solution: A = {A4, A2, A8, A6}
|A| = 4



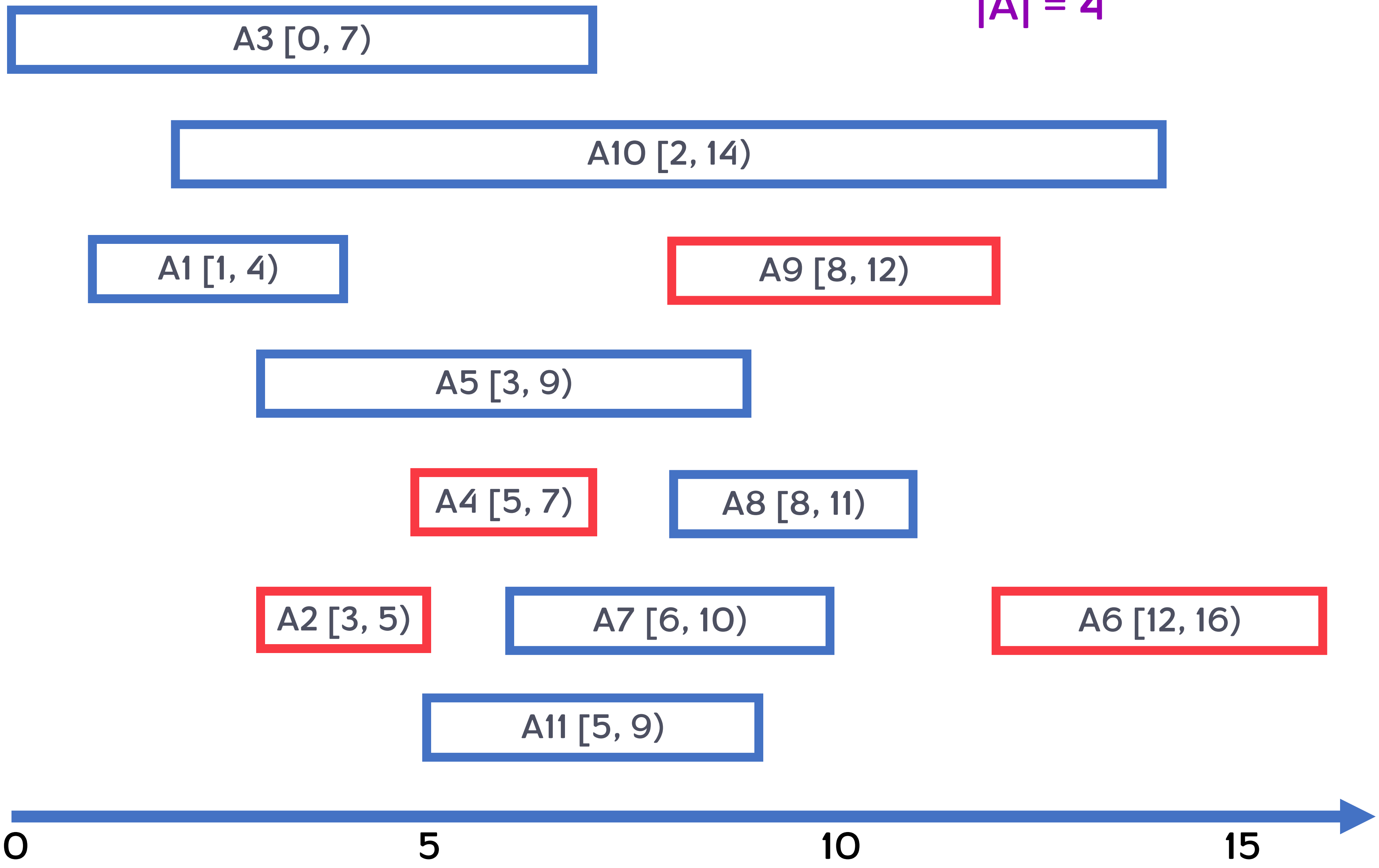
Another solution: 4 activities

Solution: A = {A1, A4, A9, A6}
|A| = 4



Another solution: 4 activities

Solution: A = {A2, A4, A8, A6}
|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?

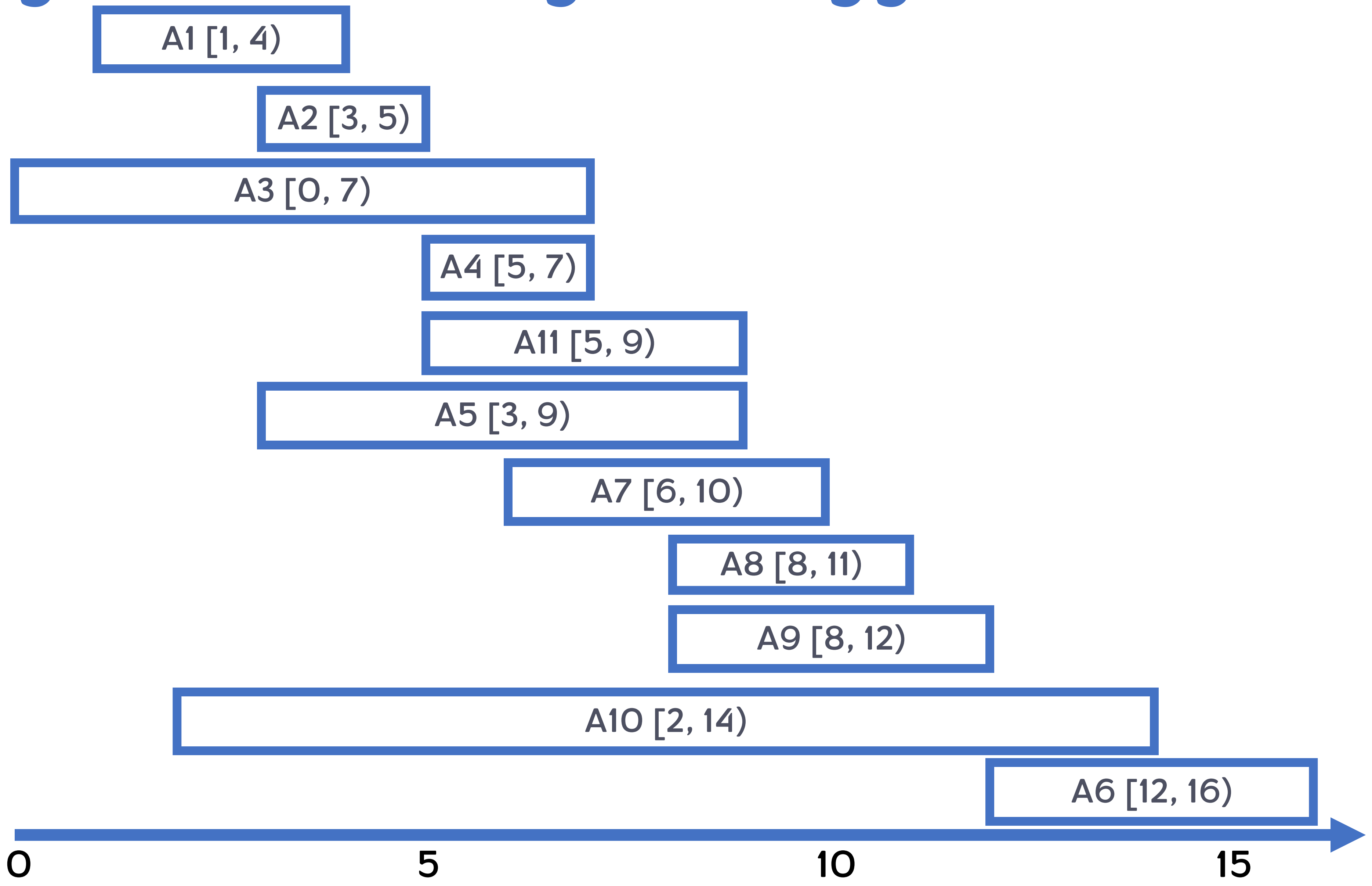
Smallest first: 1 activity
Optimal solution: 2 activities

Early Finish Greedy Strategy

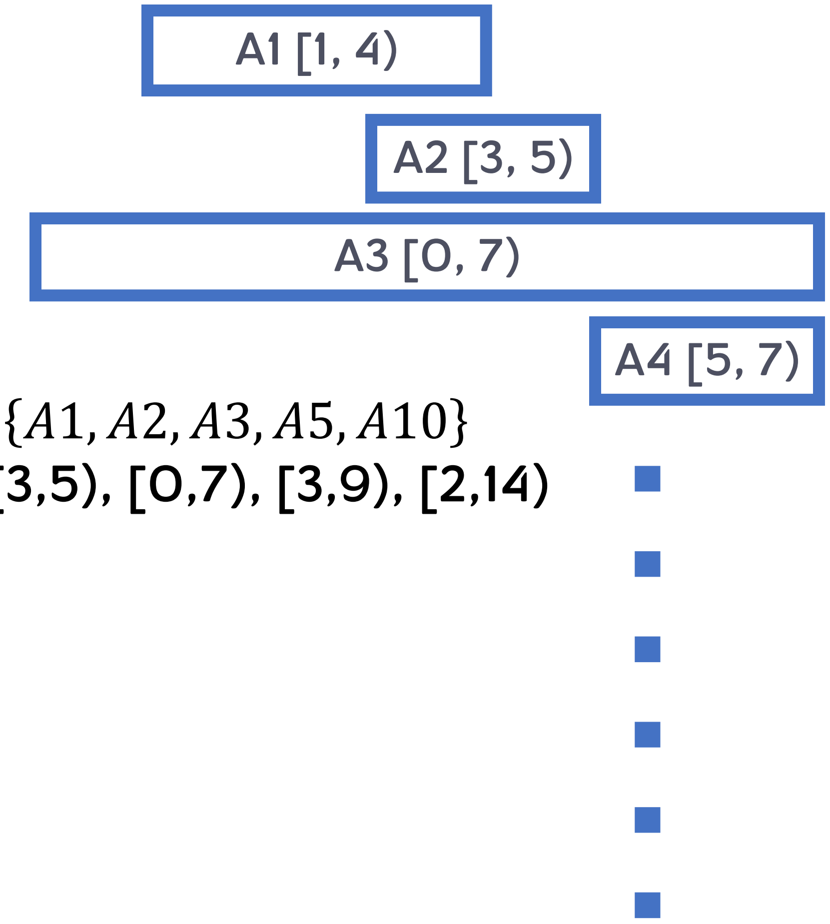
activity_selection (S: a set of activities)

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

Early Finish Greedy Strategy



Why it's optimal?

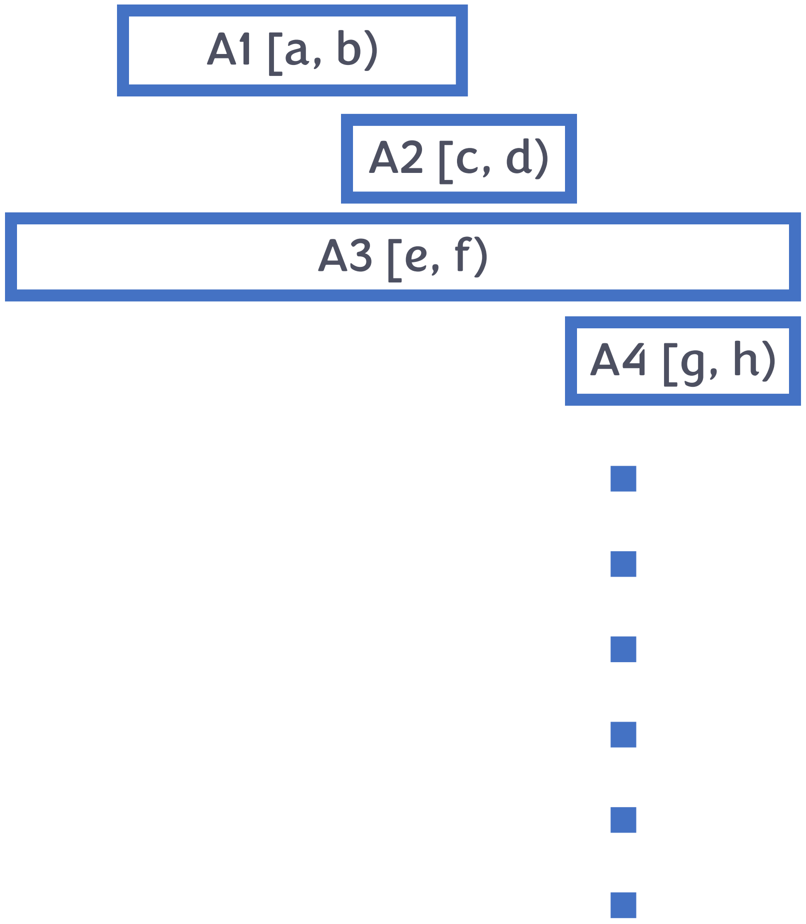


$T = \{A1, A2, A3, A5, A10\}$
= [1,4), [3,5), [0,7), [3,9), [2,14)

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 $\geq$ 4: otherwise [1, 4) is not the earliest finish
  - All of them must start $\leq$ 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 use induction to 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 ≥ b: otherwise [a, b) is not the earliest finish
  - All of them must start ≤ 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 $\boldsymbol{t}$ 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 $\boldsymbol{t}$**
- **We want to prove that t is part of an optimal solution**
  - Choosing $\boldsymbol{t}$ is always "good"!
- **Look at an optimal solution $\boldsymbol{A} = \{\boldsymbol{a_1}, \dots, \boldsymbol{a_k}\}$ 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 $\boldsymbol{t}$ may not be in $\boldsymbol{A}$, we can construct another valid solution with $\boldsymbol{t}$ that is as good as $\boldsymbol{A}$! - t is a "**good choice**"!

Greedy Choice

The earliest finish task t is part of some optimal solution

- Look at an optimal solution $A = \{a_1, \ldots, a_k\}$ 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)…
- We want to prove that t is part of an optimal solution
- (case 1) If $a_1 = t$ then we are done
  - A = [1,4), [5,7), [8,12), [12,16) Then $a_1 = [1,4), t = [1,4)$

Greedy Choice

The earliest finish task t 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)…
- **We want to prove that t is part of an optimal solution**
- **(case 1) If $a_1 = t$ then we are done**
  - A = [1,4), [5,7), [8,12), [12,16) Then $a_1 = [1,4), t = [1,4)$
- **(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 $a_1 = [3,5), t = [1,4)$

Greedy Choice

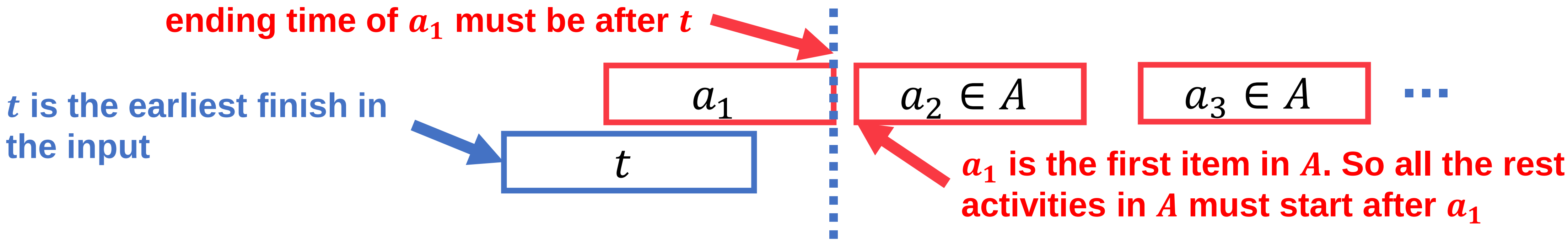
The earliest finish task t is part of some optimal solution

- Look at an optimal solution $A = \{a_1, \ldots, a_k\}$ 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)…
- 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 $a_1 = [3,5), t = [1,4)$
  - 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 t 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 $a_1 = [3,5), t = [1,4)$
  - 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 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 $A' = A - \{a_1\} \cup \{t\}$**
  - 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 a_m is part of some optimal solution

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

Optimal Substructure

Optimal solution $\{a_i,\ldots\}$ without a_i is "the best solution of $S-$ {those incompatible with a_i}"

- **We want to prove that, optimal solution A = $\{a,\ldots\}$ without a is "the best solution of input – {activities incompatible with a}"**
  - Input = [1,4), [3,5), [0,7), [5,7), [5,9), [3,9), [6,10)…
  - Optimal solution A = [1,4), [5,7), [8,12), [12,16) = [1,4) + **[?]**
  - Input* = ~~[1,4)~~, ~~[3,5)~~, ~~[0,7)~~, [5,7), [5,9), ~~[3,9)~~, [6,10)…
  - **[?]** = [5,7), [8,12), [12,16) must be an optimal solution to Input*
  - Then A = [1,4) + **[?]** = [1,4) + "opt on Input*" = [1,4), [5,7), [8,12), [12,16)
- **Prove by contradiction**
  - If we don't use the optimal solution of Input*, why don't we use it?
  - See detailed proof in textbook

Prove the optimality of a greedy algorithm: activity selection

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

2. **Optimal Substructure**: optimal solution A = $\{a, \ldots\}$ 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!**

The Greedy Method

- Applied to optimization problems
- Adds items to the solution one-by-one
- Always choose the current best solution
- No backtracking

- Not necessarily optimal!
- Need to prove it

The greedy strategy is extremely widely used

- **Standard Greedy Algorithms:**
  - Activity Selection Problem
  - Egyptian Fraction
  - Job Sequencing Problem
  - Job Sequencing Problem (Using Disjoint Set)
  - Job Sequencing Problem – Loss Minimization
  - Job Selection Problem – Loss Minimization Strategy | Set 2
  - Huffman Coding
  - Efficient Huffman Coding for sorted input
  - Huffman Decoding
  - Water Connection Problem
  - Policemen catch thieves
  - Minimum Swaps for Bracket Balancing
  - Fitting Shelves Problem
  - Assign Mice to Holes

From: https://www.geeksforgeeks.org/greedy-algorithms/

The greedy strategy is extremely widely used

- **Greedy Algorithms in Graphs:**
  - Kruskal’s Minimum Spanning Tree
  - Prim’s Minimum Spanning Tree
  - Boruvka’s Minimum Spanning Tree
  - Reverse delete algorithm for MST
  - Problem Solving for Minimum Spanning Trees (Kruskal’s and Prim’s)
  - Dijkastra’s Shortest Path Algorithm
  - Dial’s Algorithm
  - Dijkstra’s Algorithm for Adjacency List Representation
  - Prim’s MST for adjacency list representation
  - Correctness of Greedy Algorithms
  - Minimum cost to connect all cities
  - Max Flow Problem Introduction
  - Number of single cycle components in an undirected graph

From: https://www.geeksforgeeks.org/greedy-algorithms/

The greedy strategy is extremely widely used

- **Greedy Algorithms in Arrays :**
  - Minimum product subset of an array
  - Maximum product subset of an array
  - Maximize array sum after k-negations | Set 1
  - Maximize array sum after k-negations | Set 2
  - Maximize the sum of arr[i]*i
  - Maximum sum of increasing order elements from n arrays
  - Maximum sum of absolute difference of an array
  - Maximize sum of consecutive differences in a circular array
  - Maximum height pyramid from the given array of objects
  - Partition into two subarrays of lengths k and (N – k) such that the difference of sums is maximum
  - Minimum sum of product of two arrays
  - Minimum sum by choosing minimum of pairs from array
  - Minimum sum of absolute difference of pairs of two arrays
  - Minimum operations to make GCD of array a multiple of k

- Minimum sum of absolute difference of pairs of two arrays
- Minimum sum of two numbers formed from digits of an array
- Minimum increment/decrement to make array non-Increasing
- Making elements of two arrays same with minimum increment/decrement
- Minimize sum of product of two arrays with permutation allowed
- Sorting array with reverse around middle
- Sum of Areas of Rectangles possible for an array
- Array element moved by k using single moves
- Find if k bookings possible with given arrival and departure times
- Lexicographically smallest array after at-most K consecutive swaps
- Largest lexicographic array with at-most K consecutive swaps

From: https://www.geeksforgeeks.org/greedy-algorithms/

The greedy strategy is extremely widely used

- **Approximate Greedy Algorithms for NP Complete Problems :**
  - Set cover problem
  - Bin Packing Problem
  - Graph Coloring
  - K-centers problem
  - Shortest superstring problem
  - Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming)
  - Traveling Salesman Problem | Set 2 (Approximate using MST)
- **Greedy Algorithms for Special Cases of DP problems :**
  - Fractional Knapsack Problem
  - Minimum number of coins required

From: https://www.geeksforgeeks.org/greedy-algorithms/

The greedy strategy is extremely widely used

- **Misc:**
  - Split n into maximum composite numbers
  - Maximum trains for which stoppage can be provided
  - Buy Maximum Stocks if i stocks can be bought on i-th day
  - Find the minimum and maximum amount to buy all N candies
  - Maximum sum possible equal to sum of three stacks
  - Maximum elements that can be made equal with k updates
  - Divide cuboid into cubes such that sum of volumes is maximum
  - Maximum number of customers that can be satisfied with given quantity
  - Minimum Fibonacci terms with sum equal to K
  - Divide 1 to n into two groups with minimum sum difference
  - Minimize cash flow among friends
  - Minimum rotations to unlock a circular lock
  - Paper cut into minimum number of squares
  - Minimum difference between groups of size two
  - Minimum rooms for m events of n batches with given schedule
  - Connect n ropes with minimum cost
  - Minimum Cost to cut a board into squares
  - Minimum cost to process m tasks where switching costs
  - Minimum cost to make array size 1 by removing larger of pairs
  - Minimum cost for acquiring all coins with k extra coins allowed with every coin
  - Minimum time to finish all jobs with given constraints
  - Minimum number of Platforms required for a railway/bus station
  - Minimize the maximum difference between the heights of towers
  - Minimum increment by k operations to make all elements equal
  - Minimum edges to reverse to make path from a source to a destination
  - Find minimum number of currency notes and values that sum to given amount
  - Minimum initial vertices to traverse whole matrix with given conditions
  - Find the Largest Cube formed by Deleting minimum Digits from a number
  - Check if it is possible to survive on Island
  - Largest palindromic number by permuting digits
  - Smallest number with sum of digits as N and divisible by 10^N
  - Find Smallest number with given number of digits and digits sum
  - Rearrange characters in a string such that no two adjacent are same
  - Rearrange a string so that all same characters become d distance away
  - Print a closest string that does not contain adjacent duplicates
  - Smallest subset with sum greater than all other elements
  - Lexicographically largest subsequence such that every character occurs at least k times
- **Quick Links :**
  - Top 20 Greedy Algorithms Interview Questions
  - ‘Practice Problems’ on Greedy Algorithms
  - Practice Questions on Huffman Encoding
  - ‘Quiz’ on Greedy Algorithms

From: https://www.geeksforgeeks.org/greedy-algorithms/

Huffman Tree and Huffman Codes

Merge pebbles

- We have piles of pebbles:

12

7

8

15

4

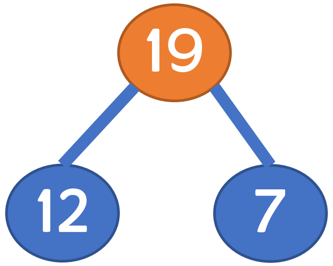
- We want to merge them into one pile, but
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy (Let’s assume you need to move both piles)
  - (e.g., merging 12 and 7 results in a new pile of size (12+7=)19, and cost you 19 units of energy
- **How can we merge all of them with the least energy?**

Merge pebbles

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**

Merge pebbles

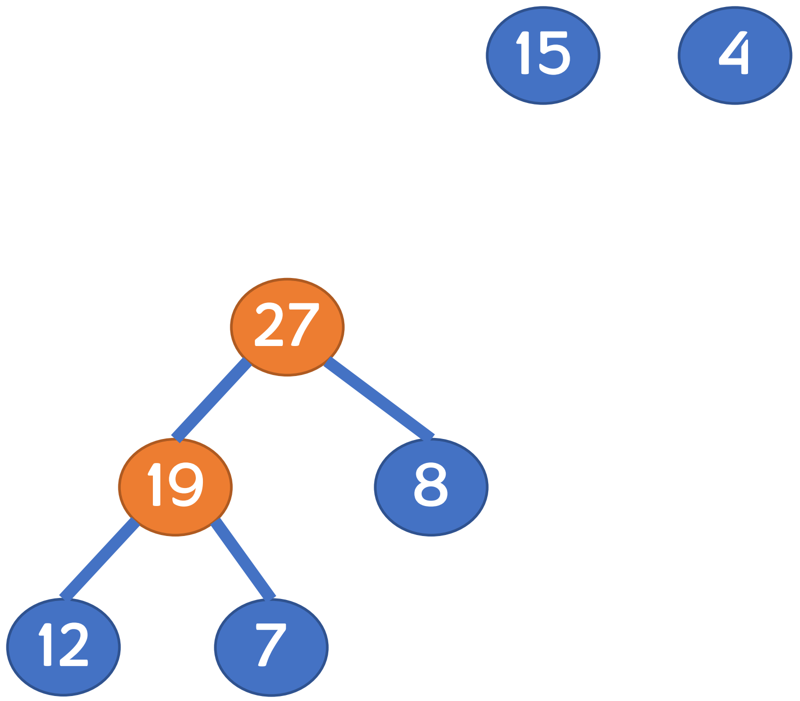
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19

Merge pebbles

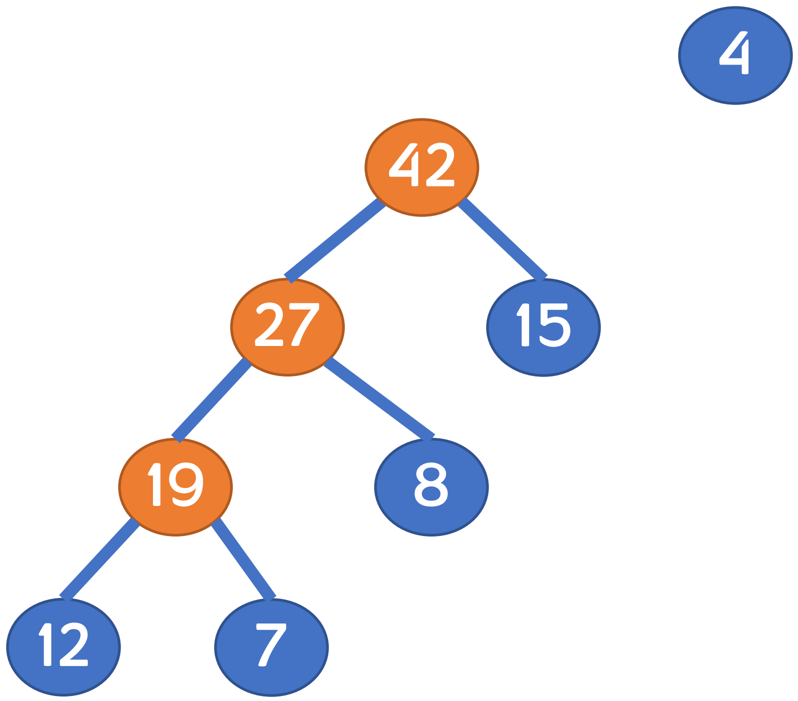
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +27

Merge pebbles

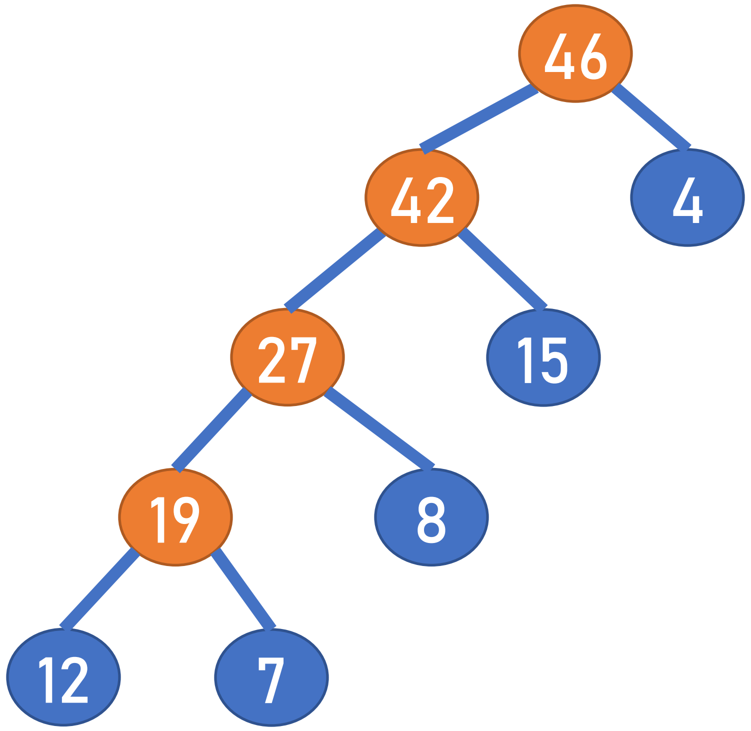
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +27 +42

Merge pebbles

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



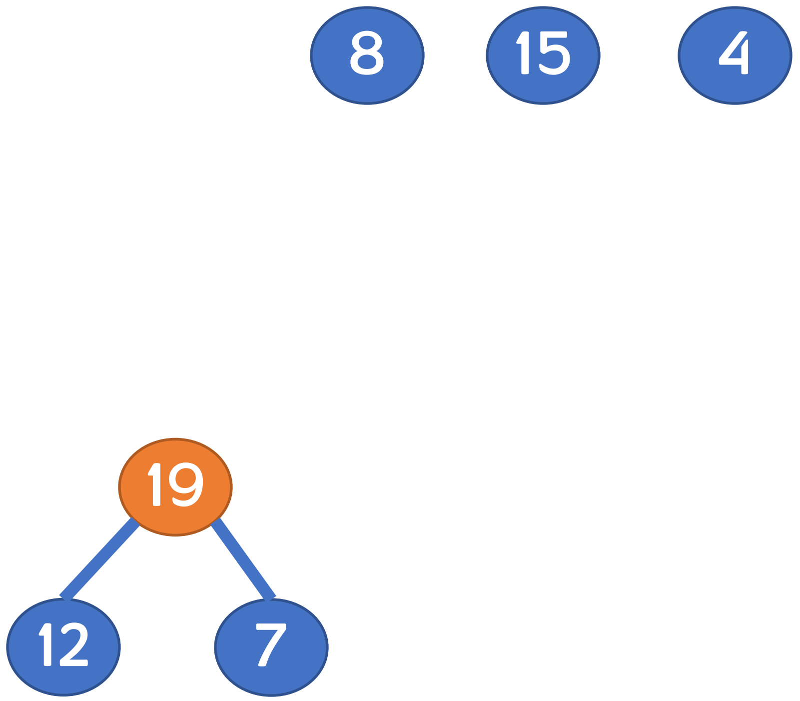
Energy cost: 19 +27 +42 +46 =134

Merge pebbles – another solution

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**

Merge pebbles – another solution

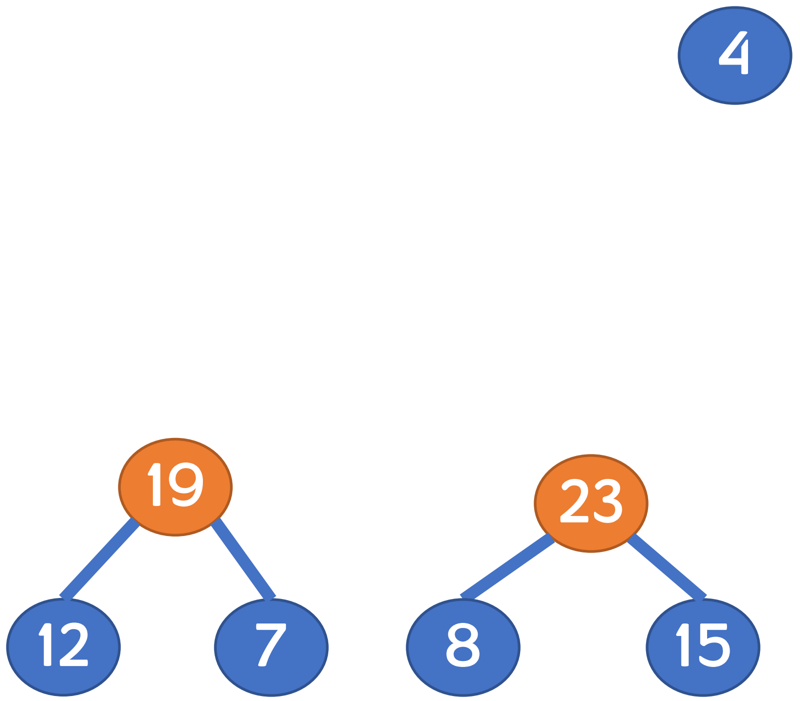
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19

Merge pebbles – another solution

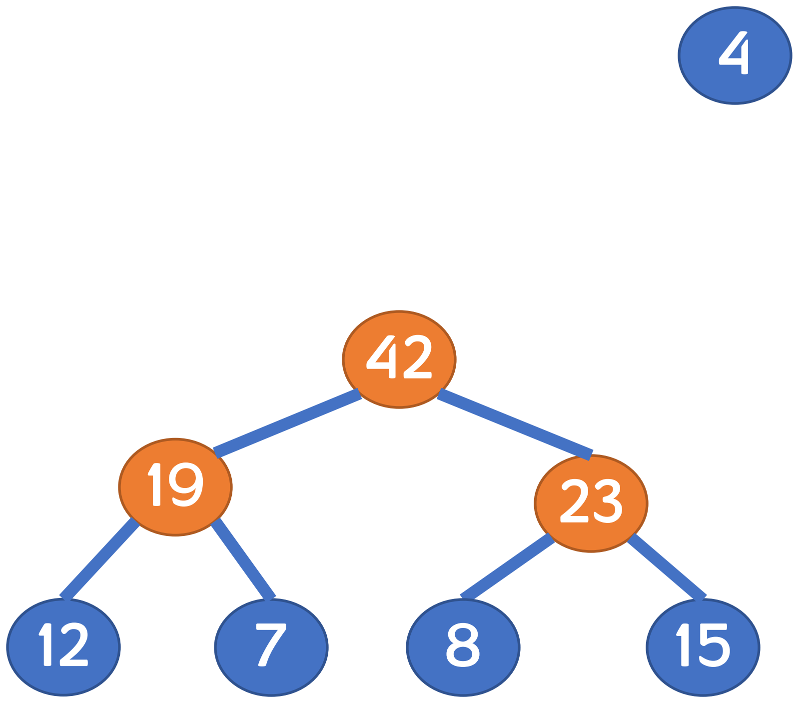
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +23

Merge pebbles – another solution

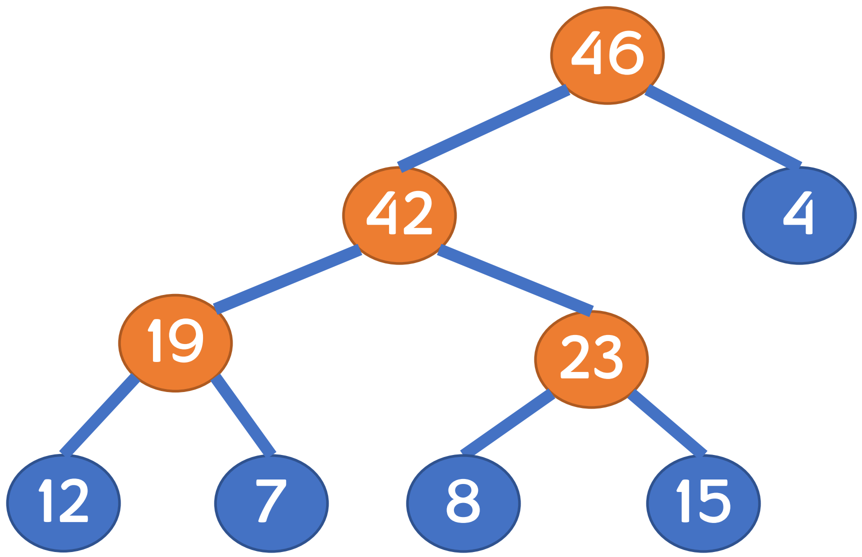
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +23 +42

Merge pebbles – another solution

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**



Energy cost: 19 +23 +42 +46 **=130**

Merge pebbles – Can you come up with a greedy solution?

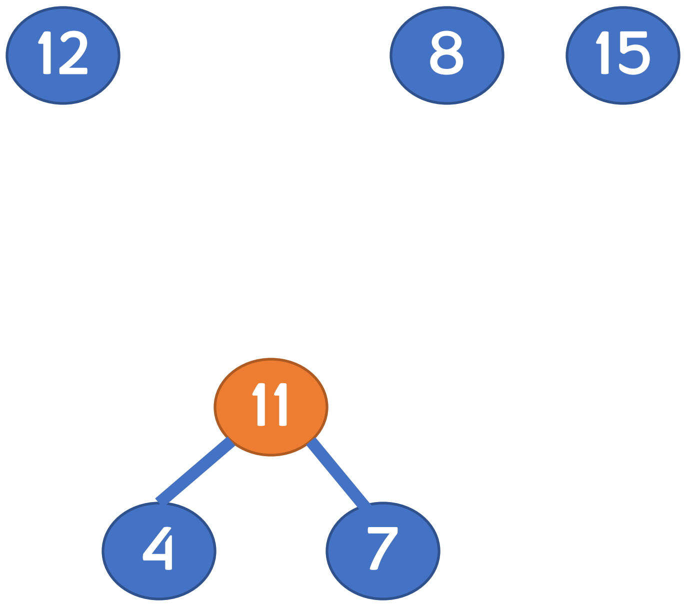
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Use a tree to represent the trace of merging**

Merge pebbles – greedy solution

- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**

Merge pebbles –greedy solution?

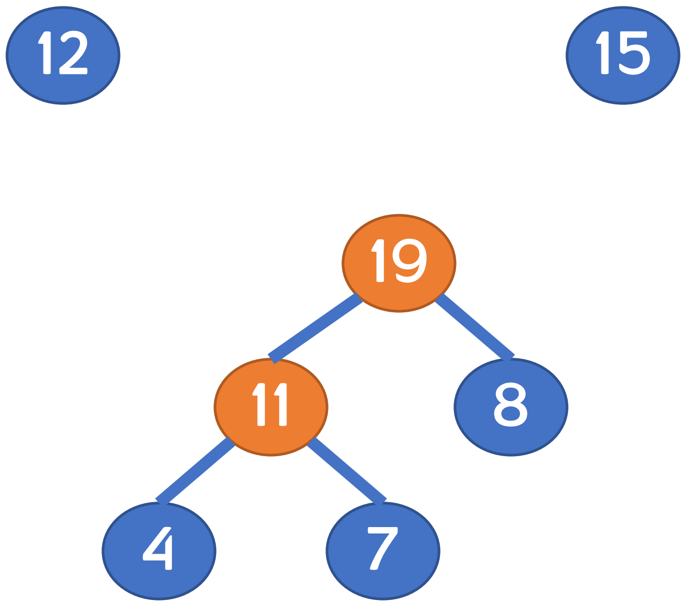
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**



Energy cost: 11

Merge pebbles –greedy solution?

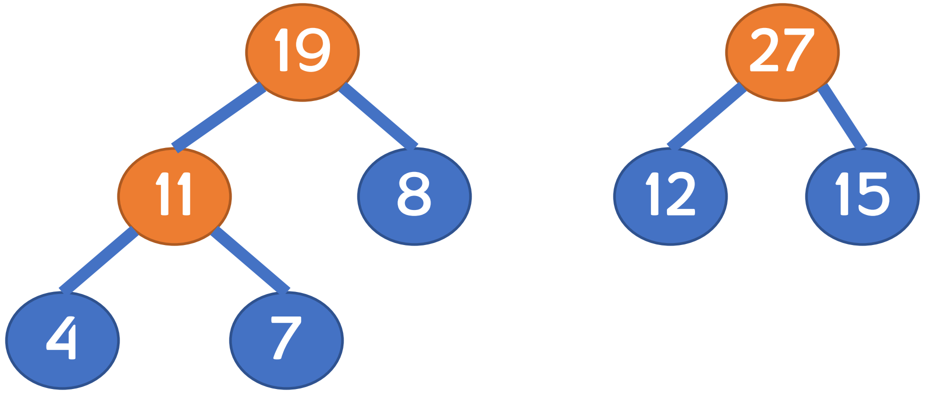
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**



Energy cost: 11 +19

Merge pebbles –greedy solution?

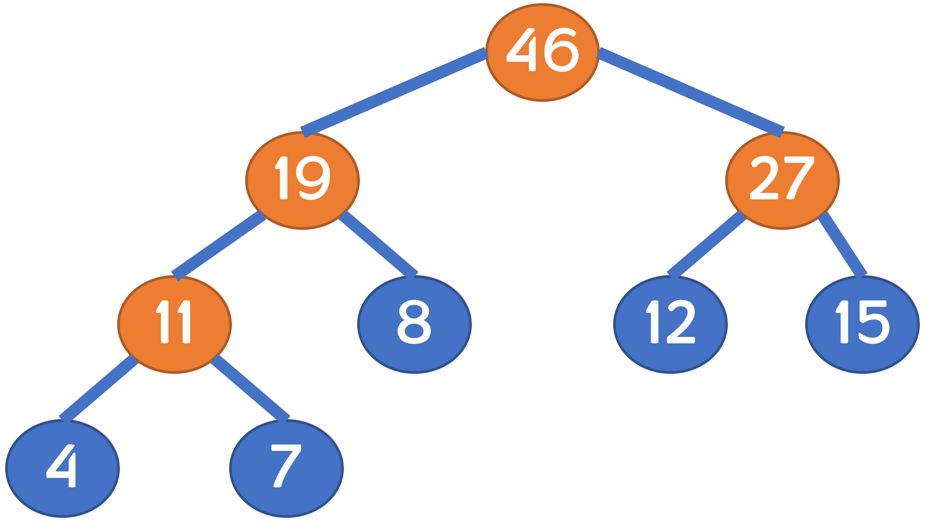
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**



Energy cost: 11 +19 +27

Merge pebbles –greedy solution?

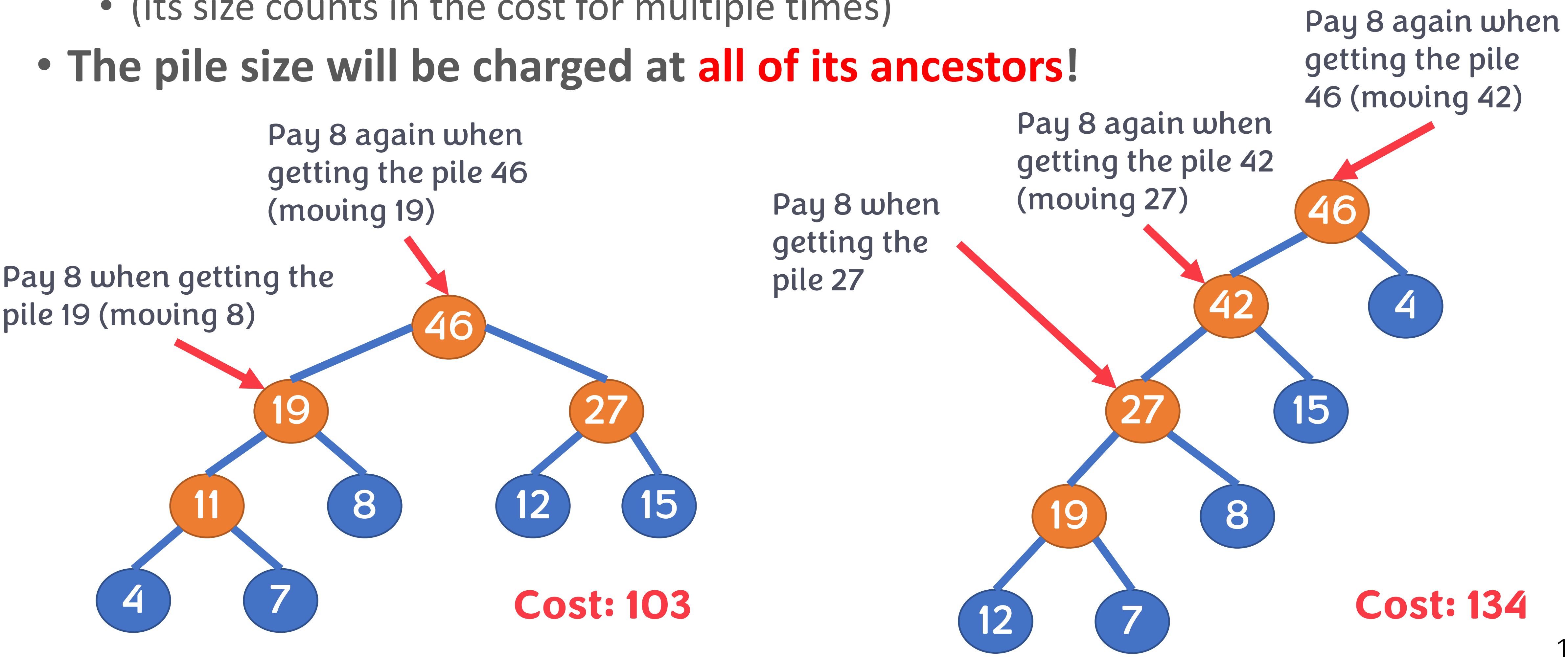
- **We have pebble piles and want to merge them into one pile, but**
  - We can only merge two of them at a time
  - Merging two piles of size a and b cost you a+b units of energy
- **Always merge the two with the fewest pebbles!**



Energy cost: 11 +19 +27 +46 **=103**

Merge pebbles – Why greedy is good?

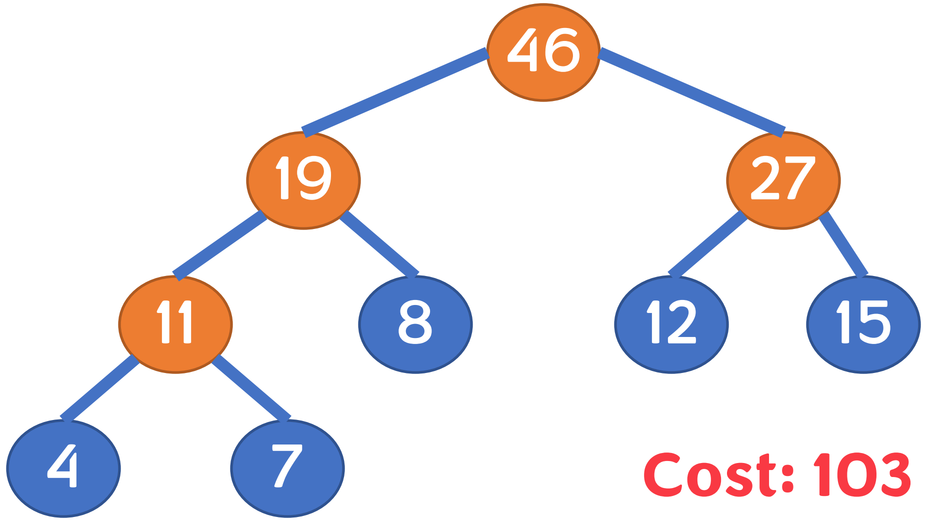
- **You may need to move a pile multiple times**
  - (its size counts in the cost for multiple times)
- **The pile size will be charged at all of its ancestors!**



Merge pebbles – Why greedy is good?

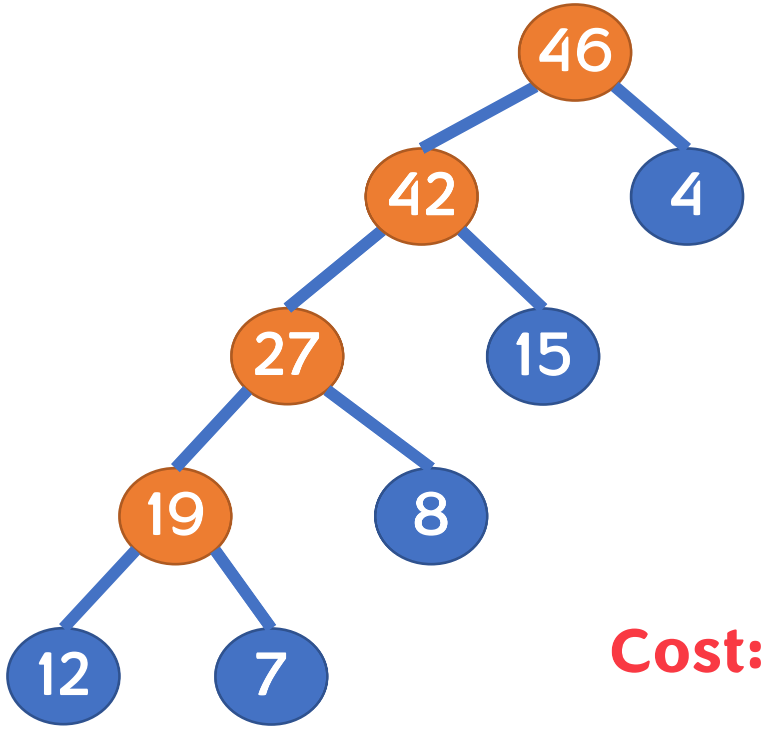
- **You may need to move a pile multiple times**
  - (its size counts in the cost for multiple times)
- **The pile size will be charged at all of its ancestors!**
- **How many times do you need to move the pile 8?**
  - The depths of it! (the number of ancestors)

Total cost: $4 \times 3 + 7 \times 3 + 8 \times 2 + 12 \times 2 + 15 \times 2 = 103$



Cost: 103

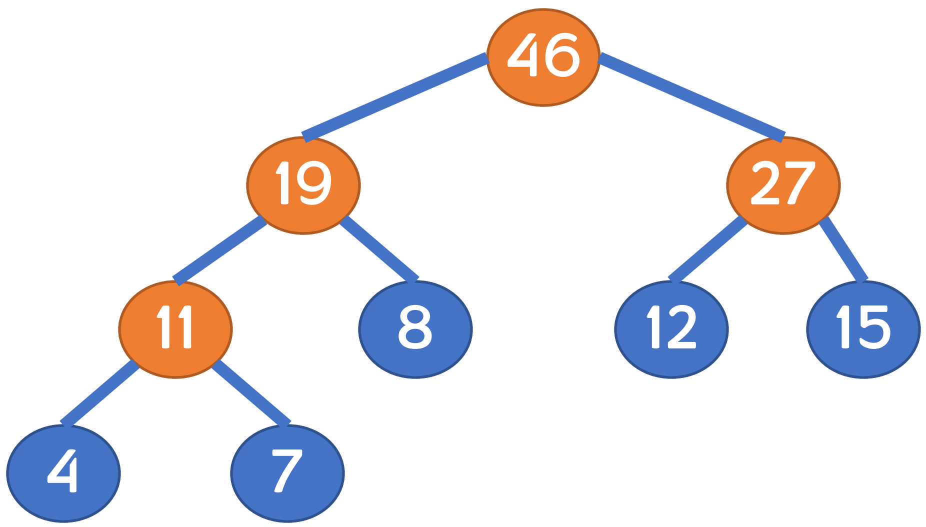
Total cost:
$4 \times 1 + 7 \times 4 + 8 \times 3 + 12 \times 4 + 15 \times 2 = 134$



Cost: 134

Merge pebbles – Why greedy is good?

- **You may need to move a pile multiple times**
  - (its size counts in the cost for multiple times)
- **The pile size will be charged at all of its ancestors!**
- **How many times do you need to move the pile 8?**
  - The depths of it! (the number of ancestors)
- $cost = \sum_{leaf\ t \in T} t \times d(t)$ $\quad d(t)$ is the depth of pile t in the merging tree



Total cost: $4 \times 3 + 7 \times 3 + 8 \times 2 + 12 \times 2 + 15 \times 2 = 103$

Merge pebbles – What is your greedy choice?

- $cost = \sum_{leaf\ t \in T} t \times d(t)$ $\quad d(t)$ is the depth of pile t in the merging tree

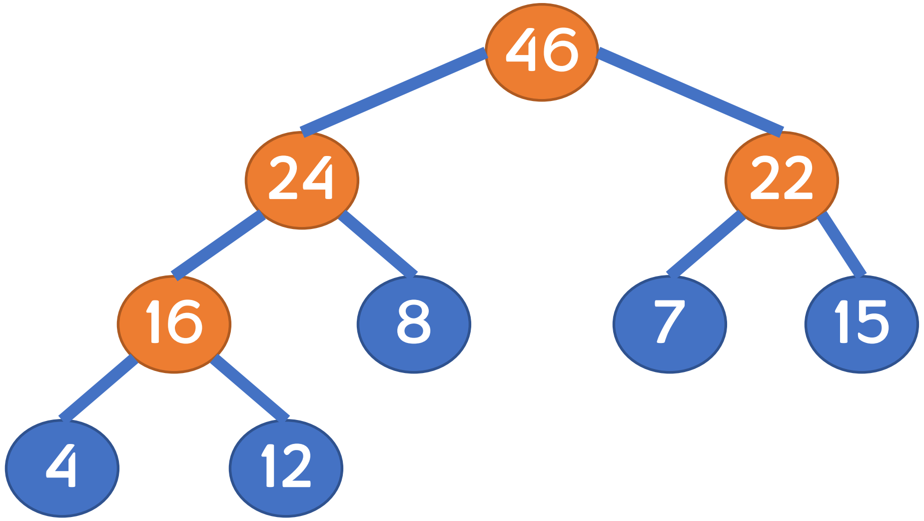
- The smaller a value is, the deeper we want to put it…

- But when we merge bottom-up, how can we know the depth?

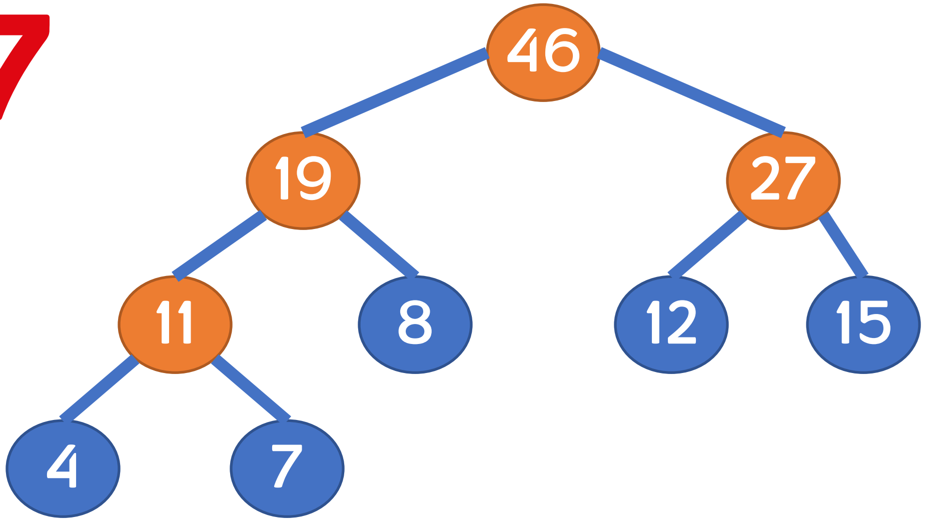
- Greedy choice: Make the two piles with the fewest pebbles as siblings!
  - They should also be in the deepest level

Greedy choice: Make the two piles with the fewest pebbles as siblings

- $cost = \sum_{leaf\ t \in T} t \times d(t)$ $\quad d(t)$ is the depth of pile t in the merging tree
- What happens if 4 and 7 are not siblings (more generally, x and y)?
- Find the deeper one (assume it is x), and swap x's sibling with y
- What happens if they are at the same level?



−12+7

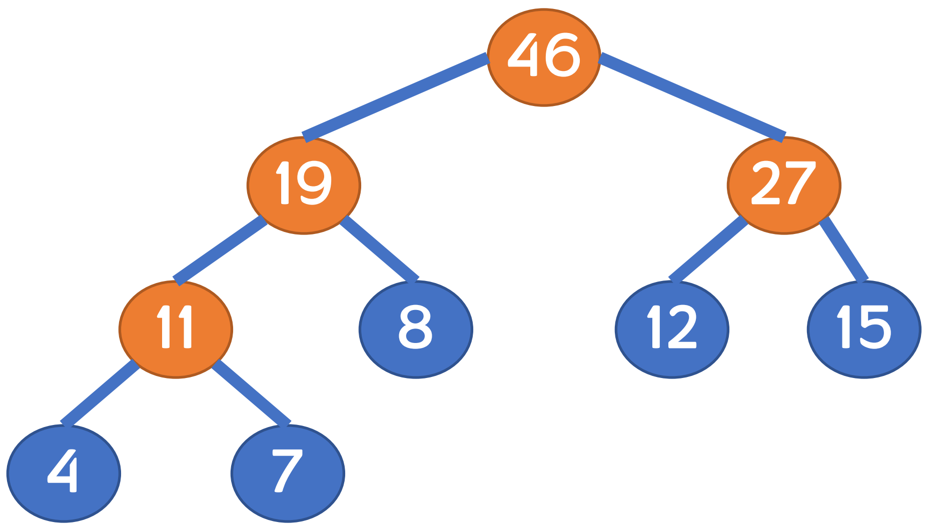


Total cost: $4 \times 3 + {\color{red}7 \times 2} + 8 \times 2 + {\color{red}12 \times 3} + 15 \times 2 = 108$

Total cost: $4 \times 3 + {\color{red}7 \times 3} + 8 \times 2 + {\color{red}12 \times 2} + 15 \times 2 = 103$

Merge pebbles – Why greedy is good? (intuitively)

- $cost = \sum_{leaf\ t \in T} t \times d(t)$ $\quad d(t)$ is the depth of pile t in the merging tree



Total cost: $4 \times 3 + 7 \times 3 + 8 \times 2 + 12 \times 2 + 15 \times 2 = 103$

- **Should make two smallest piles siblings**
- **We can always merge them first**
- **Optimal substructure: the problem size decreases by 1**
  - **n-1 piles of pebbles to merge, minimize energy**
- **A formal prove is in the textbook [CLRS16.3]**

Merge pebbles – Why greedy is good?

- **You may need to move a pile multiple times (its size counts in the cost for multiple times)**
- **The pile size will be charged at all of its ancestors!**
- $cost = \sum_{leaf\ t \in T} t \times d(t)$ $\quad d(t)$ is the depth of pile t in the merging tree
- **We should make small piles as deep as possible**
  - The deepest two must be the smallest two piles
  - Then.. Why don't we merge them first?
- **Optimal substructure: the problem size decreases by 1**
  - n-1 piles of pebbles to merge, minimize energy
- **(a similar) more formal prove could be found in the textbook**

Optimal substructure

- **After merging two piles x and y into one, we get a new pile $x + y$**
- **Should just merging the $n - 1$ piles with the optimal solution!**

- **Also prove by contradiction: If we don't use the optimal solution S_{opt} for the $n - 1$ piles, what will happen?**
  - We should be able to get a better solution by changing to the optimal solution S_{opt} for the $n - 1$ piles

- **Proof in Lemma 16.3**

However, why do we care about moving pebble piles???

Huffman Codes

Encoding

- **How data is represented?**
- **Fixed-size codes, e.g., ASCII**
  - A: 1000001   (65)
  - B: 1000010   (66)
- **Variable-size codes, e.g., Morse Codes**
  - A: ●▬
  - B: ▬●●●
  - E: ●
  - T: ▬

```
int x = ‘A’ + ‘B’;
cout << “x = ” << x;
```

x = 131

Example: Morse Code

“SOS” (sound):

● ● ● ▬ ▬ ▬ ● ● ●

A ● ▬
B ▬ ● ● ●
C ▬ ● ▬ ●
D ▬ ● ●
E ●
F ● ● ▬ ●
G ▬ ▬ ●
H ● ● ● ●
I ● ●

J ● ▬ ▬ ▬
K ▬ ● ▬
L ● ▬ ● ●
M ▬ ▬
N ▬ ●
O ▬ ▬ ▬
P ● ▬ ▬ ●
Q ▬ ▬ ● ▬
R ● ▬ ●

S ● ● ●
T ▬
U ● ● ▬
V ● ● ● ▬
W ● ▬ ▬
X ▬ ● ● ▬
Y ▬ ● ▬ ▬
Z ▬ ▬ ● ●

IJS: (..)(.---)(...)

STZE: (…)(-)(--..)(.)

IAGI: (..)(.-)(--.)(..)

VMS: (...-)(--)(...)

Dash (Dah) = 3 dots (dits)
Separators between letters: 3 dits
Separators between words: 7 dits

Prefix Codes

- No code is allowed to be a prefix of another code
- To encode, simply concatenate all the codes
- Decoding **does not entail any ambiguity**

0011101

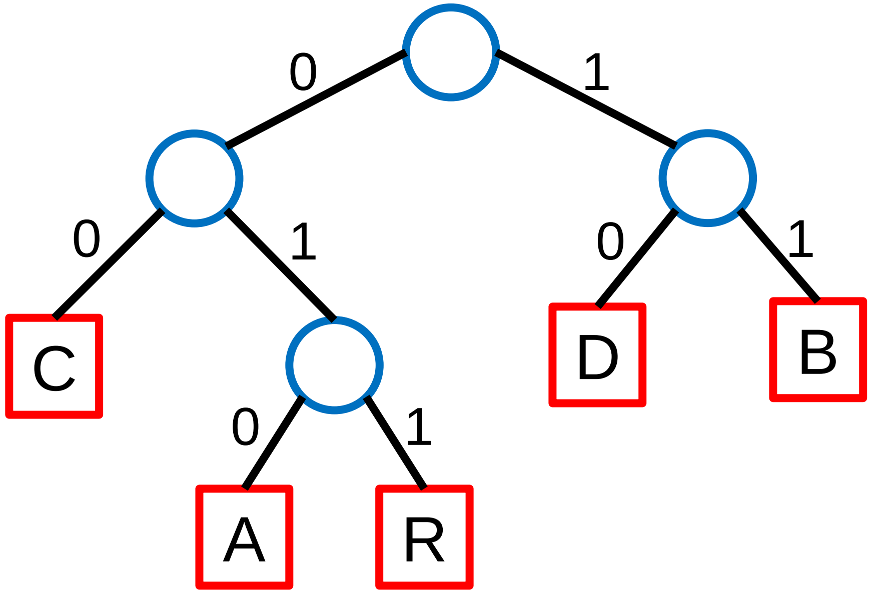
character	Prefix code	Non prefix code
A	00	00
B	01	001
C	101	11
D	100	111
E	11	01

AEC ADE?

BCE?

Trie

- We can use a trie to find prefix codes
- the characters are stored at the external nodes
- a left child (edge) means 0
- a right child (edge) means 1
- No code can be prefix of another code



A=010

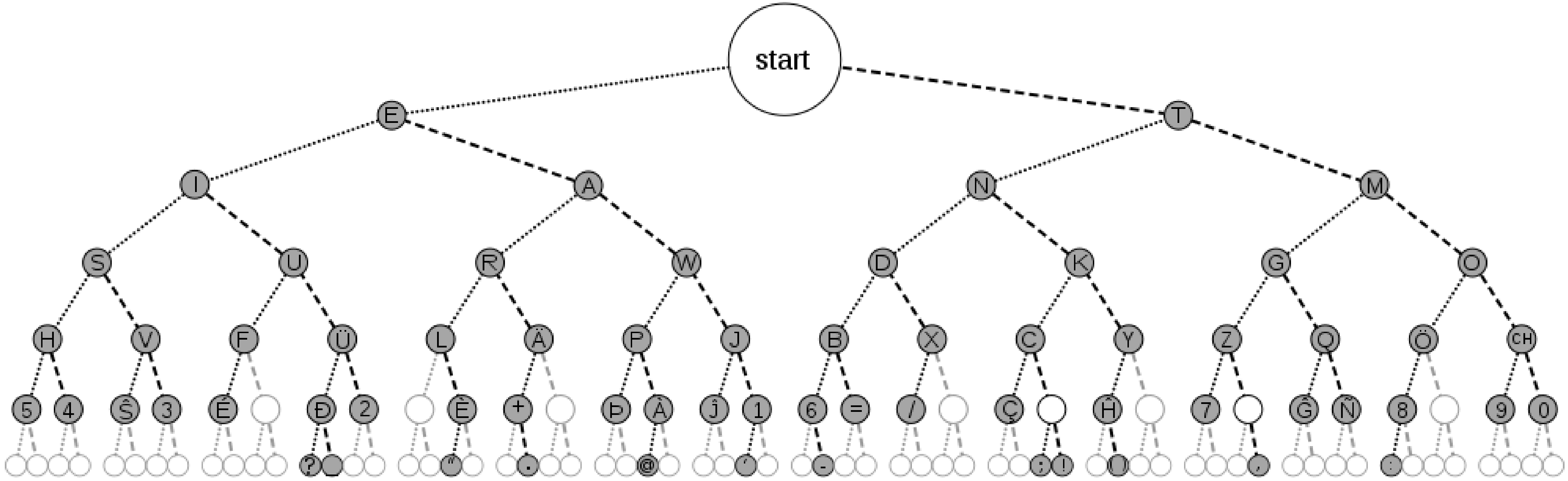
B=11

C=00

D=10

R=011

Morse code (not a prefix code)



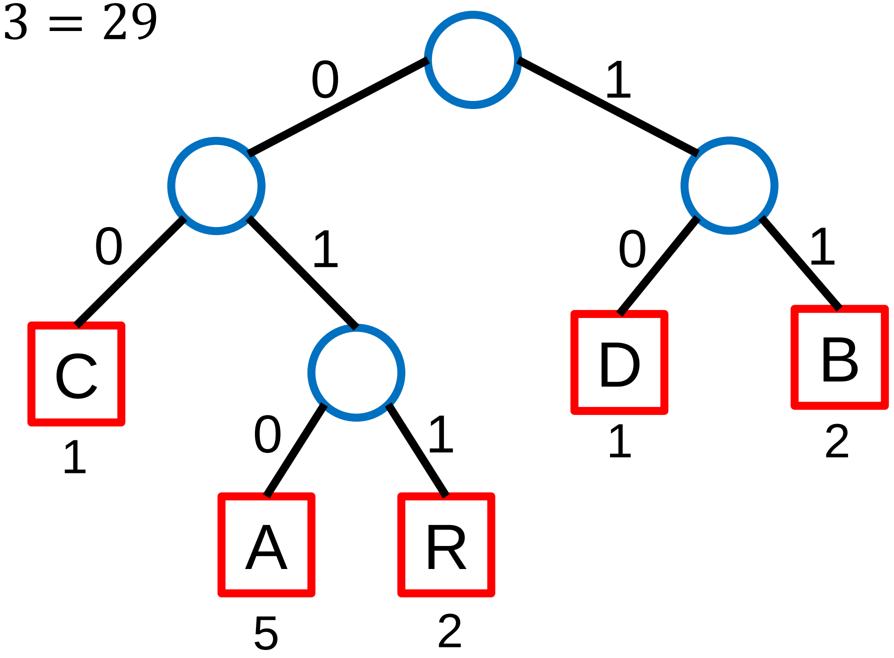
Source: Wikipedia

Example of Encoding

- Message: 'ABRACADABRA' (11 characters)
- Encoded message: '01011011010000101001011011010'
- Length: 29 bits

5 As
2 Bs
1 C
1 D
2 Rs

Total length: $5 \times 3 + 2 \times 2 + 1 \times 2 + 1 \times 2 + 2 \times 3 = 29$



A=010

B=11

C=00

D=10

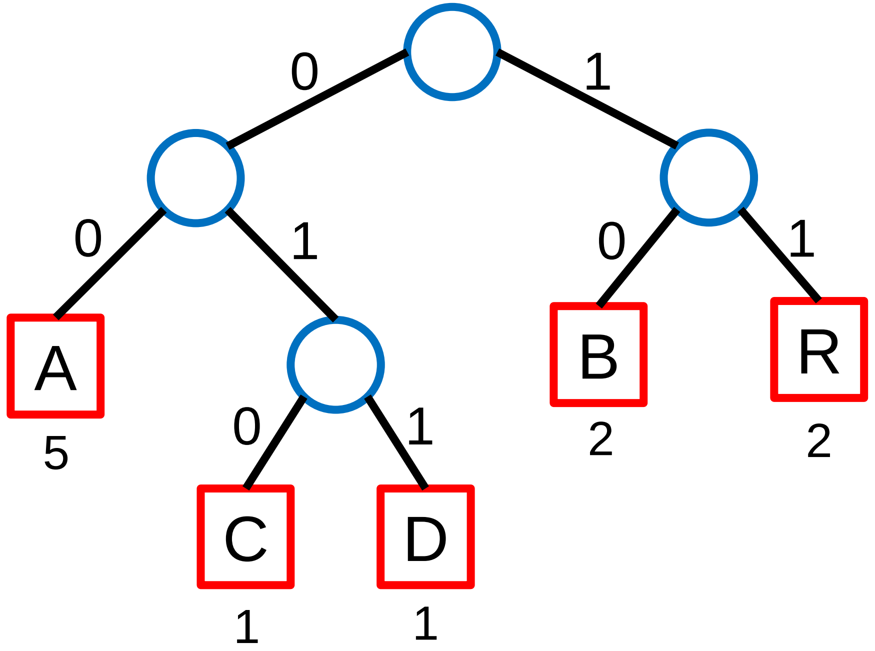
R=011

Example of Encoding

- Message: 'ABRACADABRA' (11 characters)
- Encoded message: '001011000100001100101100'
- Length: 24 bits

Total length: $5 \times 2 + 1 \times 3 + 1 \times 3 + 2 \times 2 + 2 \times 2 = 24$

5 As
2 Bs
1 C
1 D
2 Rs



A=00

B=10

C=010

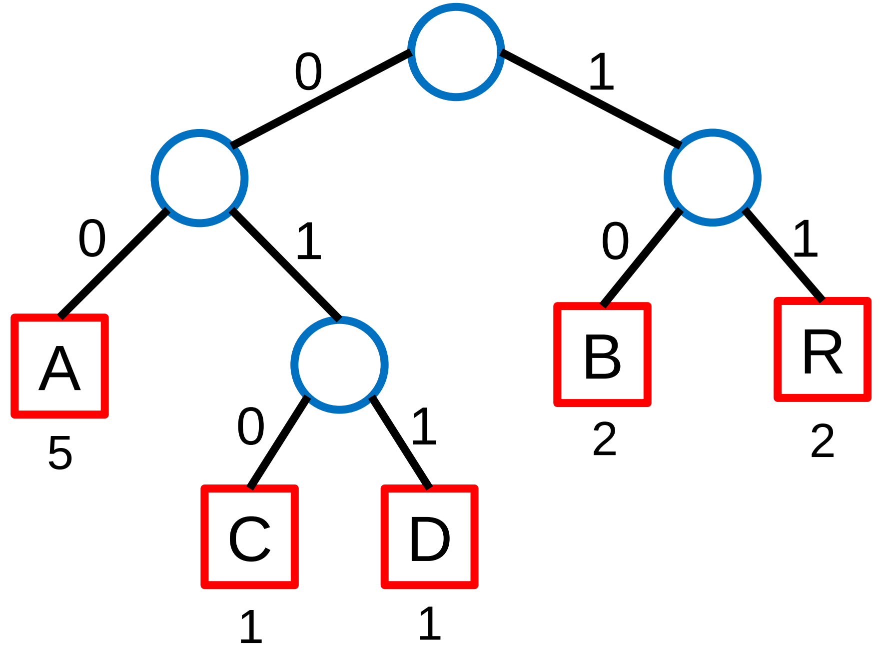
D=011

R=11

Optimal Encoding Problem

- Given a set C of n characters, for each character $c \in C$. Let $c.freq$ be the frequency of c in the file
- We would like to find a **prefix encoding** for each $c \in C$ with a length $d(c)$ such that we **minimize the total length**

The length of the code for character c is just its depth $d(c)$!

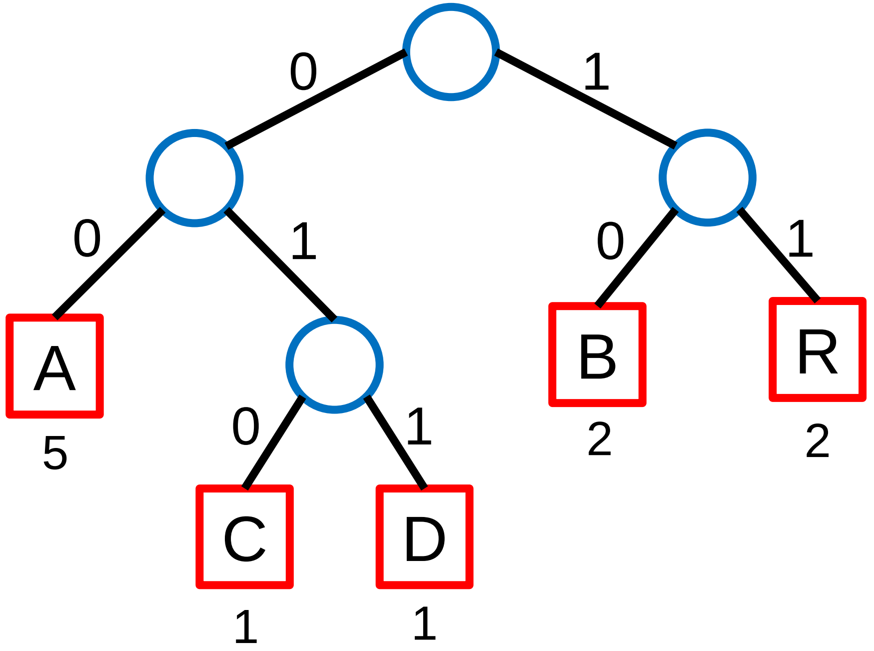


Total length: $5 \times 2 + 1 \times 3 + 1 \times 3 + 2 \times 2 + 2 \times 2 = 24$

Optimal Encoding Problem

- Given a set C of n characters, for each character $c \in C$. Let $c.freq$ be the frequency of c in the file
- We would like to find a **prefix encoding** for each $c \in C$ with a length $d(c)$ such that we **minimize the total length**

$$\boldsymbol{length} = \sum_{c \in C} c.freq \times \boldsymbol{d}(c)$$

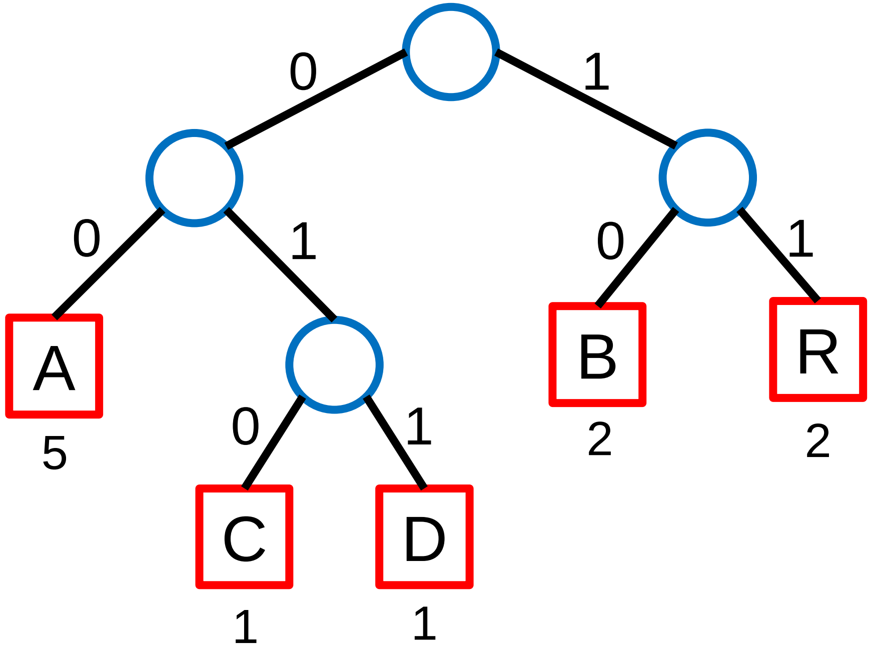


Total length: $5 \times 2 + 1 \times 3 + 1 \times 3 + 2 \times 2 + 2 \times 2 = 24$

Optimal Encoding Problem

- Given a set C of n characters, for each character $c \in C$. Let $c.freq$ be the frequency of c in the file.
- We would like to find a **prefix encoding** for each $c \in C$ with a length $d(c)$ such that we **minimize the total length**

$$\boldsymbol{length} = \sum_{c \in C} c.freq \times \boldsymbol{d}(c)$$



- That's the same with our **pebble merging** problem!
  - Frequency = initial pebble pile size
- Solution: **Huffman Codes**

Huffman codes

- **Find the two characters with the least frequency x and y**
  - Find to piles of pebbles with smallest size
- **Combine them in to one temporary character (internal node) with frequency $x + y$**
  - Combine them into one pile of size $x + y$
- **Repeat until there is only one node**
  - Repeat until there is only one pile

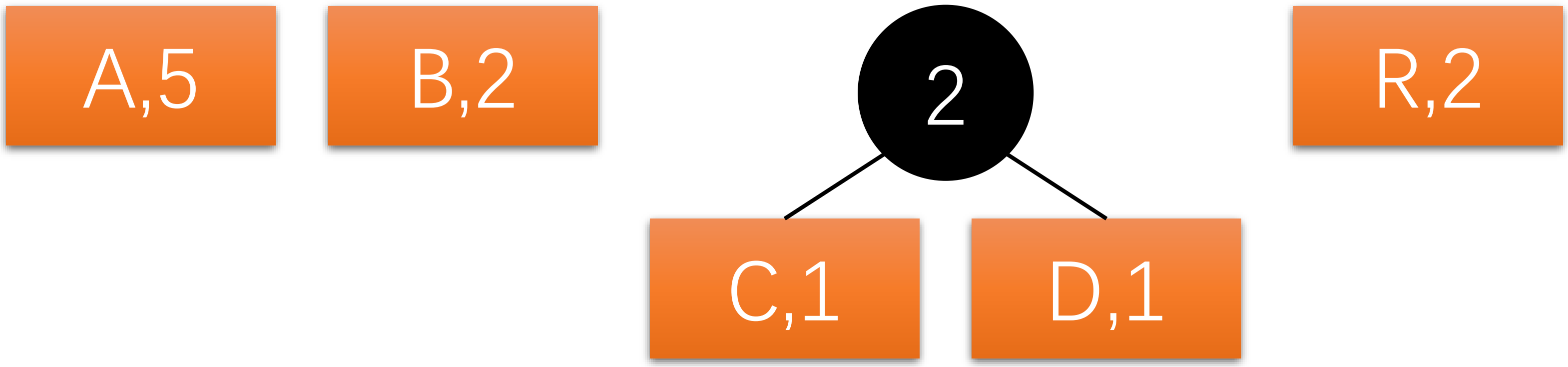
Example

"ABRACADABRA"

A,5	B,2	C,1	D,1	R,2

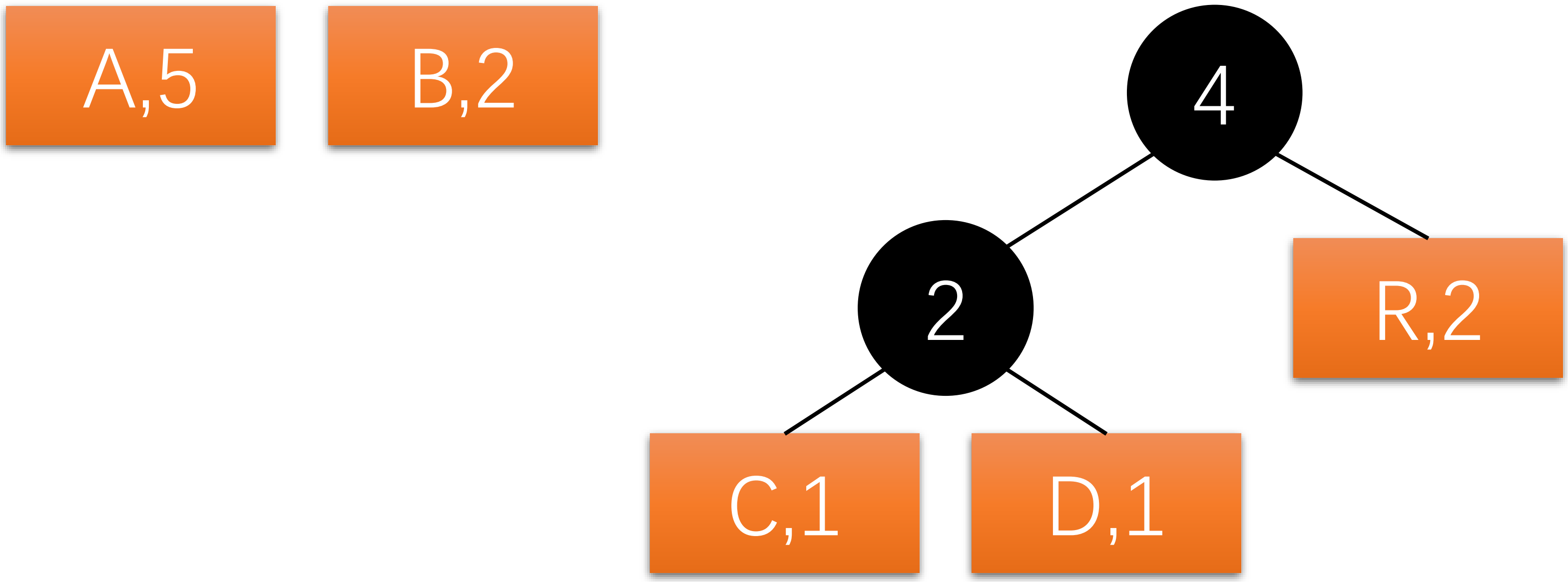
Example

“ABRACADABRA”

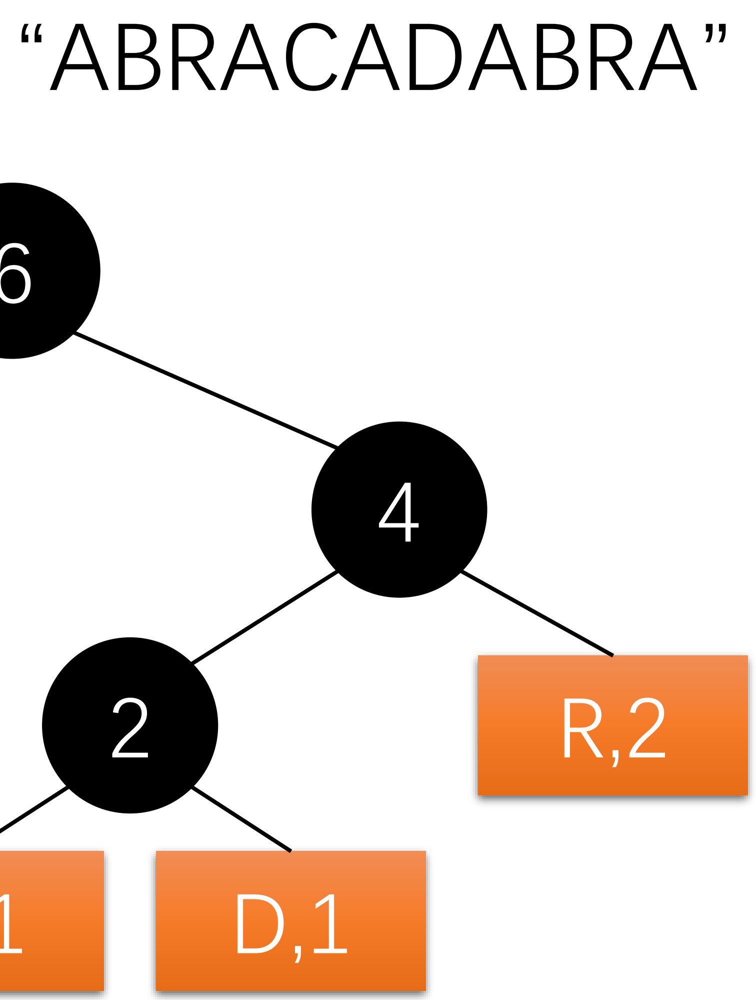


Example

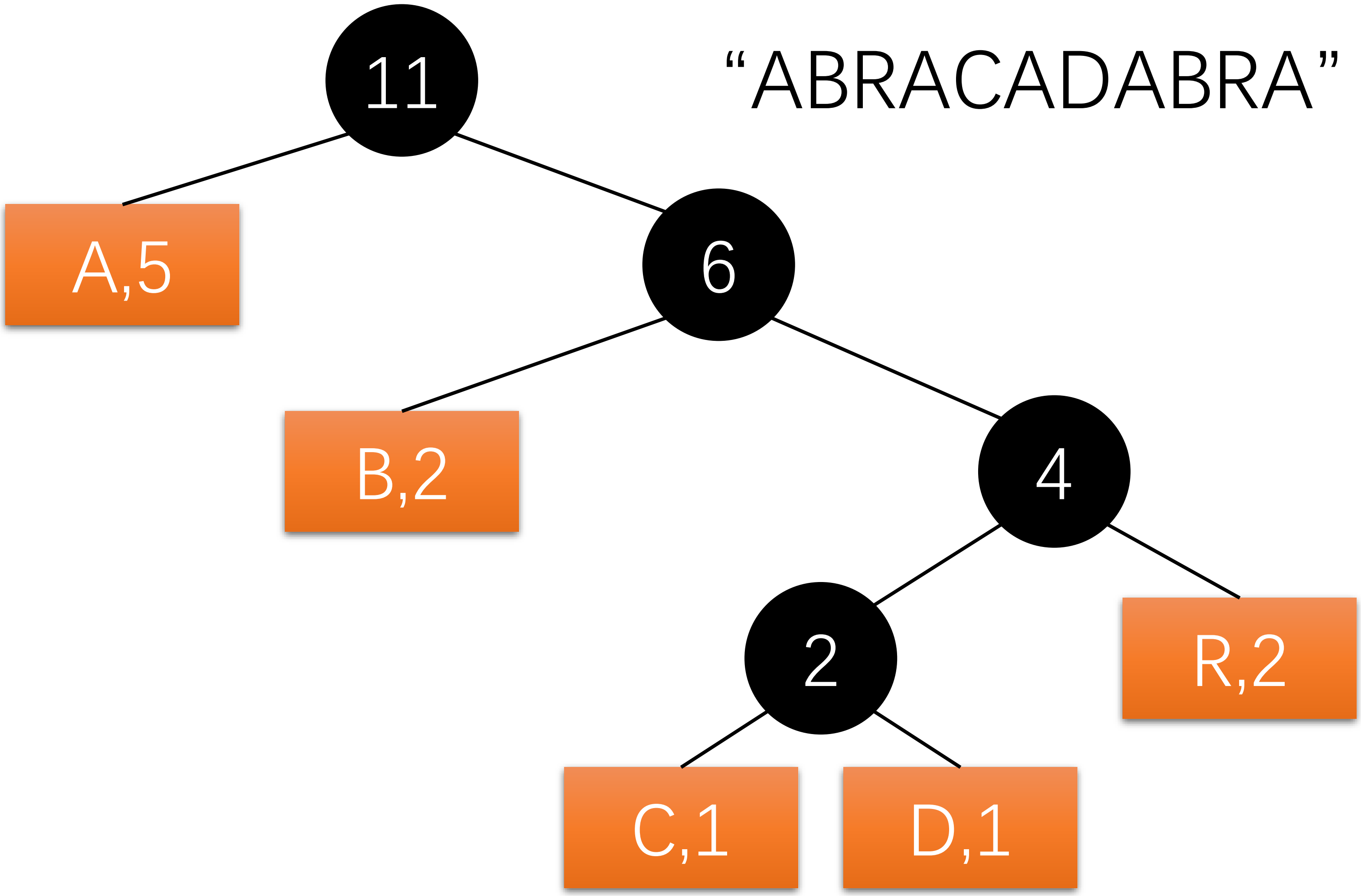
“ABRACADABRA”



Example

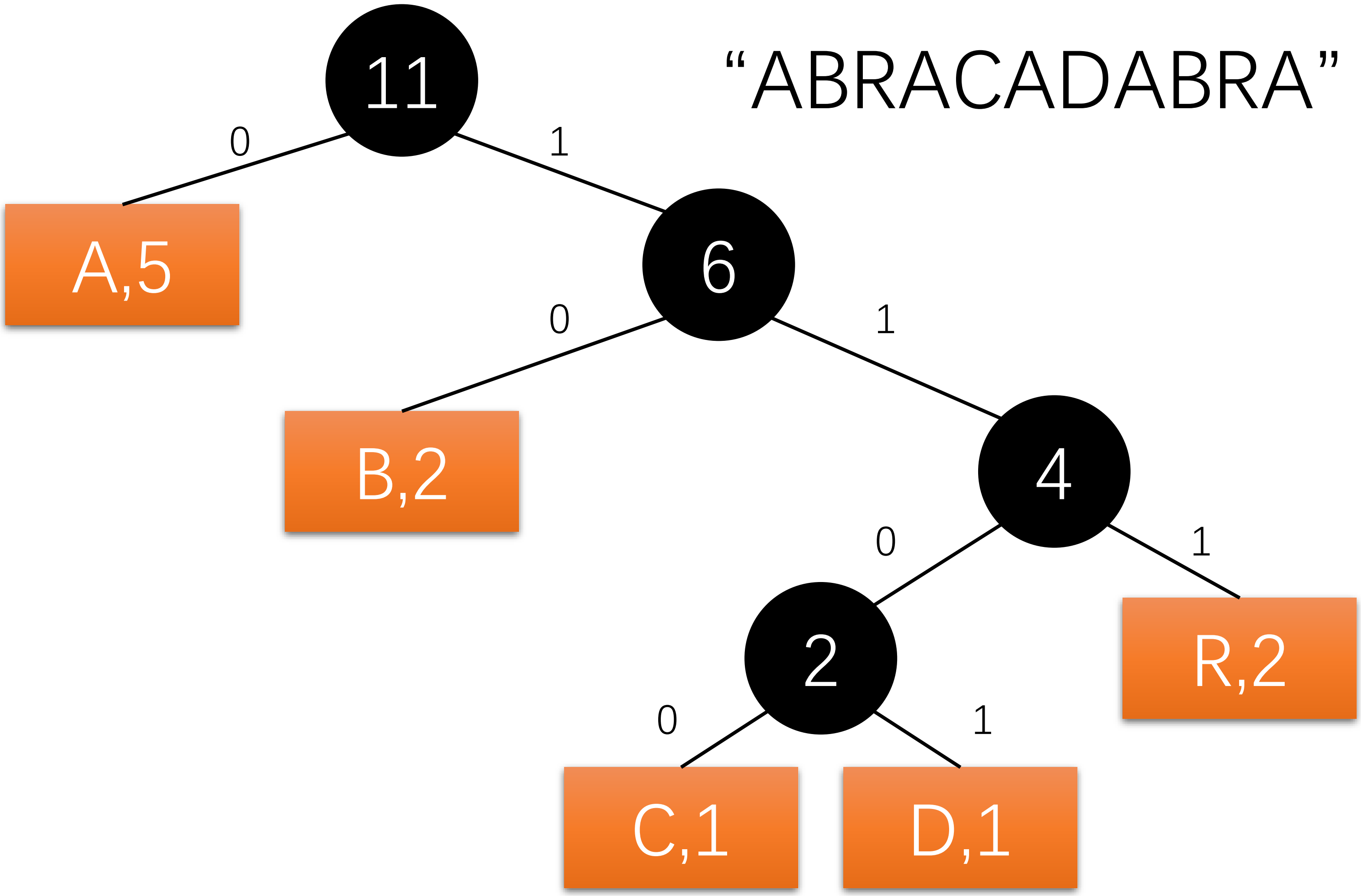


Example



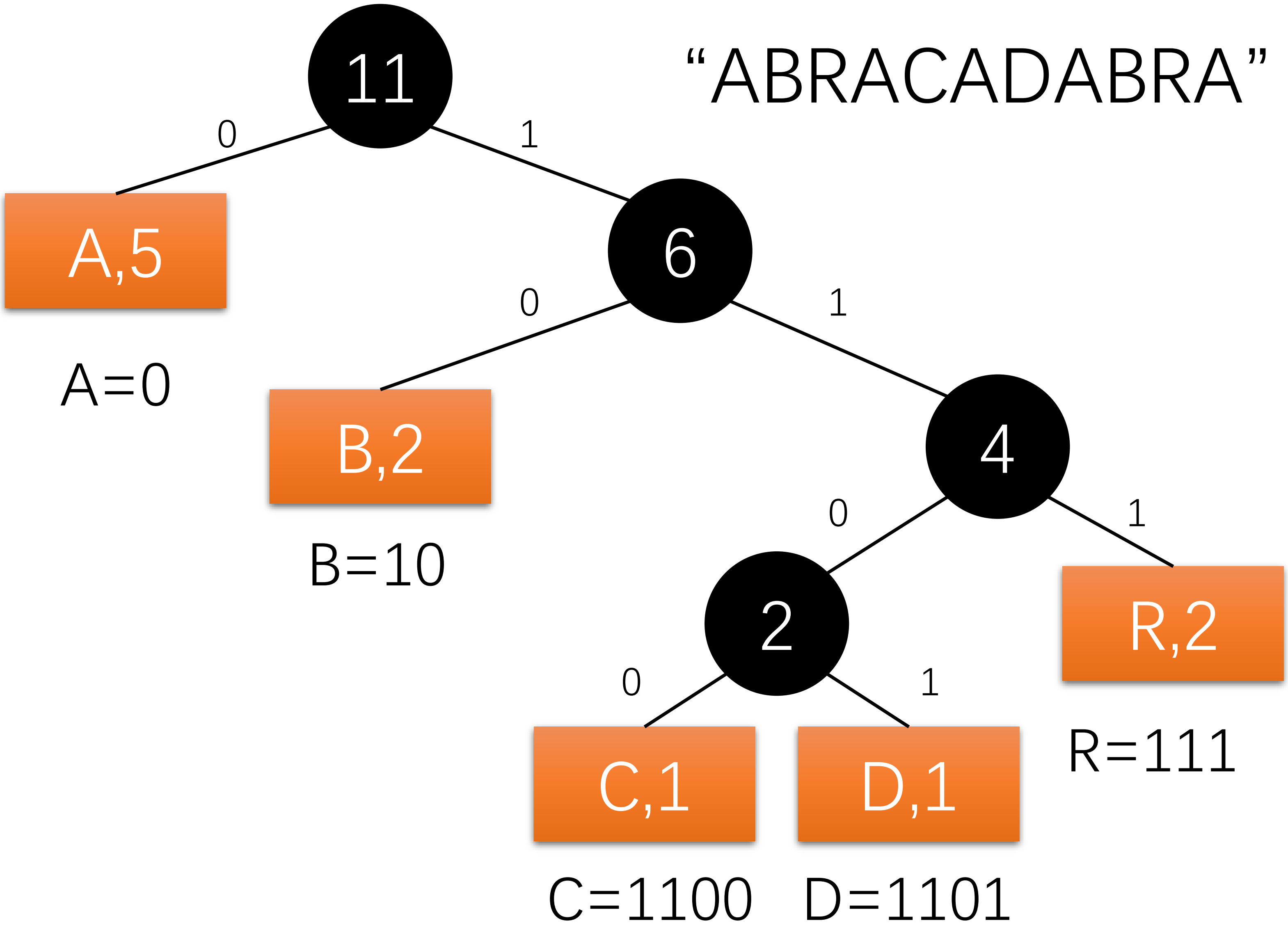
"ABRACADABRA"

Example



"ABRACADABRA"

Example



"ABRACADABRA"

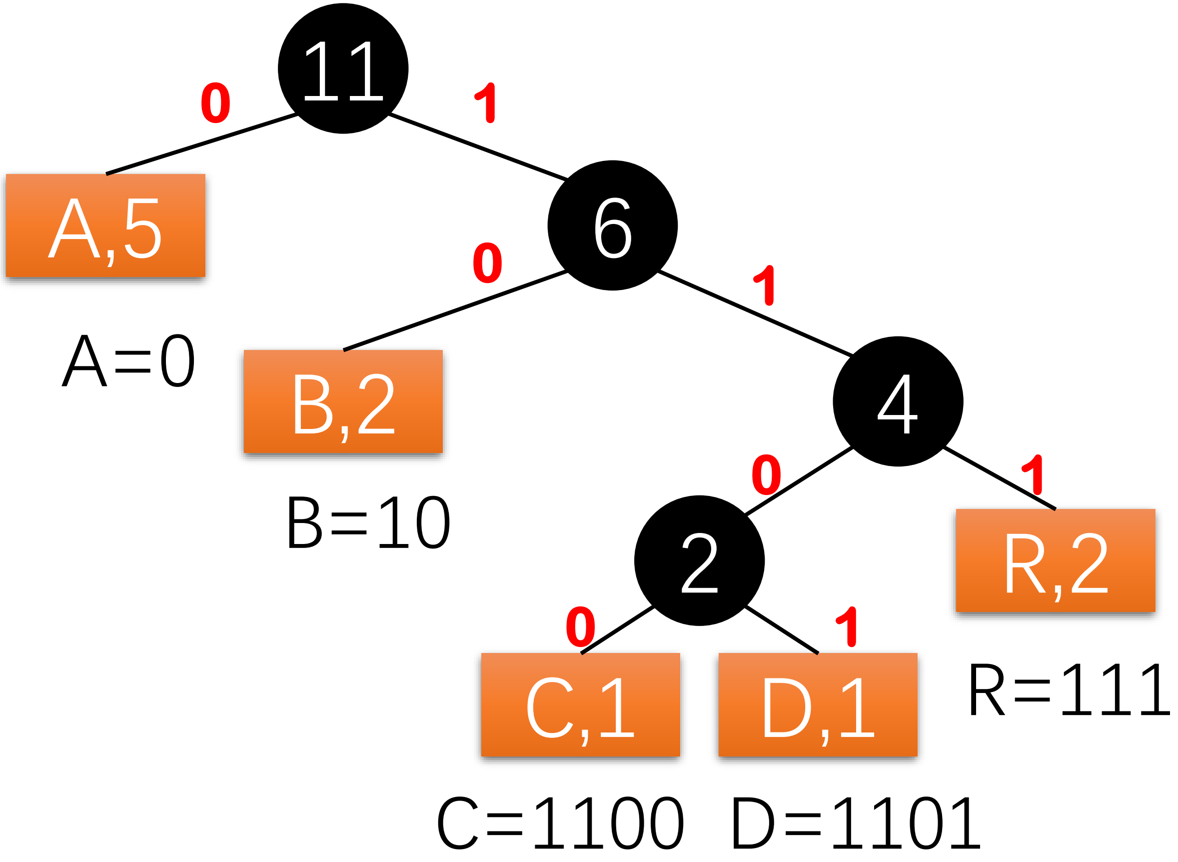
Encoding

"ABRACADABRA"

0 10 111 0 1100 0 1101 0 10 111 0

Length= 23

Optimal!



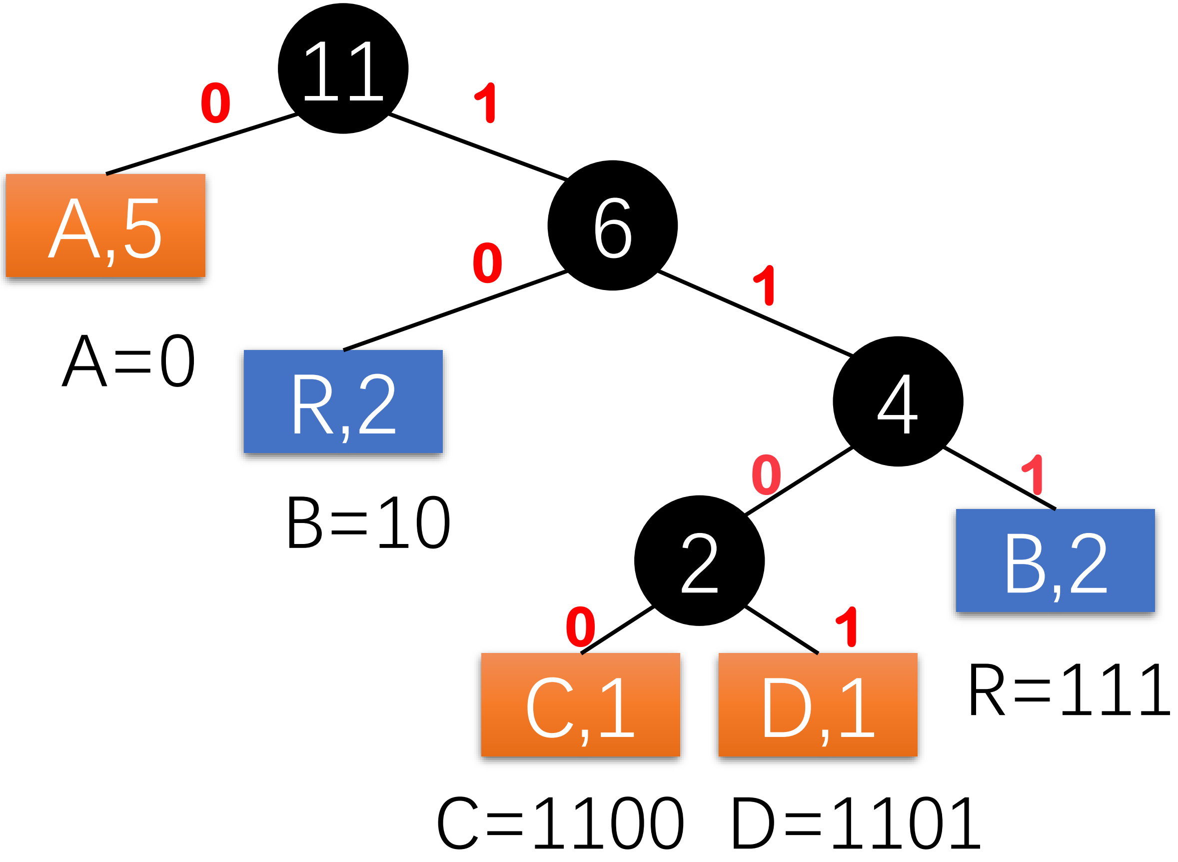
Encoding

"ABRACADABRA"

0 111 10 0 1100 0 1101 0 111 10 0

Length= 23

Optimal!



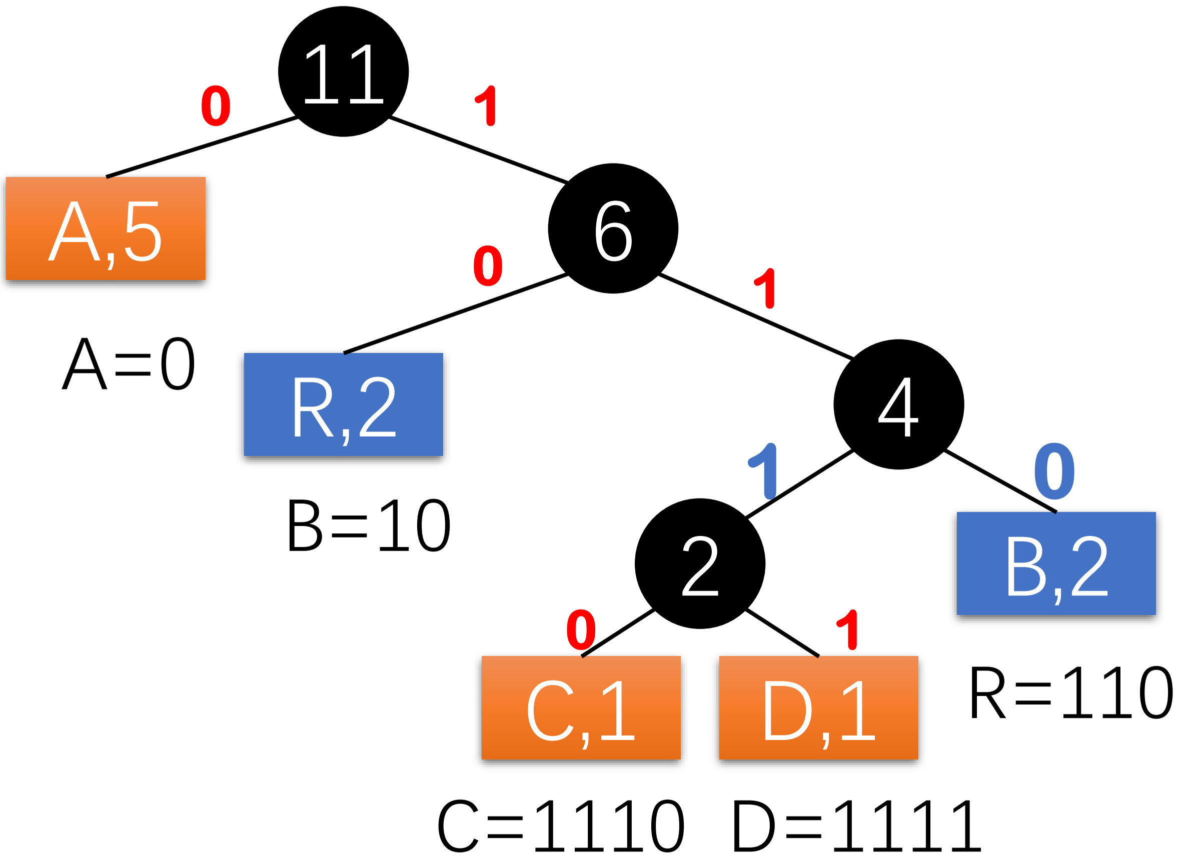
Encoding

"ABRACADABRA"

0 110 10 0 1110 0 1111 0 110 10 0

Length= 23

Optimal!



Construction of Huffman Tree

Note: Can also be done in linear time with pre-sorted frequencies

- **Huffman(C)**
  - n=|C|
  - Q=C // construct a priority queue of all character's frequency → $O(n \log n)$
  - for i = 1 to n-1
    - allocate a new node z
    - z.left = x = Extract-Min(Q)
    - z.right = y = Extract-Min(Q)
    - z.freq = x.freq + y.freq
    - Insert(Q, z)
  - return Extract-Min(Q) // Root of the tree

n

$\Theta(\log n)$

$$T(n) = \Theta(n \log n)$$

Assuming using a binary heap or a binary search tree

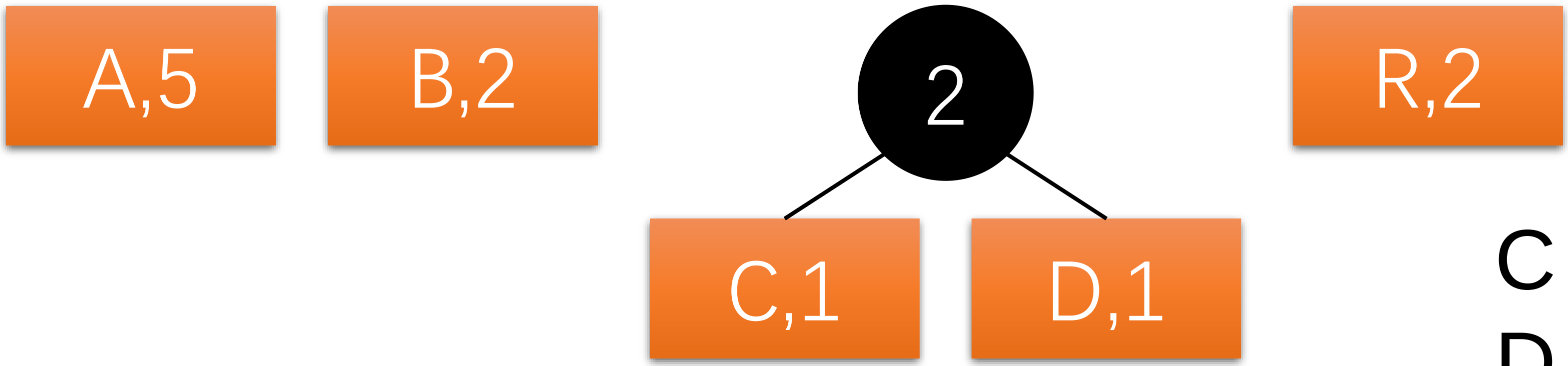
Optimality of Huffman Codes

- **Greedy-choice**
  - The greedy choice yields an optimal solution.
- **Optimal substructure**
  - The optimal solution for the bigger problem contains the optimal solution of the sub-problem.

- **Similar to the pebble merging**
- **Detailed proof in the textbook**

Optimal substructure

“ABRACADABRA”



C = $p0$
D = $p1$

Merging C and D

- They must share the same prefix p, and ending with 0 and 1, respectively
- Consider them as a whole: the frequency of p is 1+1=2.
- Create a new node (represents the prefix p) of frequency 2

Optimal substructure

"ABRApApABRA"

"ABRA(p0)A(p1)ABRA"

A,5 B,2 p, 2 R,2

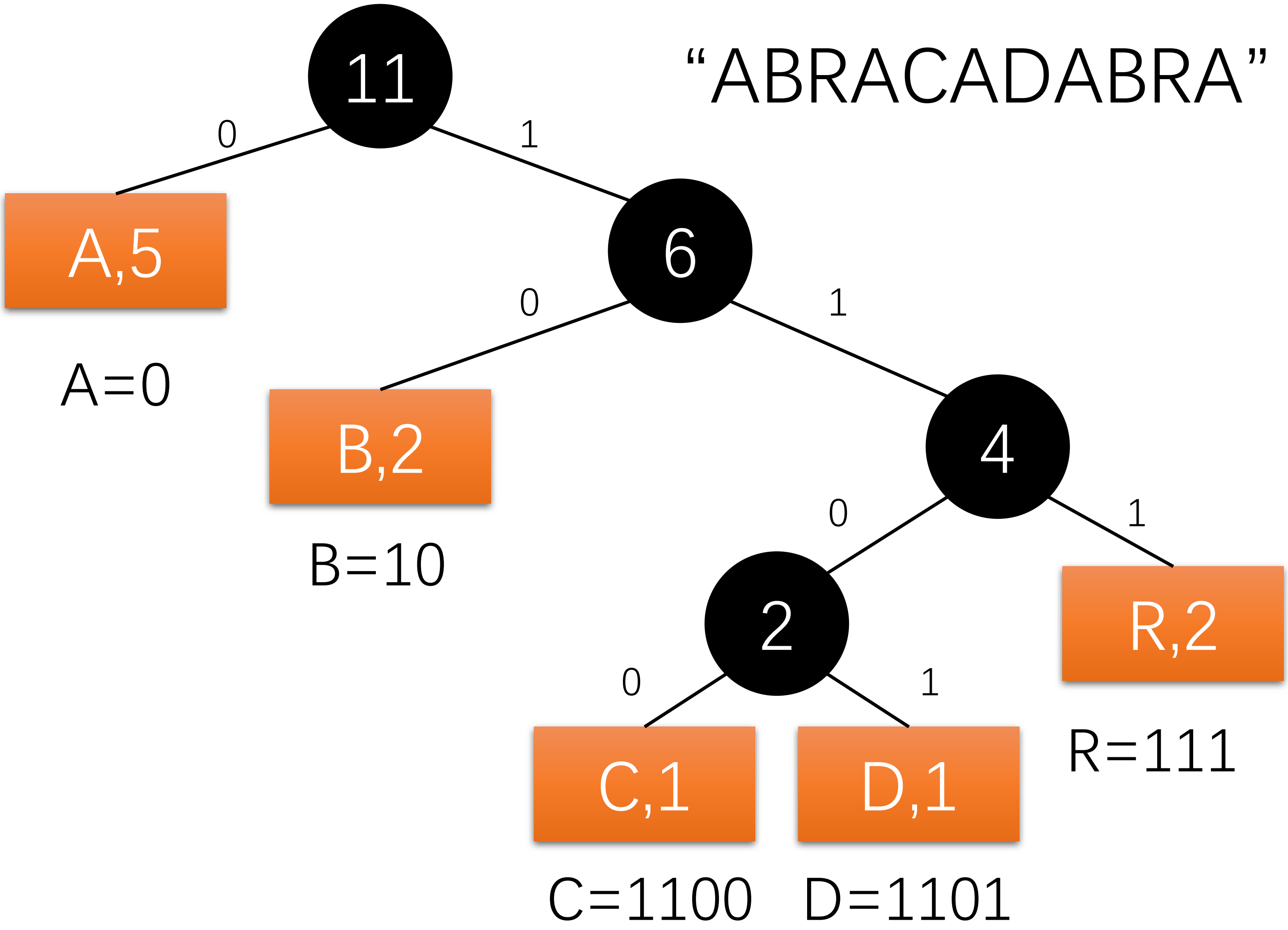
C = $p0$

D = $p1$

Merging C and D

- They must share the same prefix p, and ending with 0 and 1, respectively
- Consider them as a whole: the frequency of p is 1+1=2.
- Create a new node (represents the prefix p) of frequency 2
- Repeat the process – find the string for p recursively

Example



"ABRACADABRA"

Recall:
C = p0
D = p1
p is 110

Greedy Algorithms

- Among the commonly-used algorithm design strategies, greedy probably is the most intuitive and easiest to understand
- Once decision at a time
- When you need to make a decision, choose the “best” based on a certain criterion
- Not necessarily optimal, need to prove it

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!**

Well, sometimes greedy is not optimal, is it useless?

- They may still provide you with some nice features!
- We will talk about them more in the next lecture