Sorting Algorithms and Dynamic Programming

2022-10-07
Ping-Han Hsieh

Overview

- Data Structures:
  - Linked-List, Array
  - Stack, Queue
  - Union Find, Hash Table
  - Binary Search Tree, Heap
  - Graph

- Algorithms
  - Big-O Notation
  - Sorting
  - Graph Algorithms
  - Dynamic Programming

Sorting Algorithms

Bubble Sort (1)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

Bubble Sort (2)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n-i-1$ position
5. Repeat Step 1 n times

$i = 0$

Bubble Sort (3)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 0$

5 0 2 1 3 4

Bubble Sort (4)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 0$

Bubble Sort (5)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

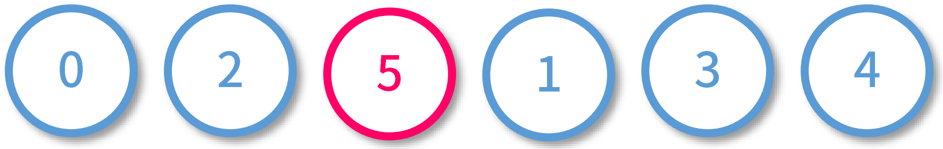
$i = 0$

Bubble Sort (6)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 0$

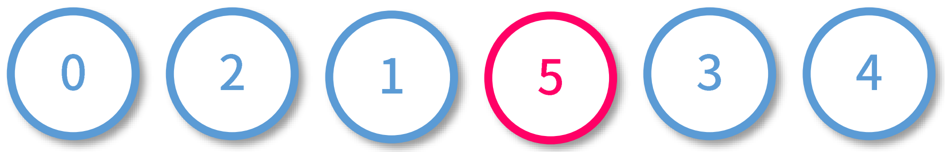


Bubble Sort (7)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 0$



Bubble Sort (8)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 0$

Bubble Sort (9)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 0$

Bubble Sort (10)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 1$

Bubble Sort (11)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 1$

Bubble Sort (12)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

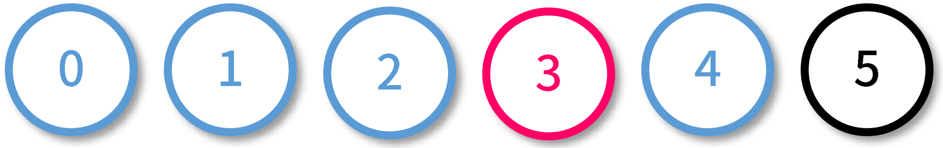
$i = 1$

Bubble Sort (13)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 1$

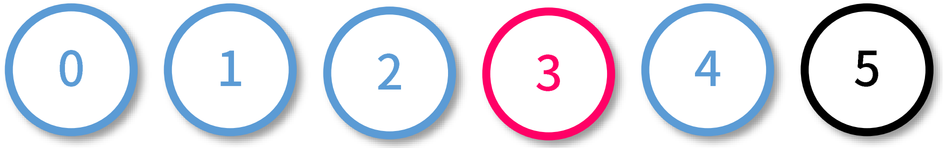


Bubble Sort (14)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 1$



Bubble Sort (15)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 1$

Bubble Sort (16)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

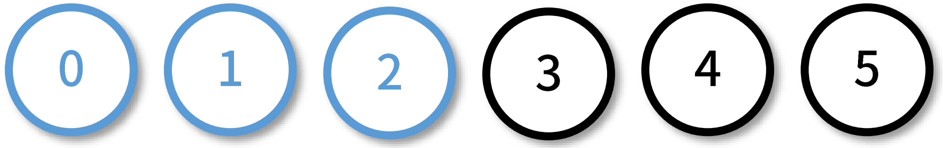
$i = 1$

Bubble Sort (17)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n-i-1$ position
5. Repeat Step 1 n times

$i=2$

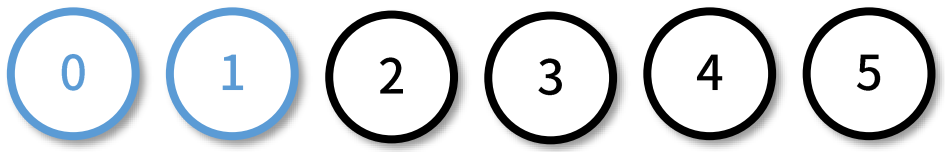


Bubble Sort (18)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 3$

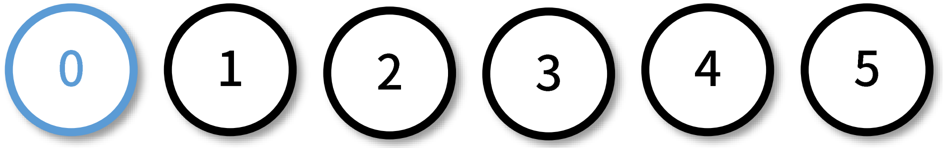


Bubble Sort (19)

Consider a series with n elements.

1. From the first position
2. Compare with the next element
3. Swap if the next element is smaller
4. Continue until $n - i - 1$ position
5. Repeat Step 1 n times

$i = 4$



Invariant
$n - i - 1$ to n is sorted.

Time complexity
$O(n^2)$.

Space complexity
$O(1)$.

Stable

Selection Sort (1)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

Selection Sort (2)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

$i = 0$

Selection Sort (3)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

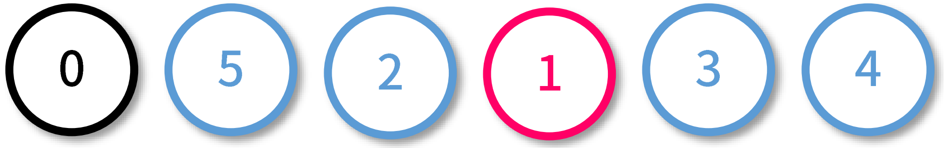
$i = 0$

Selection Sort (4)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

$i = 1$



Selection Sort (5)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

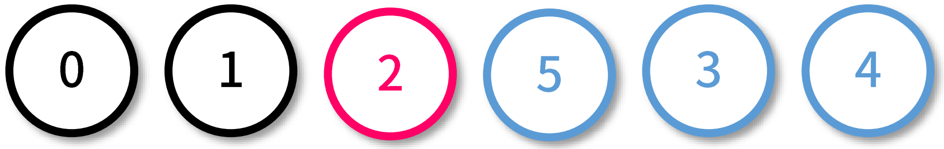
$i = 1$

Selection Sort (6)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

$i = 2$

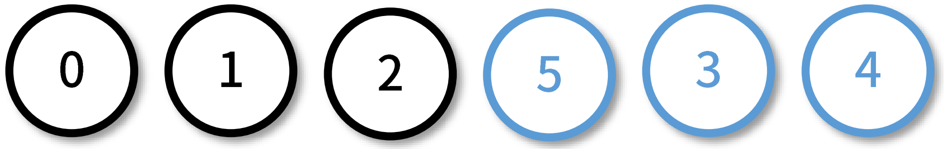


Selection Sort (7)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

$i = 3$

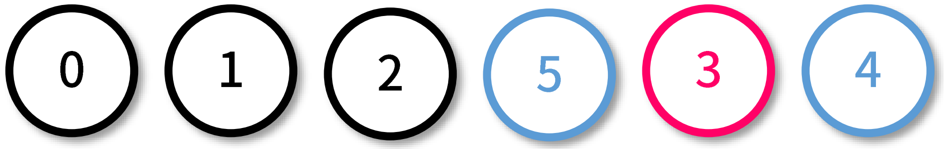


Selection Sort (8)

Consider a series with n elements.

1. Find the minimum in i to $n - 1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

$i = 3$

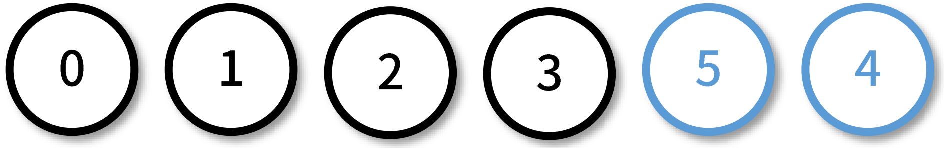


Selection Sort (9)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

$i = 3$

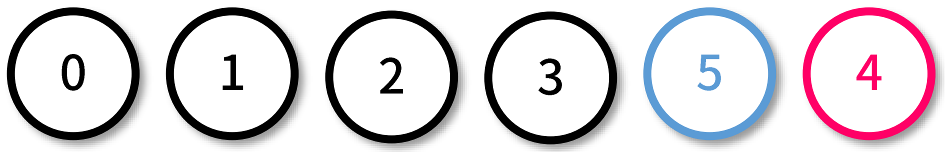


Selection Sort (10)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

$i = 4$

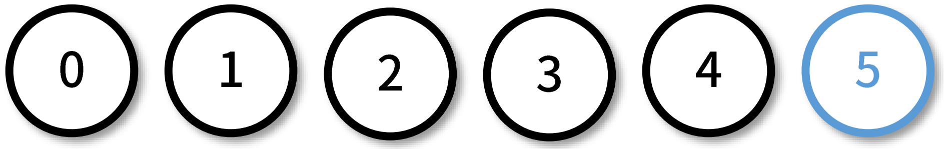


Selection Sort (11)

Consider a series with n elements.

1. Find the minimum in i to $n-1$ positions
2. Swap the minimum with the i-th element.
3. Repeat Step 1 n times

$i = 4$



Invariant
0 to i is sorted.

Time complexity
$O(n^2)$.

Space complexity
$O(1)$.

Unstable*

Quick Sort (1)

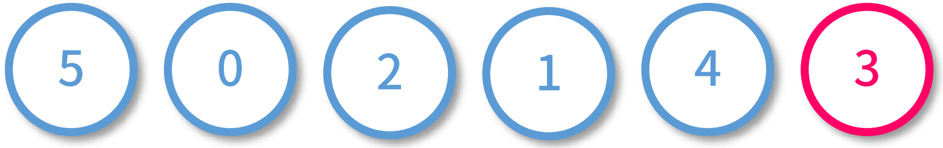
Consider a series with n elements.

1. Define the last element as the primary pivot.
2. From the first position (i)
    1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
    2. Otherwise
        1. If there's no secondary pivot, do nothing
        2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.

Quick Sort (2)

Consider a series with n elements.

1. Define the last element as the primary pivot.
2. From the first position (i)
    1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
    2. Otherwise
        1. If there's no secondary pivot, do nothing
        2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

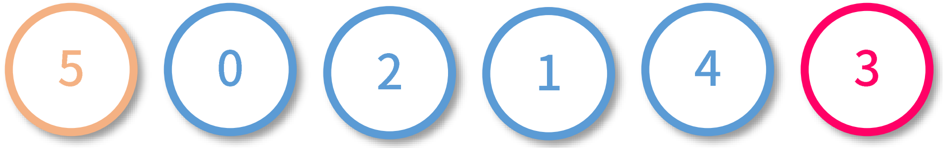
Consider a series with n elements.

1. Define the last element as the primary pivot.
2. From the first position (i)
   1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
   2. Otherwise
      1. If there's no secondary pivot, do nothing
      2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.

Quick Sort (2)

Consider a series with n elements.

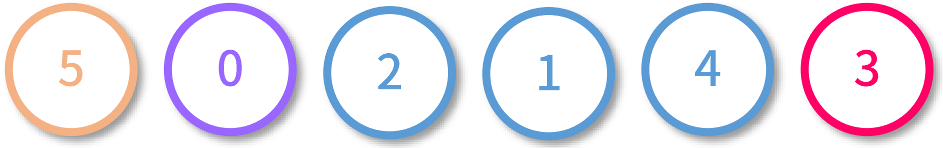
1. Define the last element as the primary pivot.
2. From the first position (i)
    1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
    2. Otherwise
        1. If there's no secondary pivot, do nothing
        2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

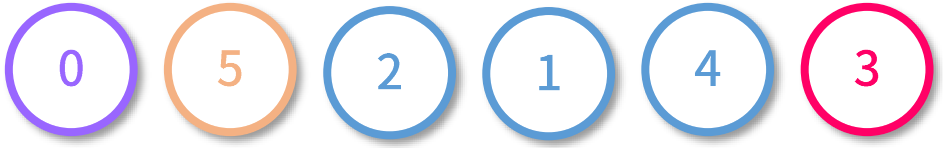
1. Define the last element as the primary pivot.
2. From the first position (i)
    1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
    2. Otherwise
        1. If there's no secondary pivot, do nothing
        2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

1. Define the last element as the primary pivot.
2. From the first position (i)
    1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
    2. Otherwise
        1. If there's no secondary pivot, do nothing
        2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

1. Define the last element as the primary pivot.
2. From the first position (i)
   1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
   2. Otherwise
      1. If there's no secondary pivot, do nothing
      2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.

Quick Sort (2)

Consider a series with n elements.

1. Define the last element as the primary pivot.
2. From the first position (i)
    1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
    2. Otherwise
        1. If there's no secondary pivot, do nothing
        2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.

Quick Sort (2)

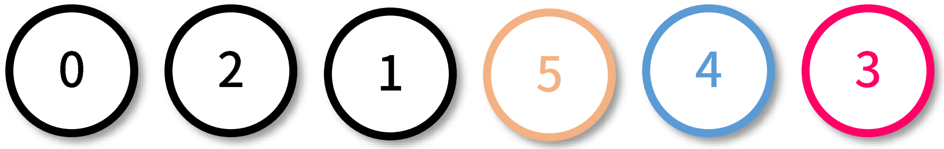
Consider a series with n elements.

1. Define the last element as the primary pivot.
2. From the first position (i)
   1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
   2. Otherwise
      1. If there's no secondary pivot, do nothing
      2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.

Quick Sort (2)

Consider a series with n elements.

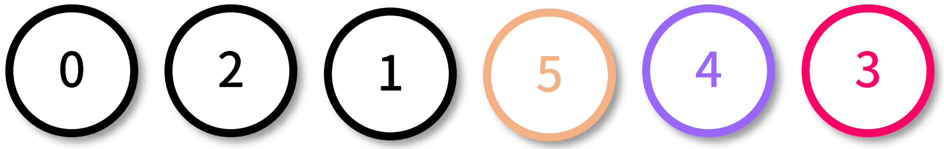
1. Define the last element as the primary pivot.
2. From the first position (i)
    1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
    2. Otherwise
        1. If there's no secondary pivot, do nothing
        2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

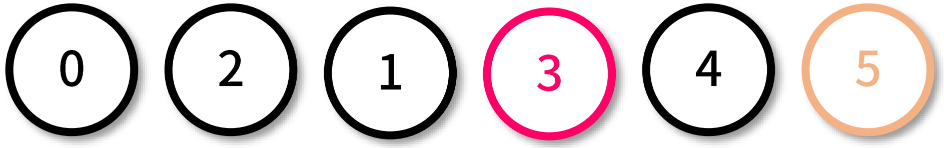
1. Define the last element as the primary pivot.
2. From the first position (i)
    1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
    2. Otherwise
        1. If there's no secondary pivot, do nothing
        2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

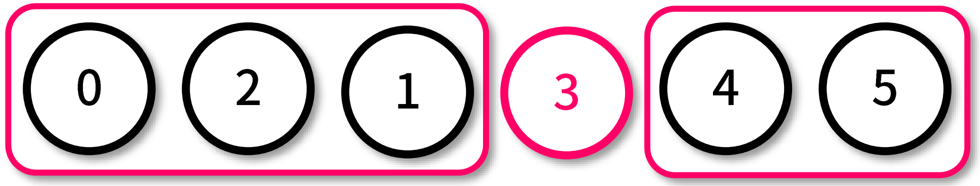
1. Define the last element as the primary pivot.
2. From the first position (i)
   1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
   2. Otherwise
      1. If there's no secondary pivot, do nothing
      2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

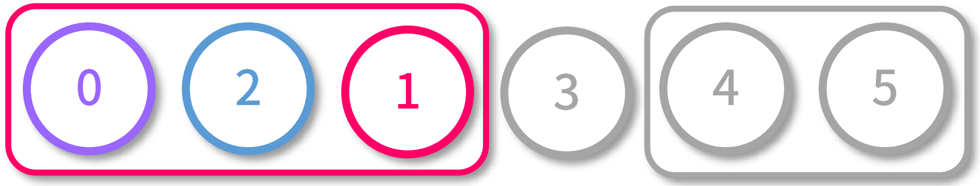
1. Define the last element as the primary pivot.
2. From the first position (i)
   1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
   2. Otherwise
      1. If there's no secondary pivot, do nothing
      2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

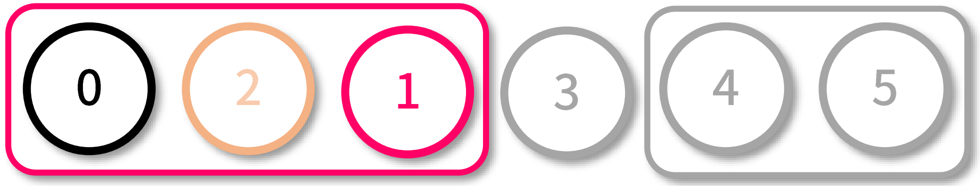
1. Define the last element as the primary pivot.
2. From the first position (i)
   1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
   2. Otherwise
      1. If there's no secondary pivot, do nothing
      2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

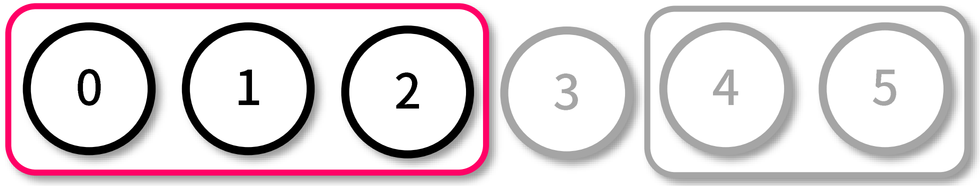
1. Define the last element as the primary pivot.
2. From the first position (i)
    1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
    2. Otherwise
        1. If there's no secondary pivot, do nothing
        2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

1. Define the last element as the primary pivot.
2. From the first position (i)
   1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
   2. Otherwise
      1. If there's no secondary pivot, do nothing
      2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.



Quick Sort (2)

Consider a series with n elements.

1. Define the last element as the primary pivot.
2. From the first position (i)
   1. If the element is larger than the pivot, set it to the secondary pivot j if it is not set.
   2. Otherwise
      1. If there's no secondary pivot, do nothing
      2. If there's a secondary pivot, swap with it and assign the secondary pivot to swapped position $+1$.
3. Swap the primary pivot with the secondary pivot.
4. Separate the series into two based on the primary pivot.
5. Repeat Step 1 for series on the left and right of the primary pivot.

Invariant
left subseries is smaller than the pivot,
right subseries is larger than the pivot

Time complexity
$O(n\log n)$.

Space complexity
$O(1)$.

Unstable

Merge Sort (1)

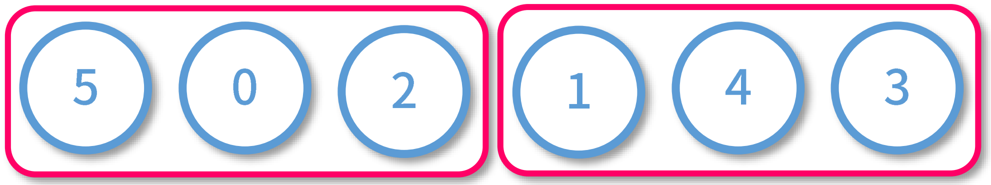
Consider a series with n elements.

1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.

Merge Sort (1)

Consider a series with n elements.

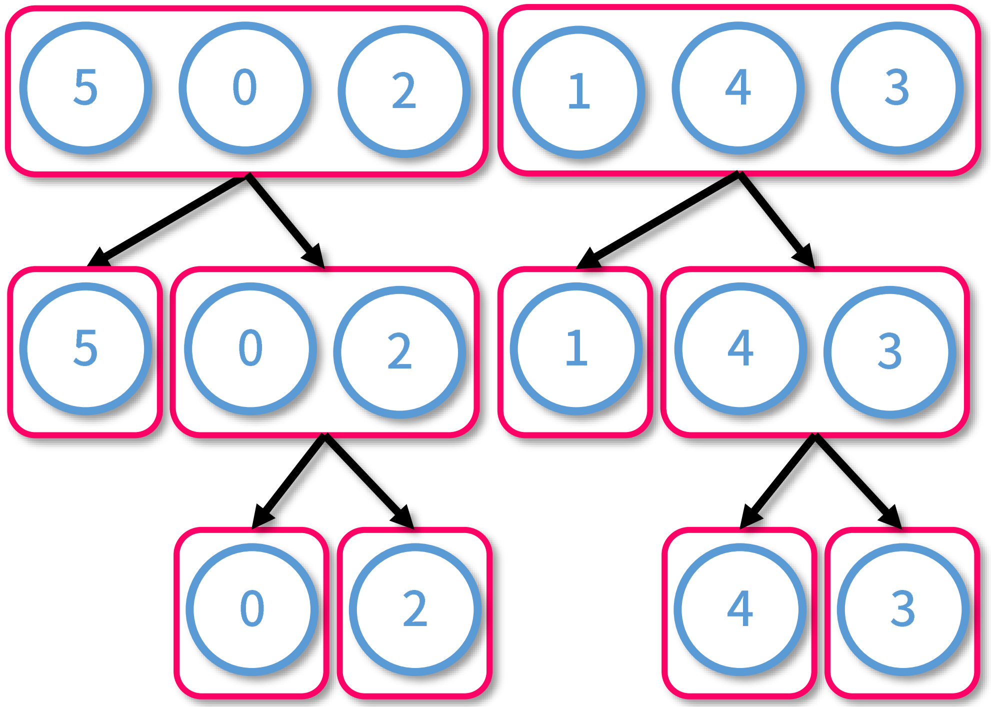
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
   1. Pick the smallest element in either left or right series one by one.
   2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

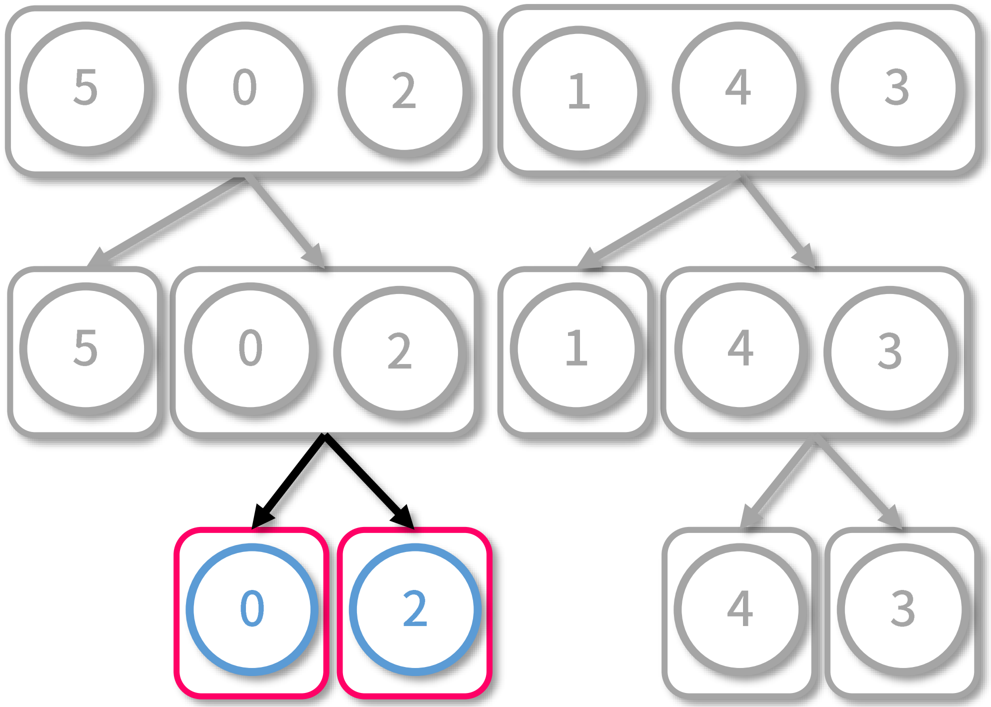
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

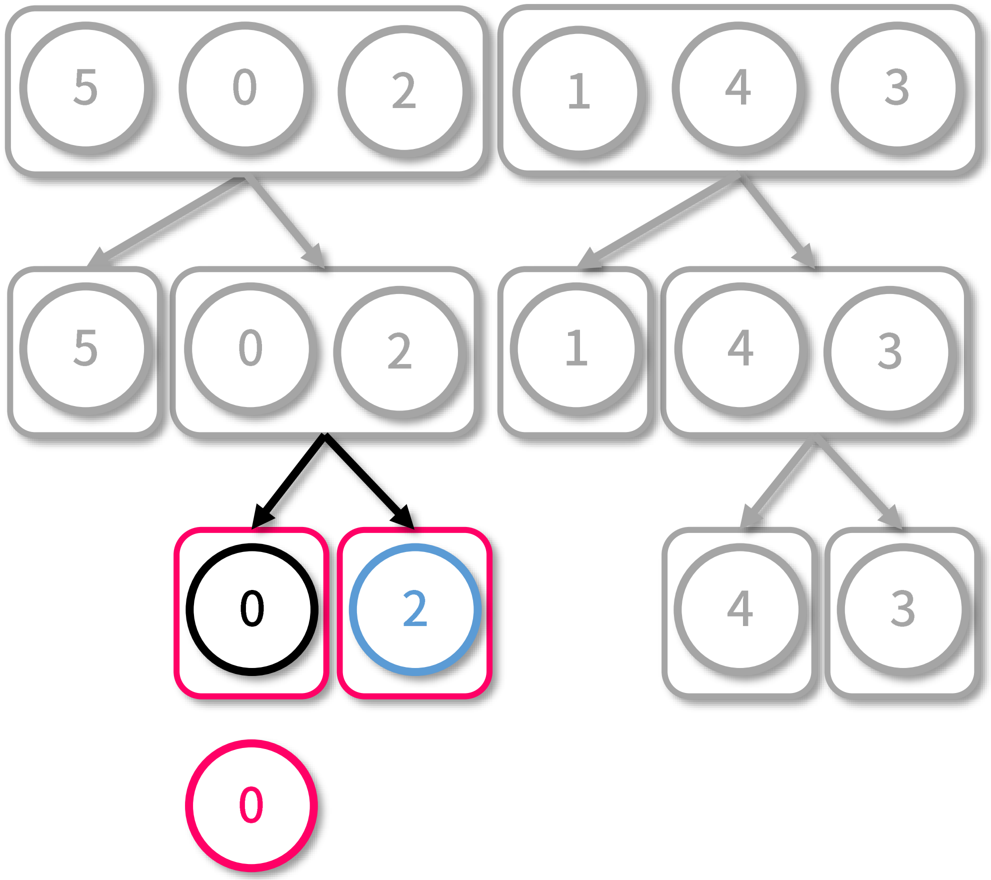
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
   1. Pick the smallest element in either left or right series one by one.
   2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

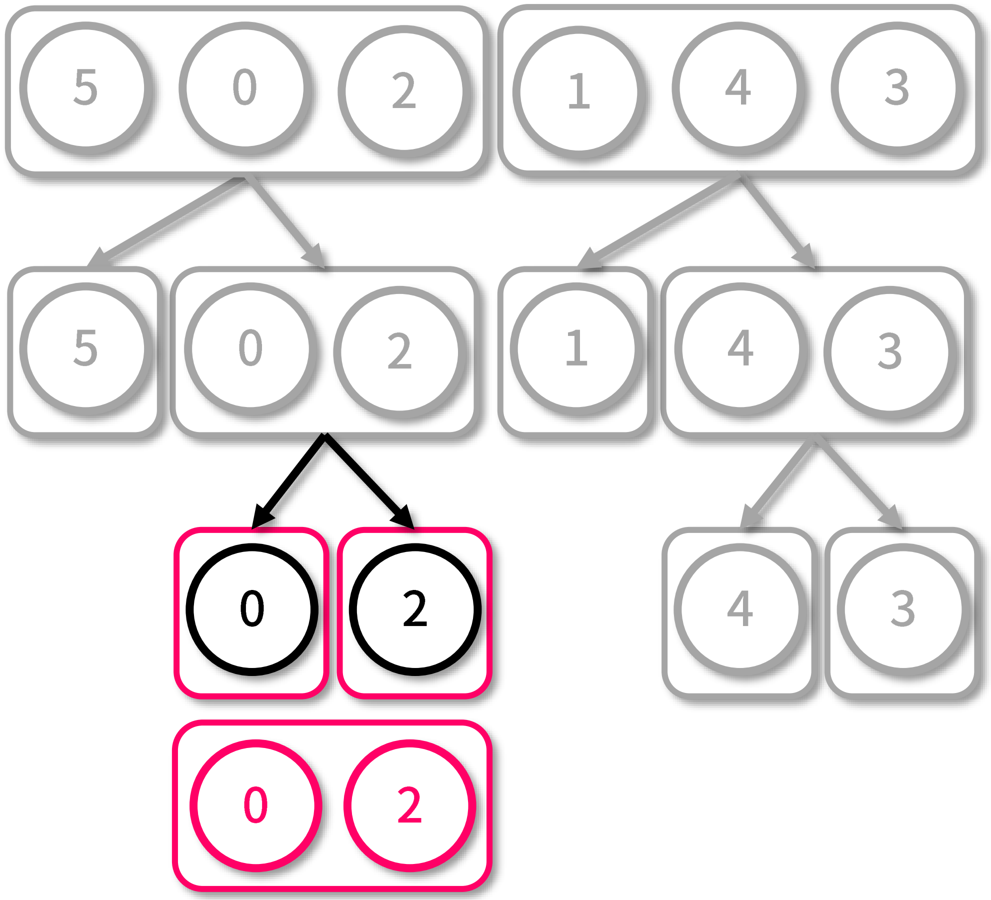
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

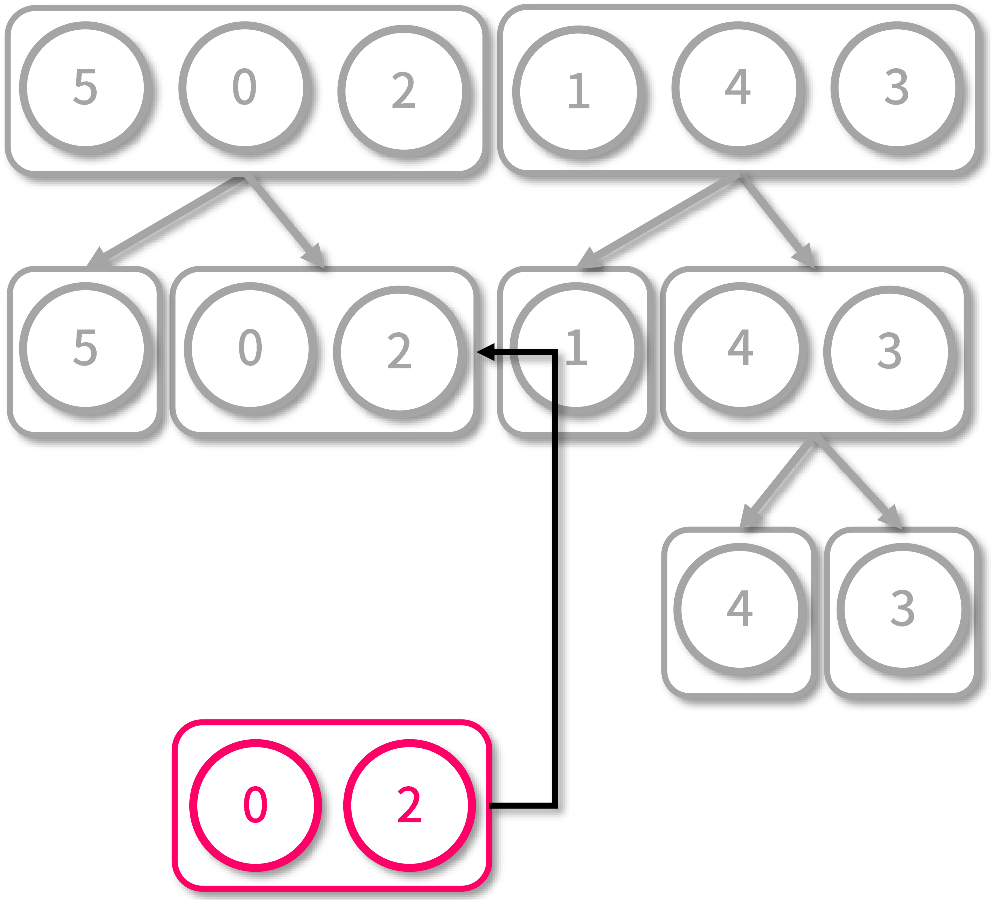
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

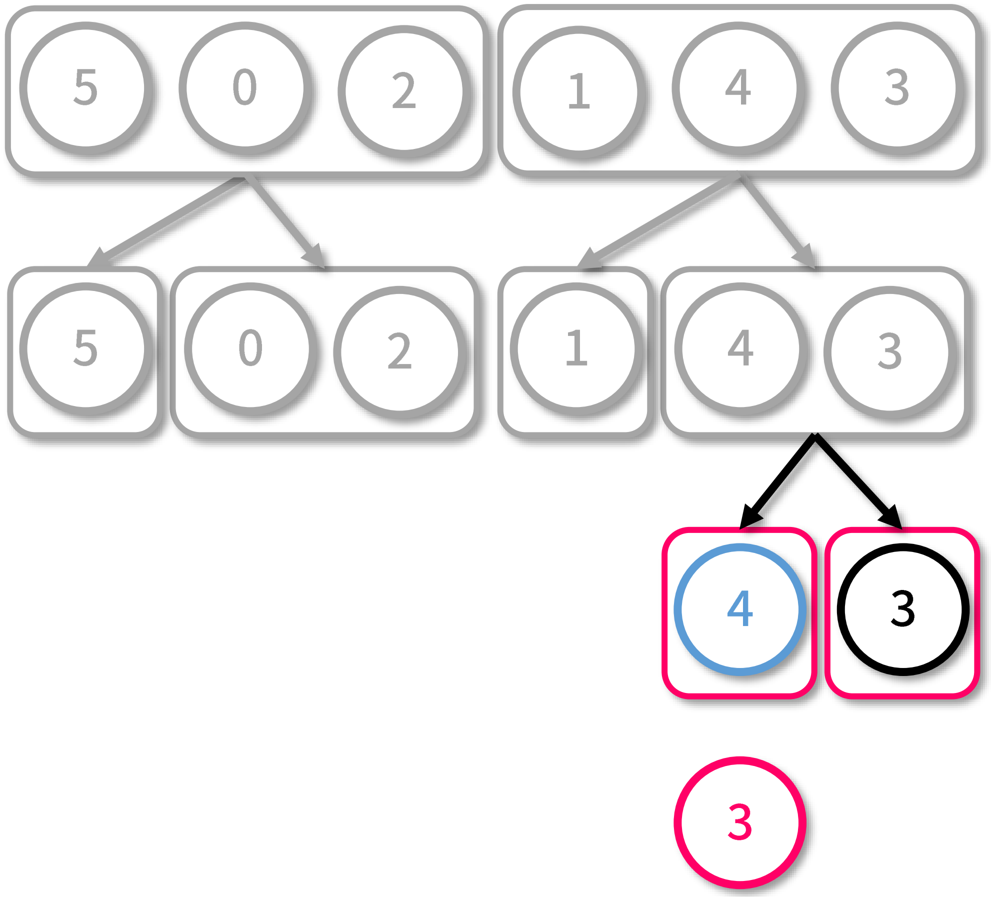
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
   1. Pick the smallest element in either left or right series one by one.
   2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

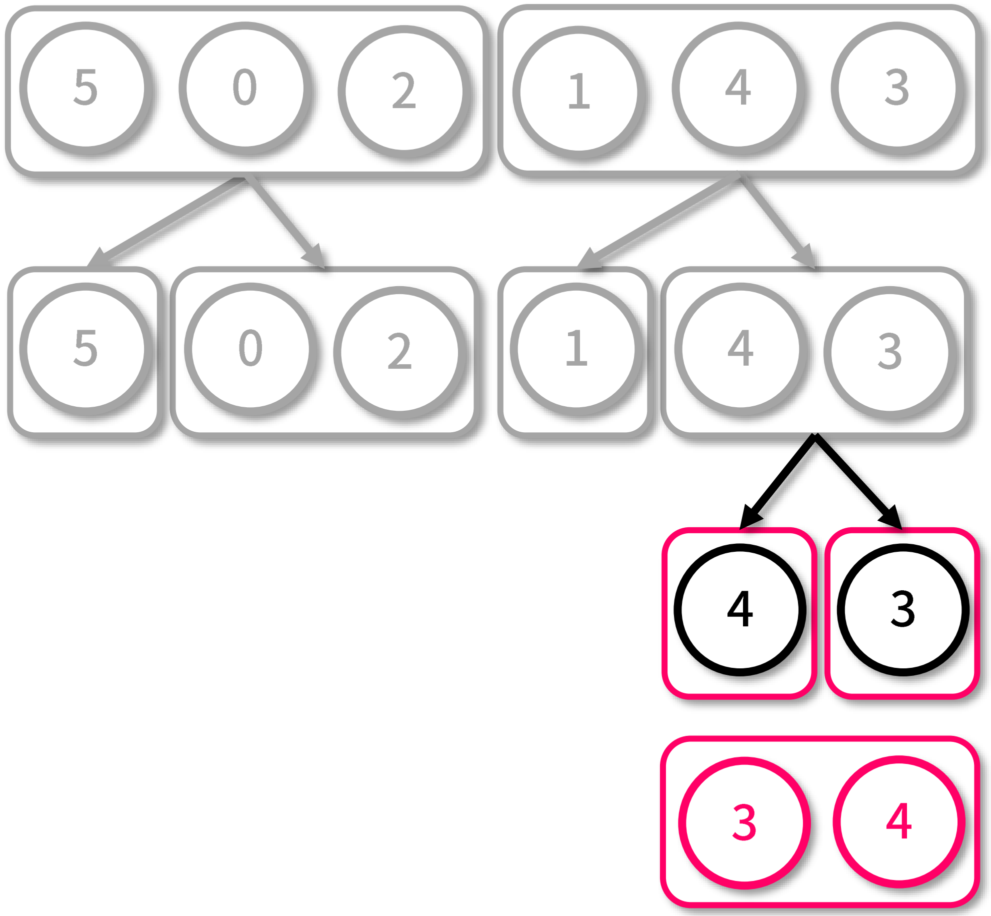
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

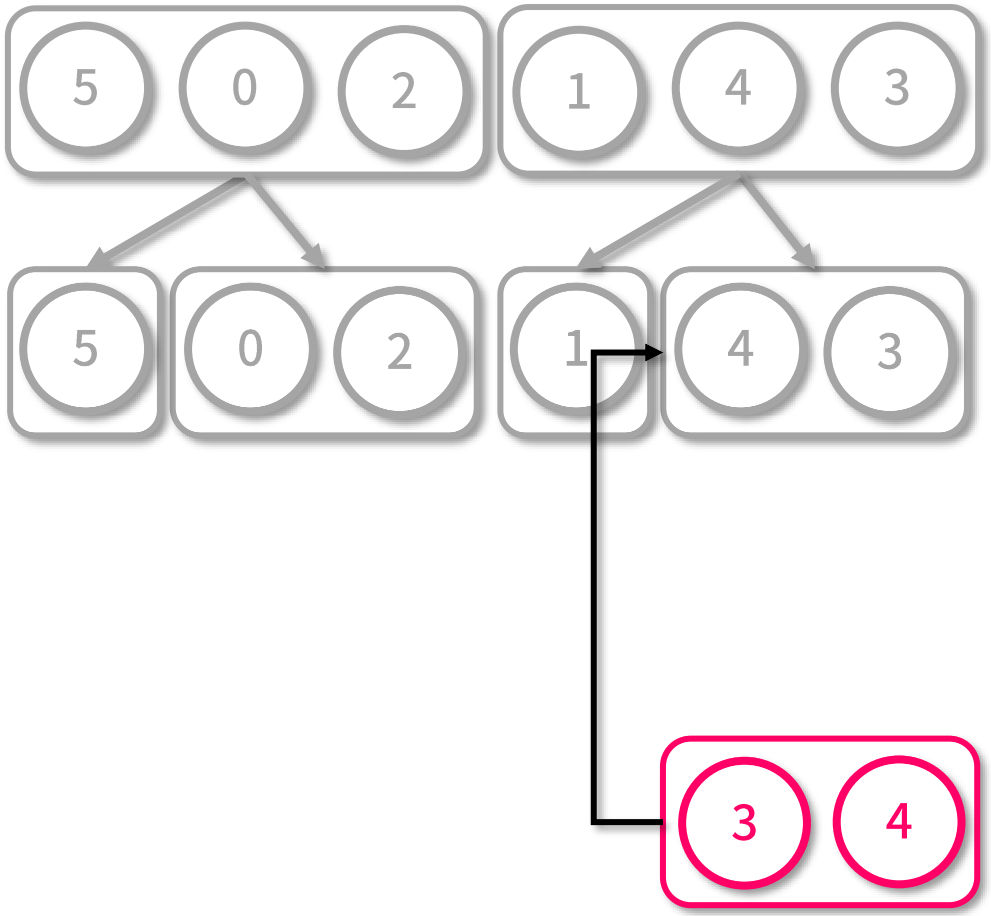
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

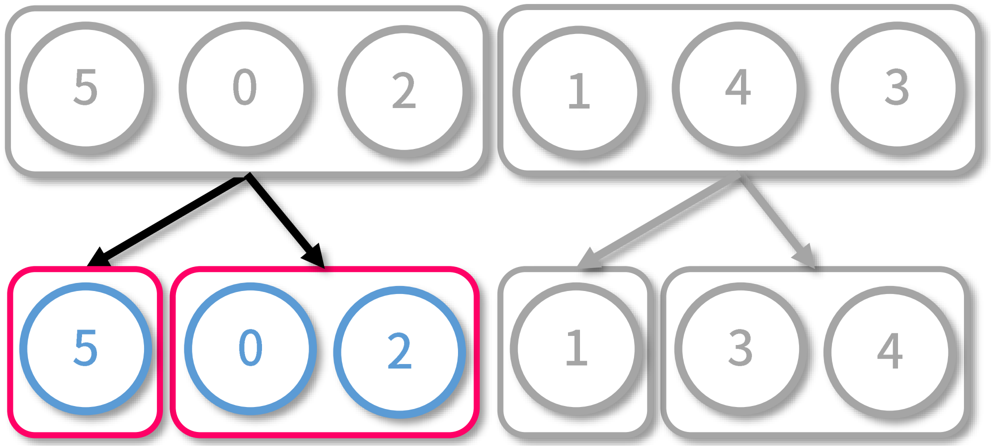
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
   1. Pick the smallest element in either left or right series one by one.
   2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

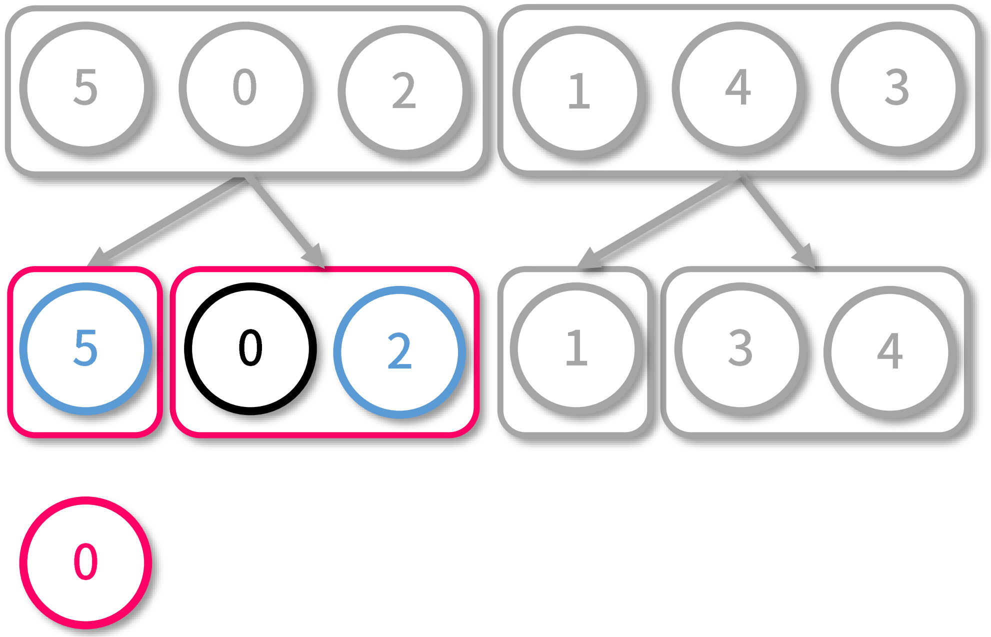
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

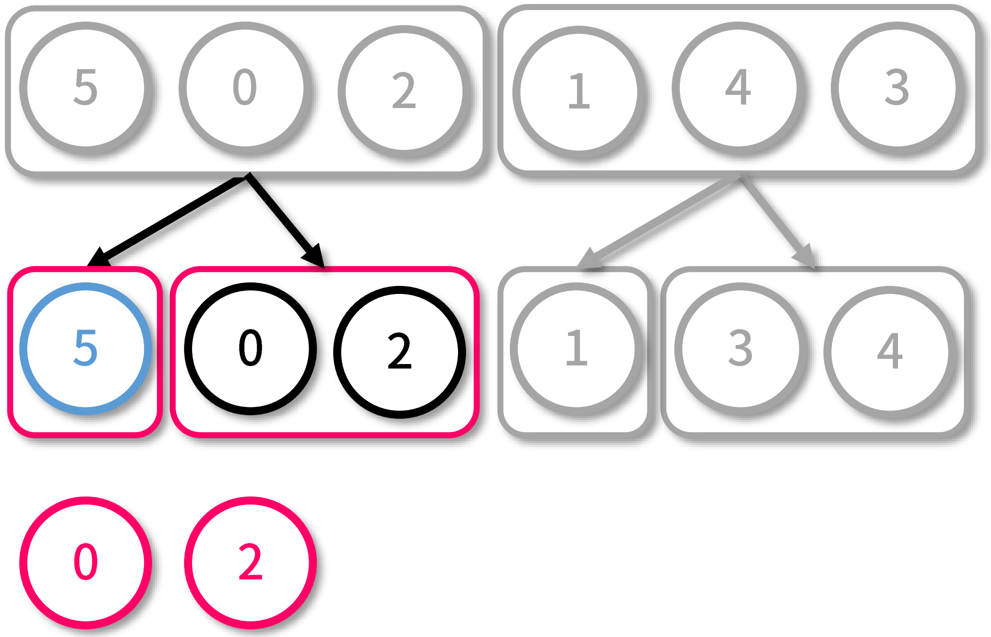
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

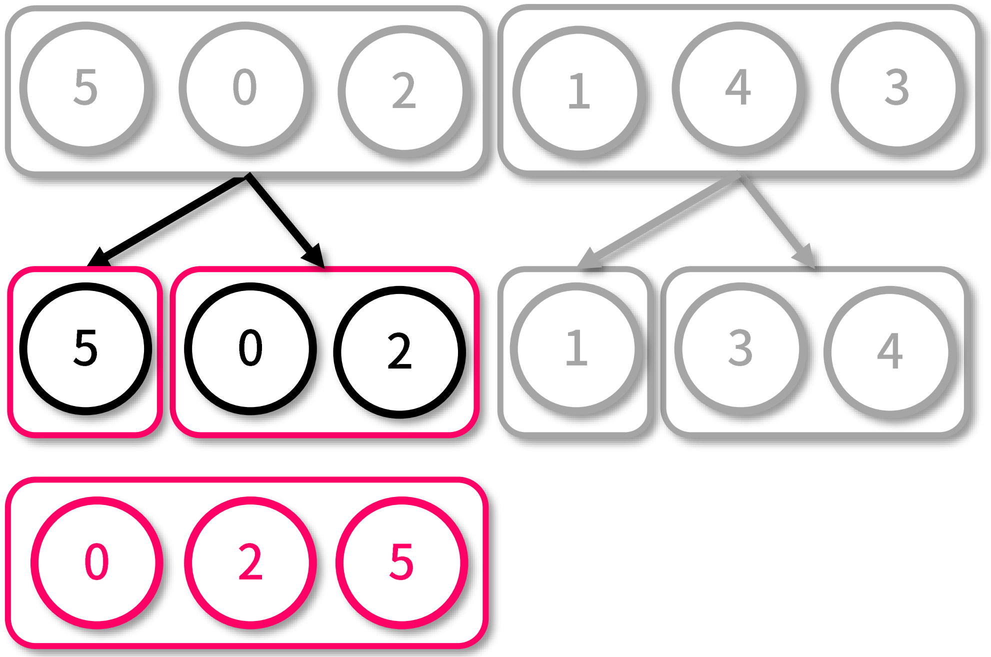
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

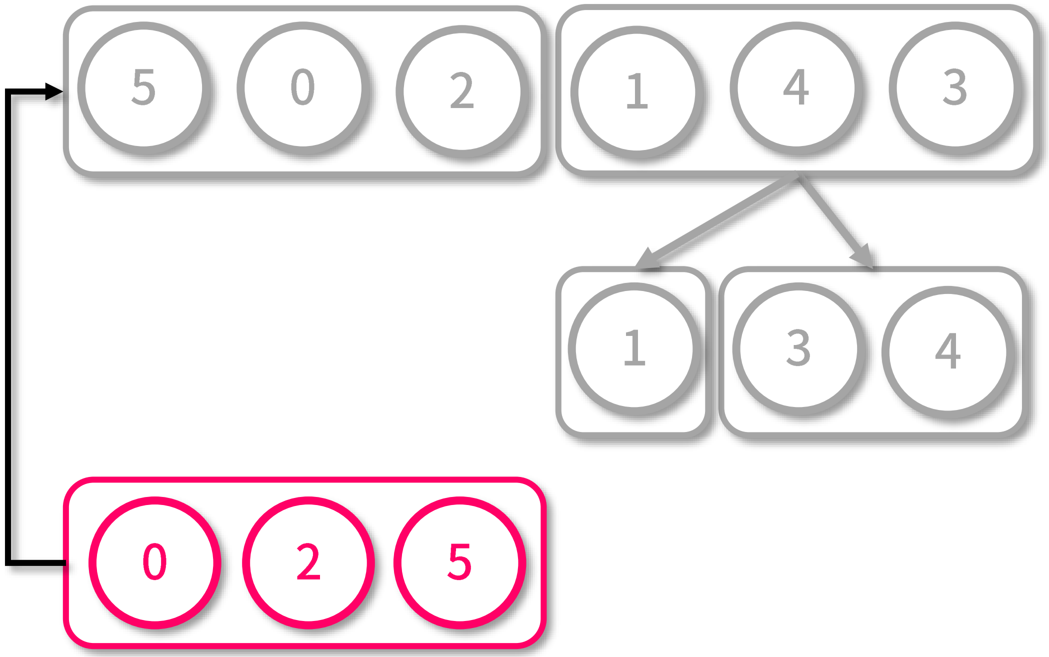
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

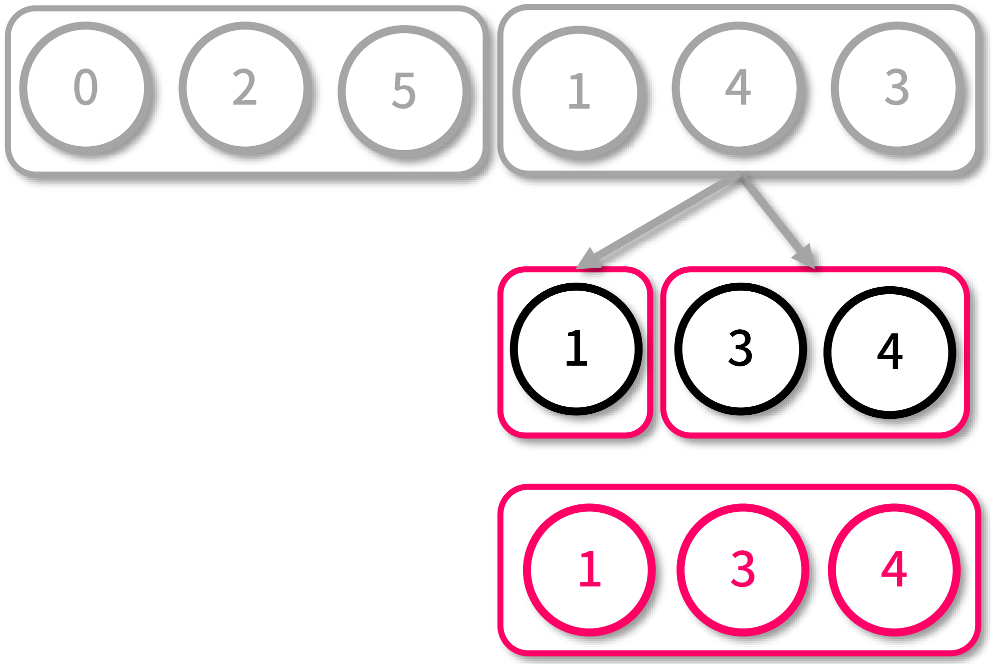
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
   1. Pick the smallest element in either left or right series one by one.
   2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

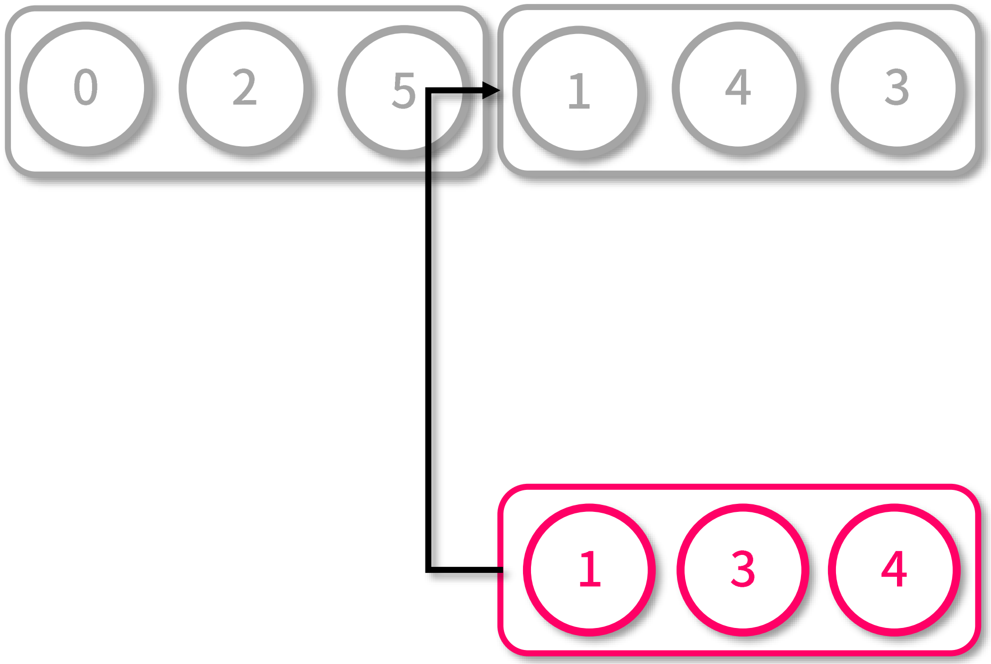
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
   1. Pick the smallest element in either left or right series one by one.
   2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

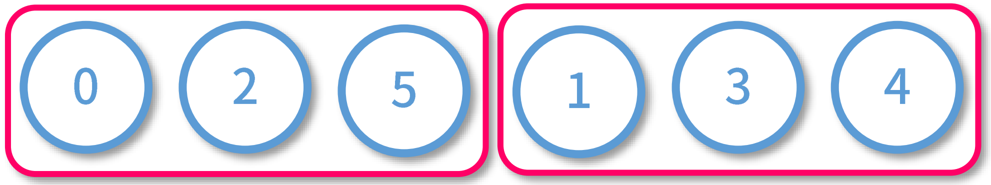
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

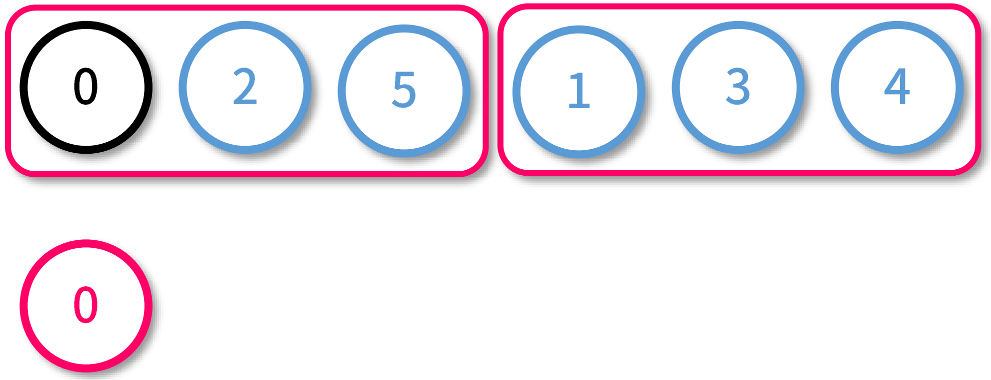
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

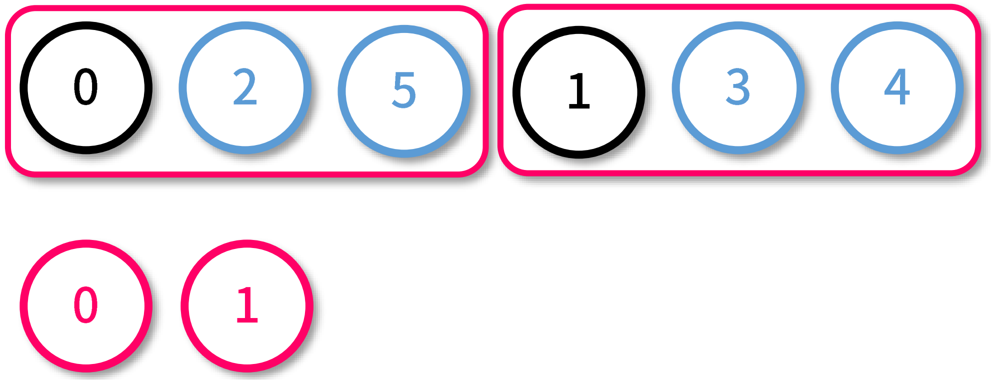
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

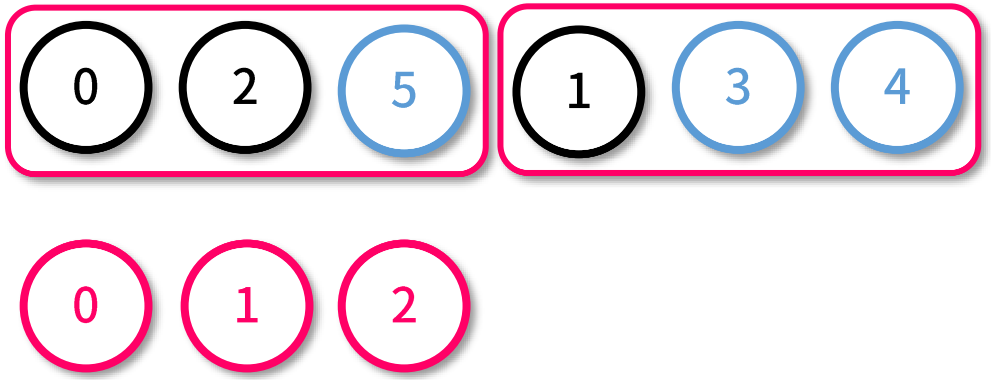
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

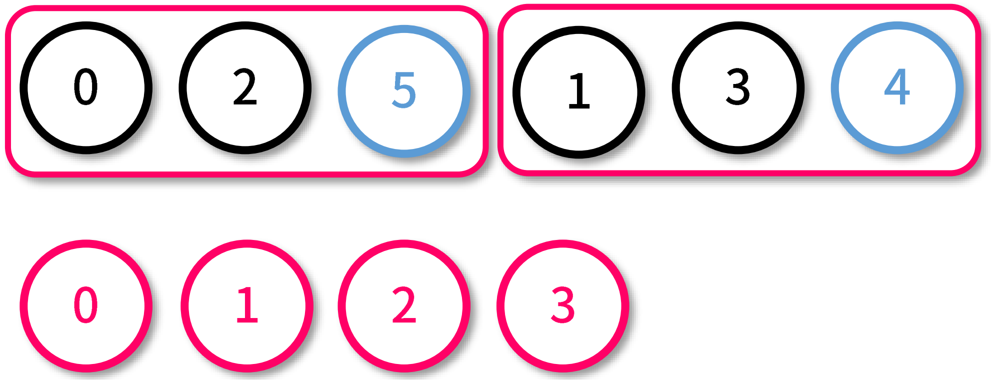
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

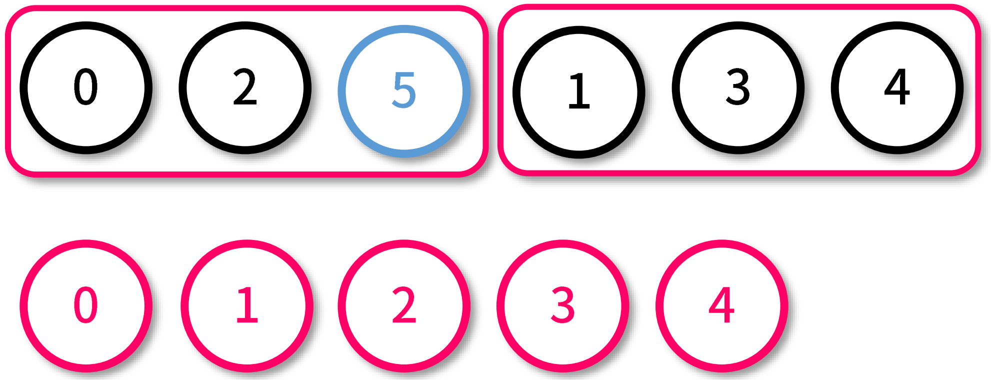
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

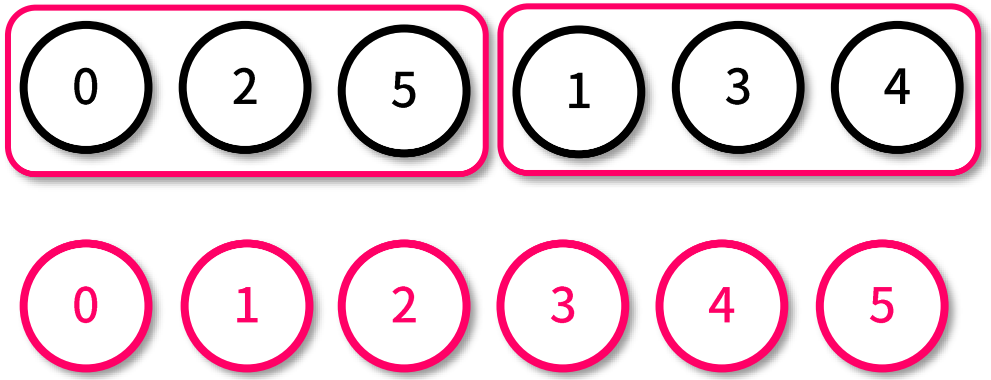
1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.



Merge Sort (1)

Consider a series with n elements.

1. Divide the series into two.
2. Recurrently call itself for left and right series until there is only one element in the inputs.
3. Merge the array.
    1. Pick the smallest element in either left or right series one by one.
    2. Return the merged result.

Invariant
left subseries is sorted, right subseries is sorted

Time complexity
$O(n\log n)$.

Space complexity
$O(n)$.

Stable

Heap Sort (1)

Consider a series with n elements.

1. Make a heap out of the series
2. Pop element n times.

Time complexity
$O(n\log n)$.

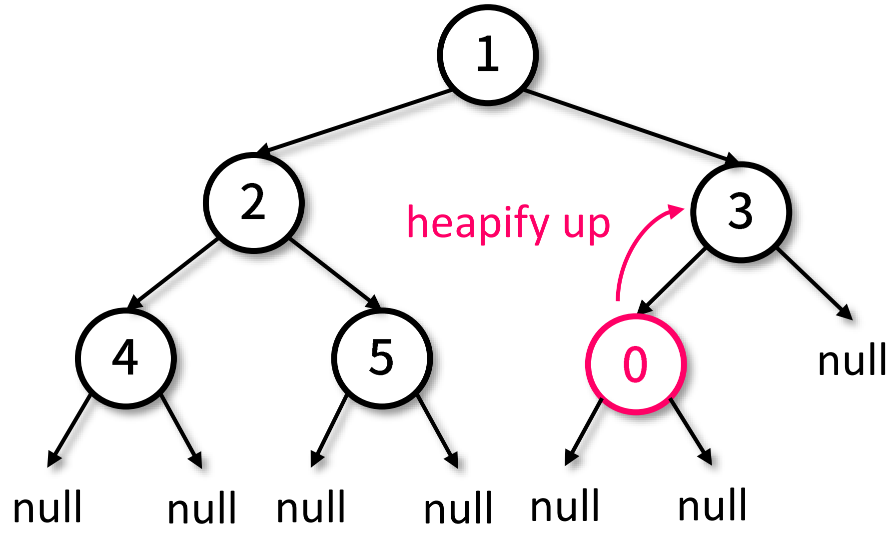
Space complexity
$O(1)$.

Unstable

Minimum Heap
Insert (1)

1. Insert to the next empty leaf node (e.g. insert node with key 0).

2. Swap the inserted node up to the tree until smaller item is encountered (heapify up).

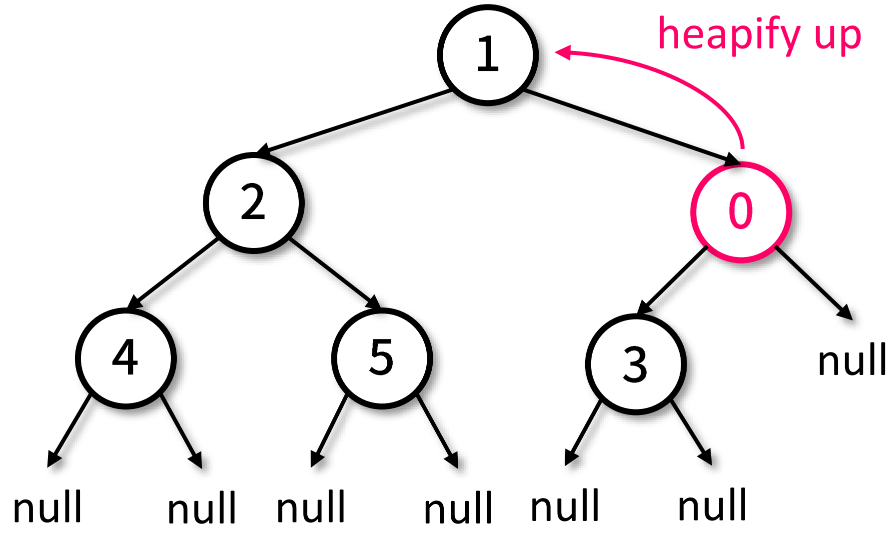


Minimum Heap

Insert (2)

1. Insert to the next empty leaf node (e.g. insert node with key 0).

2. Swap the inserted node up to the tree until smaller item is encountered (heapify up).

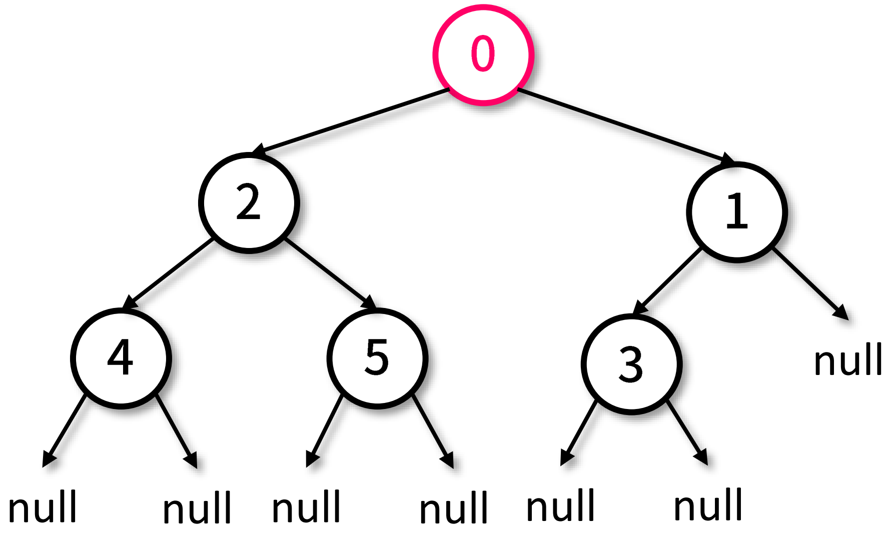


Minimum Heap

Insert (3)

1. Insert to the next empty leaf node (e.g. insert node with key 0).

2. Swap the inserted node up to the tree until smaller item is encountered (heapify up).



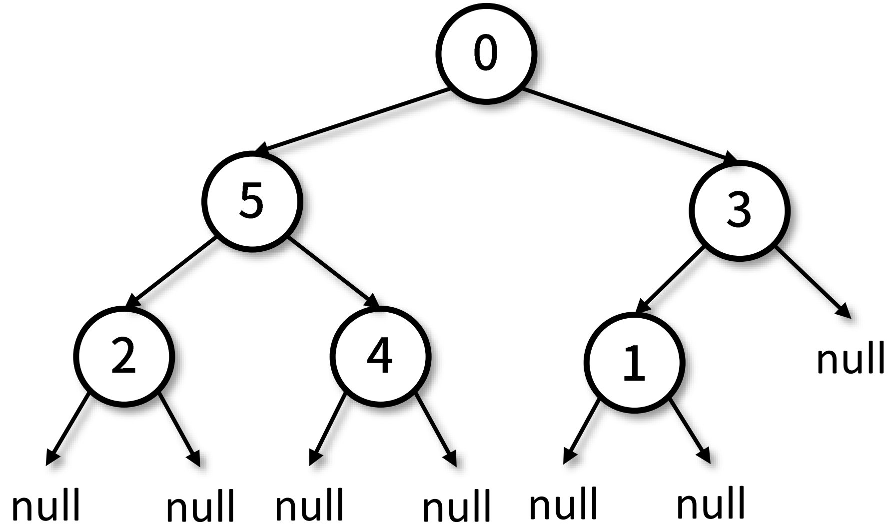
Minimum Heap

BuildHeap (1)

- Given a series of node. Build a minimum heap out of it.

1. Start from the input series in the form of heap.

2. Start from the last non-leaf node. perform heapify down if needed.



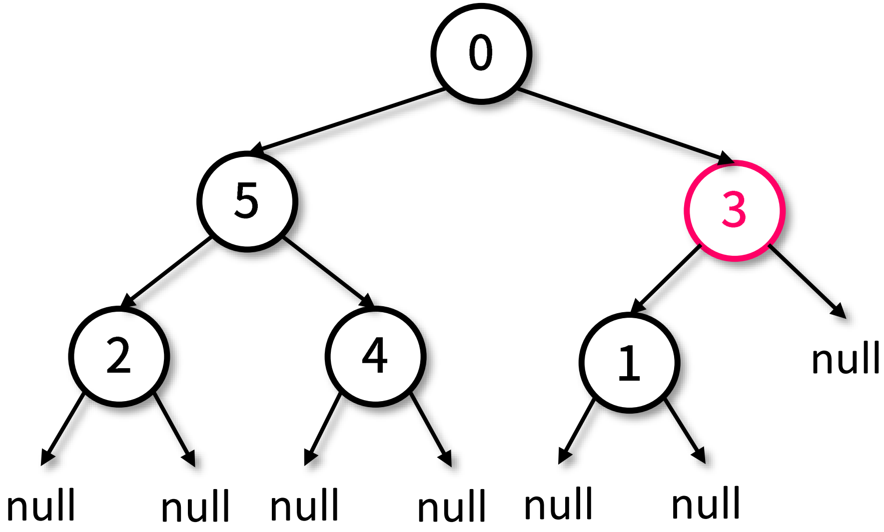
Inputs: 0, 5, 3, 2, 4, 1

Minimum Heap
BuildHeap (2)

- Given a series of node. Build a minimum heap out of it.

1. Start from the input series in the form of heap.

2. Start from the last non-leaf node. perform heapify down if needed.



Inputs: 0, 5, 3, 2, 4, 1

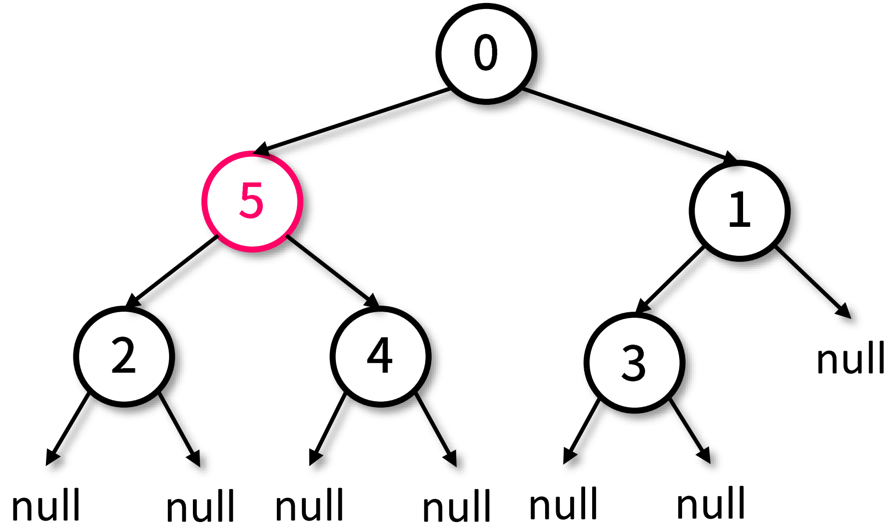
Minimum Heap

BuildHeap (3)

- Given a series of node. Build a minimum heap out of it.

1. Start from the input series in the form of heap.

2. Start from the last non-leaf node. perform heapify down if needed.



Inputs: 0, 5, 3, 2, 4, 1

Minimum Heap

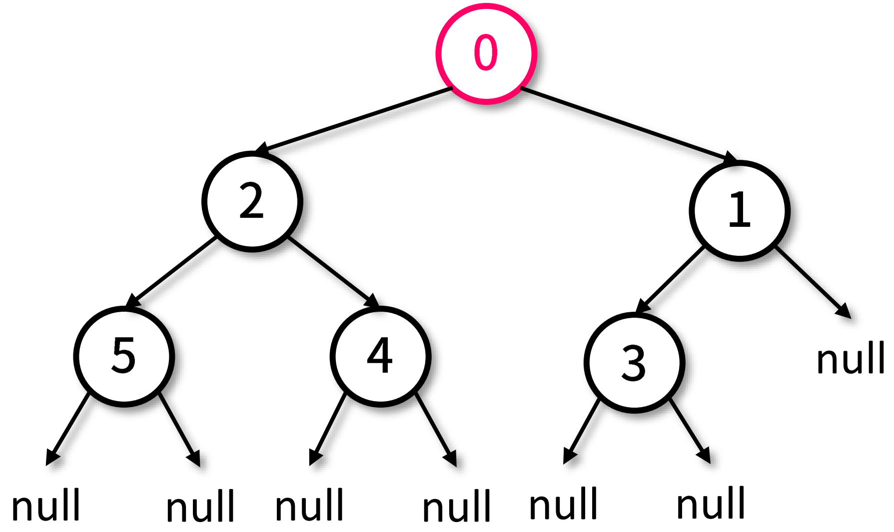
BuildHeap (4)

- Given a series of node. Build a minimum heap out of it.

1. Start from the input series in the form of heap.

2. Start from the last non-leaf node. perform heapify down if needed.

Time complexity
$O(n)$.



Inputs: 0, 5, 3, 2, 4, 1

Exercises

- Implementation of bubble sort, selection sort.
- Implementation of quick sort.
- Implementation of merge sort.

Dynamic Programming

Dynamic Programming

- Optimal substructure
  - An optimal solution can be constructed from optimal solutions of its subproblems.
  - Examples
    - Shortest path of unweighted graph.
    - Quick sort, merge sort.
- Overlapping subproblems.
  - When deriving the optimal solution, subproblems needs to be solved multiple times.
  - Examples
    - Derive Fibonacci numbers.
    - Gapped sequence alignment.

Dynamic Programming

- Solving Steps
  1. Make sure the problem can be decomposed into (overlapping) optimal substructures.
  2. Make a small example.
  3. Write down all the possibilities.
  4. Use induction to write down the relationships between the possibilities (usually this is obvious).
  5. **Define a transition function between our goal and the structures we obtained from step (3). (usually, this is the difficult part)**
  6. Store the result during recursion (**memoization**).
  7. Optimize with **tabulation** (change recursion into bottom-up solving to avoid function call stack overflow)
- Usually, dynamic programming is not the most space-efficient algorithm. But it provides a systematic way to solve complex problems with relatively good time complexity.

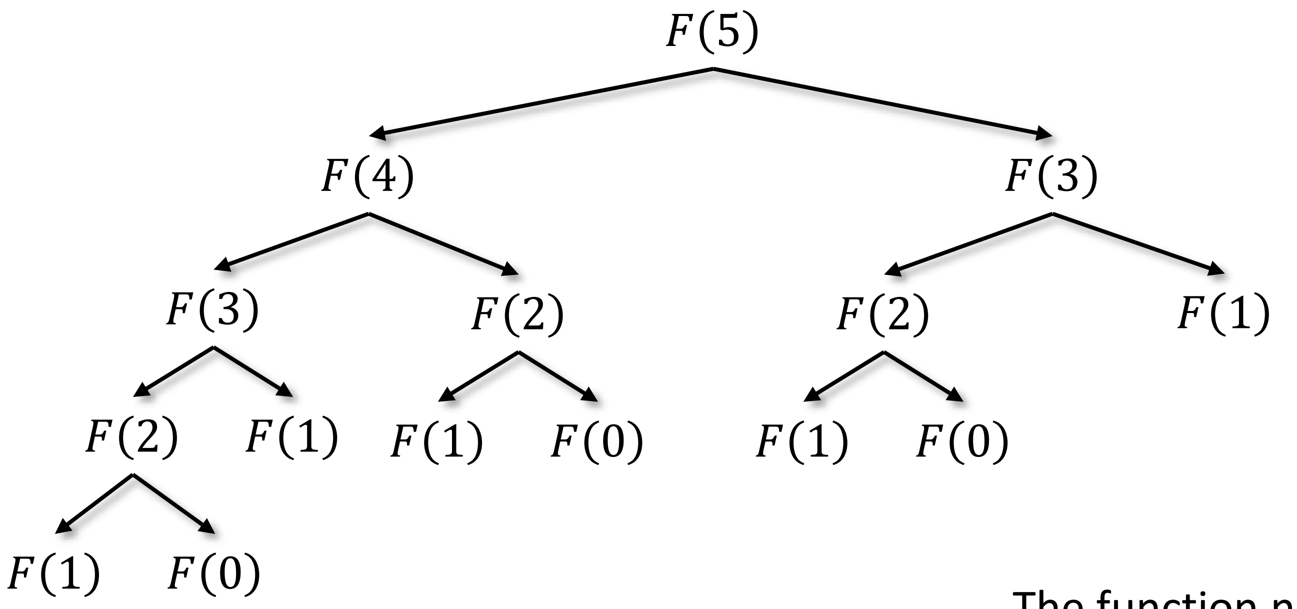
Fibonacci Numbers (1)

- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$

- Naïve approach to derive $F(5)$:
  - Make a function of Finbonacci number according to the definition.
  - Is this the optimal way to construct the function?
  - How many time does the function needs to be called?

Fibonacci Numbers (2)

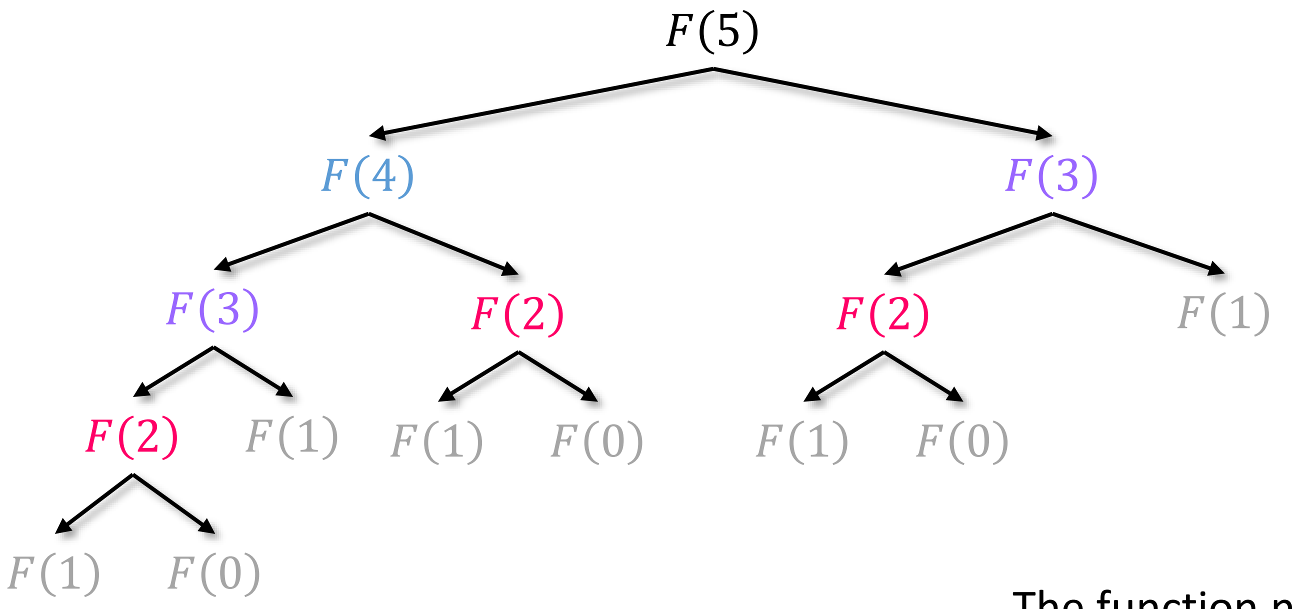
- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$



The function needs to be called 15 times.

Fibonacci Numbers (3)

- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$



The function needs to be called 15 times.

Fibonacci Numbers (4)

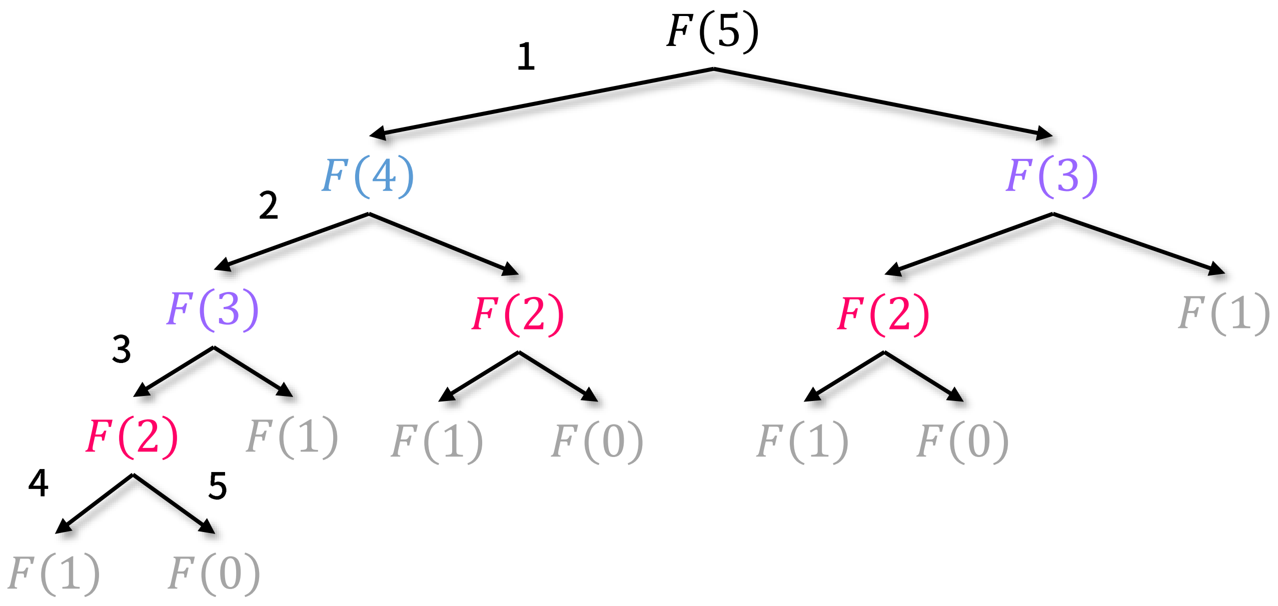
- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$

- Derive $F(5)$ algebraically:
  - $F(5) = F(4) + F(3)$
  - $F(4) = F(3) + F(2)$
  - $F(3) = F(2) + F(1)$
  - $F(2) = F(1) + F(0)$

- What if we save the result after the computation?

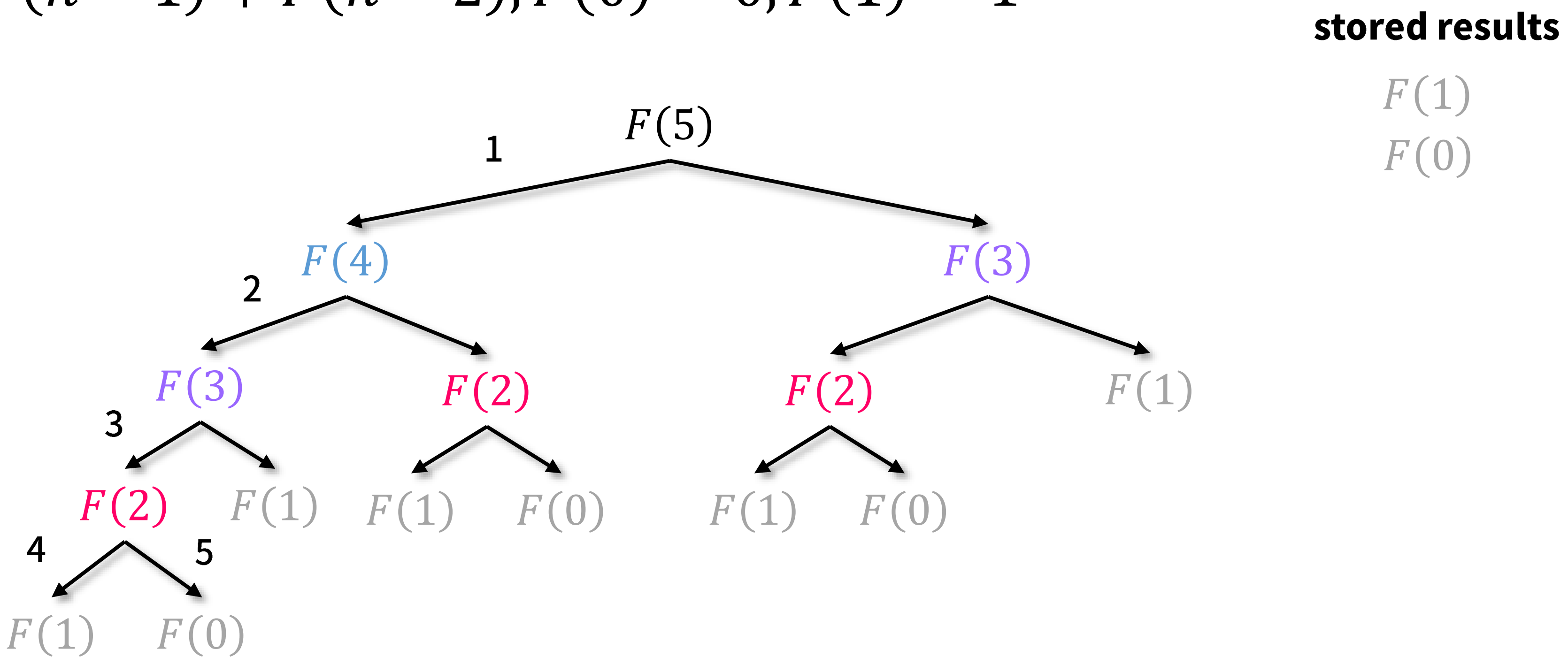
Fibonacci Numbers (5)

- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$



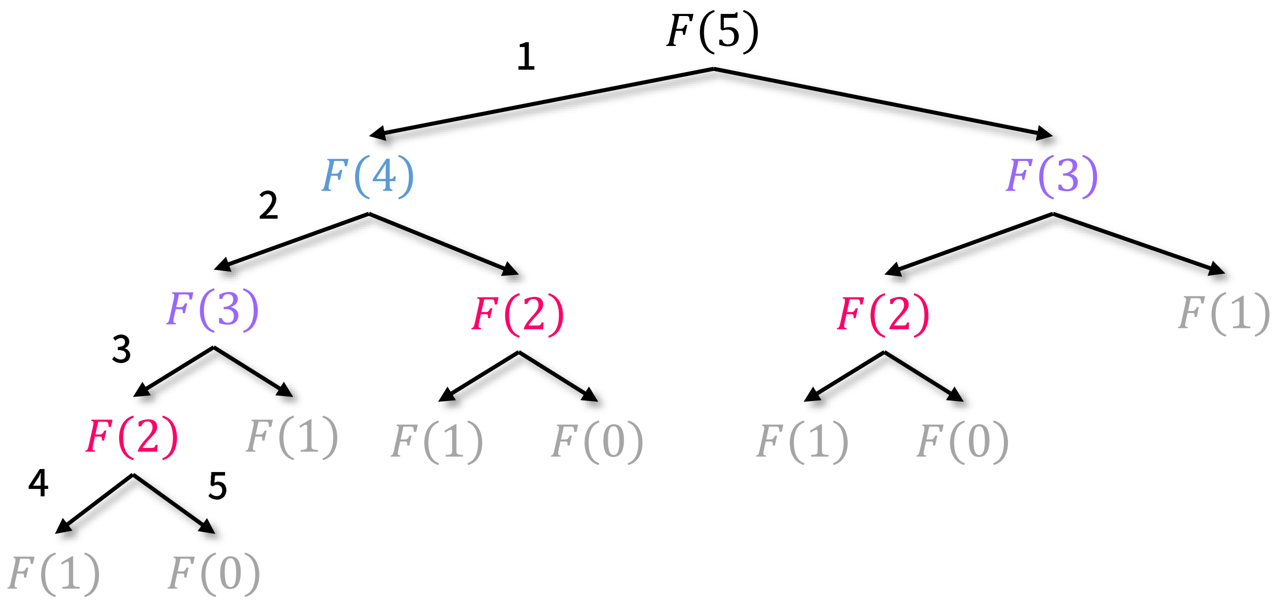
Fibonacci Numbers (6)

- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$



Fibonacci Numbers (7)

- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$



stored results

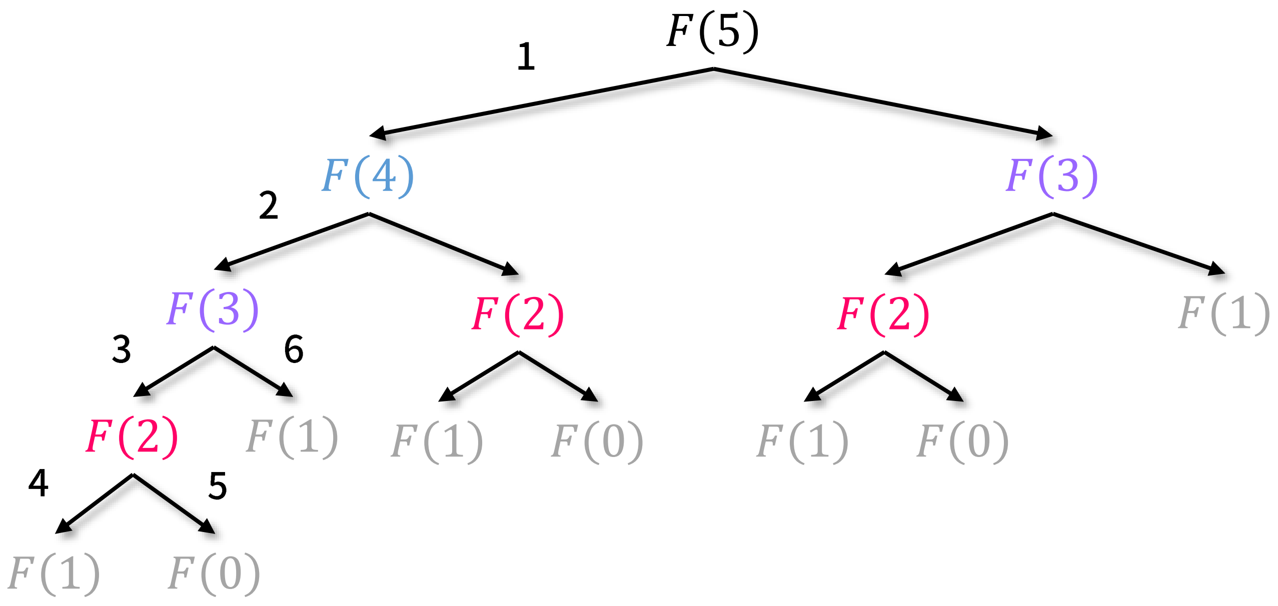
$F(1)$

$F(0)$

$F(2)$

Fibonacci Numbers (8)

- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$



stored results

$F(1)$

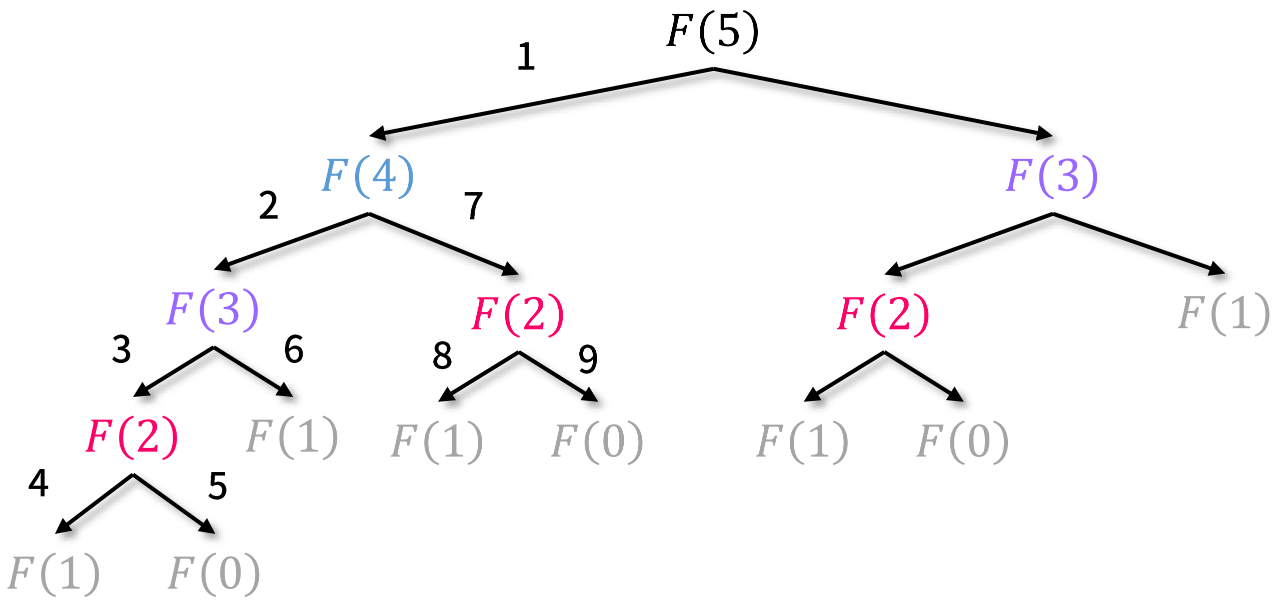
$F(0)$

$F(2)$

$F(3)$

Fibonacci Numbers (9)

- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$

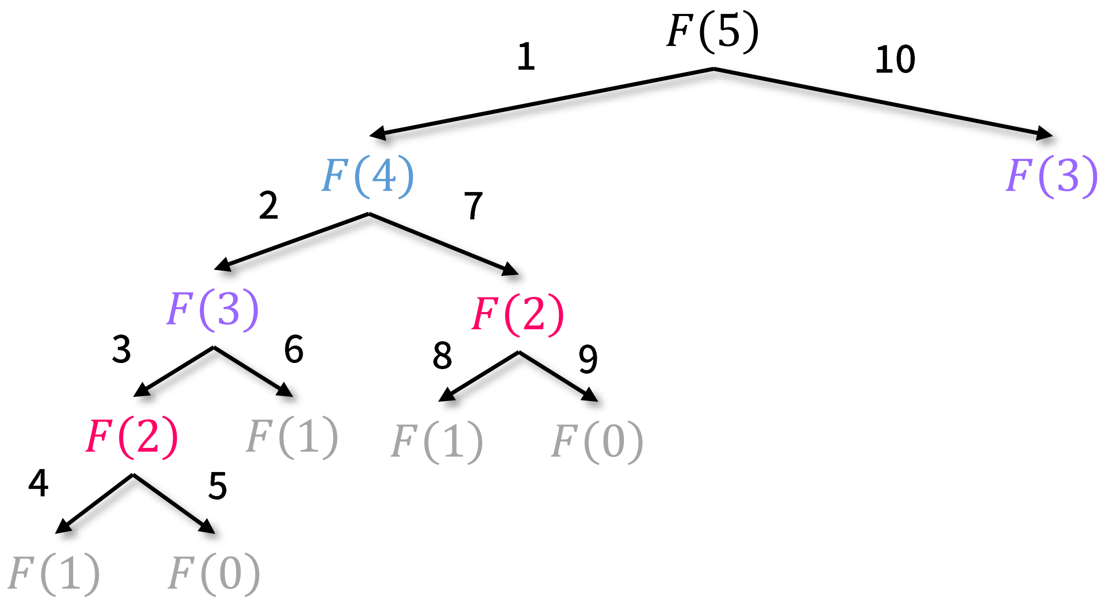


stored results

$F(1)$
$F(0)$
$F(2)$
$F(3)$
$F(4)$

Fibonacci Numbers (10)

- Definition:
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$

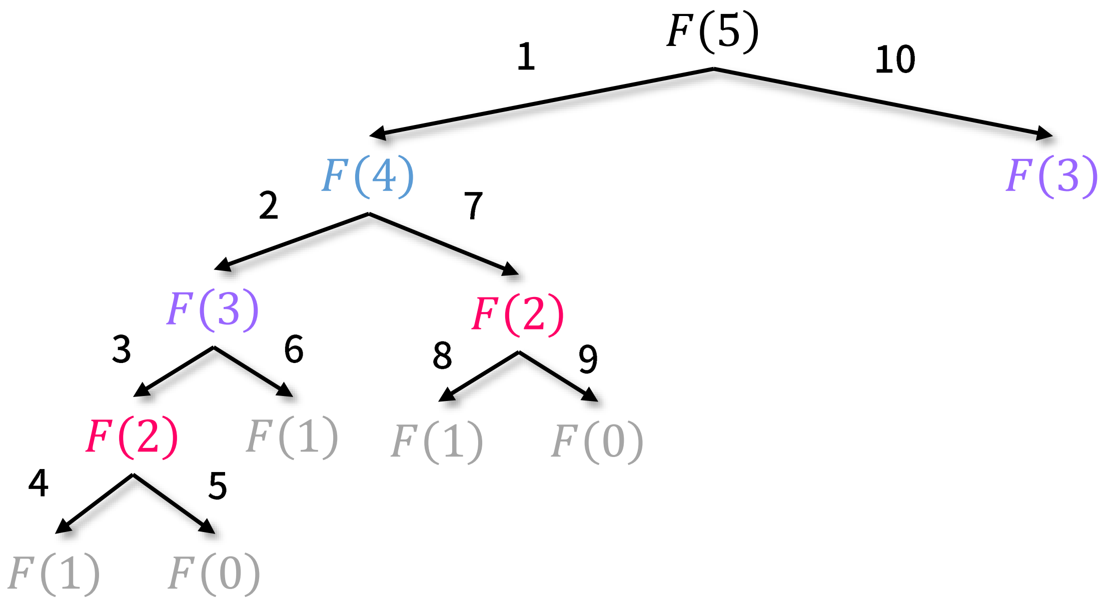


stored results

$F(1)$
$F(0)$
$F(2)$
$F(3)$
$F(4)$

Fibonacci Numbers (11)

- **Definition:**
  - $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$



stored results

$F(1)$
$F(0)$
$F(2)$
$F(3)$
$F(4)$
$F(5)$

The function is be called 10 times.

Longest Common Subsequence (1)

- Given two sequences seq_1 and seq_2.
- Find the length of longest common subsequence between seq_1 and seq_2.
- Common subsequence:
  - The characters of the subsequence have the same order in seq_1 and seq_2.
- Examples:
  - LCS(abc, ac) = ac
  - LCS(acebebac, abc) = abc
  - LCS(ac, bd) = Null

Longest Common Subsequence (2)

all possibilities of LCS

LCS(acbbac, abc)

a + LCS(cbbac, bc)

LCS(bbac, bc)

LCS(cbbac, c)

b + LCS(bac, c)

c + LCS(bbac, Null)

LCS(ac, c)

LCS(bac, Null)

LCS(c, c)

LCS(ac, Null)

c + LCS(Null, Null)

Longest Common Subsequence (2)

Transition function

LCS(acbbac, abc) 0 + 3

1 + max(2, 1) a + LCS(cbbac, bc)

(0 + 2) LCS(bbac, bc)

LCS(cbbac, c) (0 + 1)

1 + max(1, 0) b + LCS(bac, c)

c + LCS(bbac, Null) (1 + 0)

max(1, 0) LCS(ac, c)

LCS(bac, Null) (0)

(0 + 1) LCS(c, c)

LCS(ac, Null) (0)

(1 + 0) c + LCS(Null, Null)

Longest Common Subsequence (3)

Transition function

1 + max(2, 1) LCS(acbbac, abc)

1 + max(1, 0) LCS(bbac, bc) LCS(cbbac, c) 1 + max(0, 0)

max(1, 0) LCS(ac, c) LCS(bac, Null) (0)

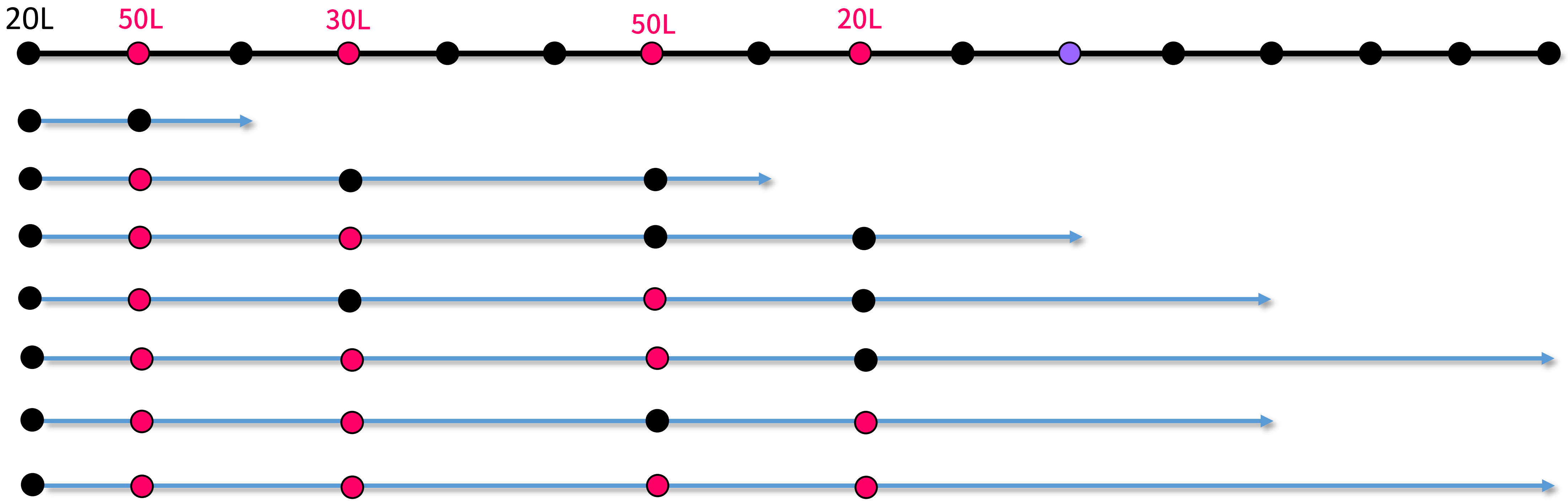
1 + max(0, 0) LCS(c, c) LCS(ac, Null) (0)

Gas Station Problem (1)

- We are driving a car with 1L/1km efficiency.
- The car has 20L of fuel for now.
- The capacity of the fuel is infinite for the car.
- We want to drive to a destination 100km away.
- There are 3 gas station along the way located at [10, 30, 60, 80].
- The 3 gas stations have [50L, 30L, 50L, 20L] fuel left.
- What's the minimum stops to reach the destination?

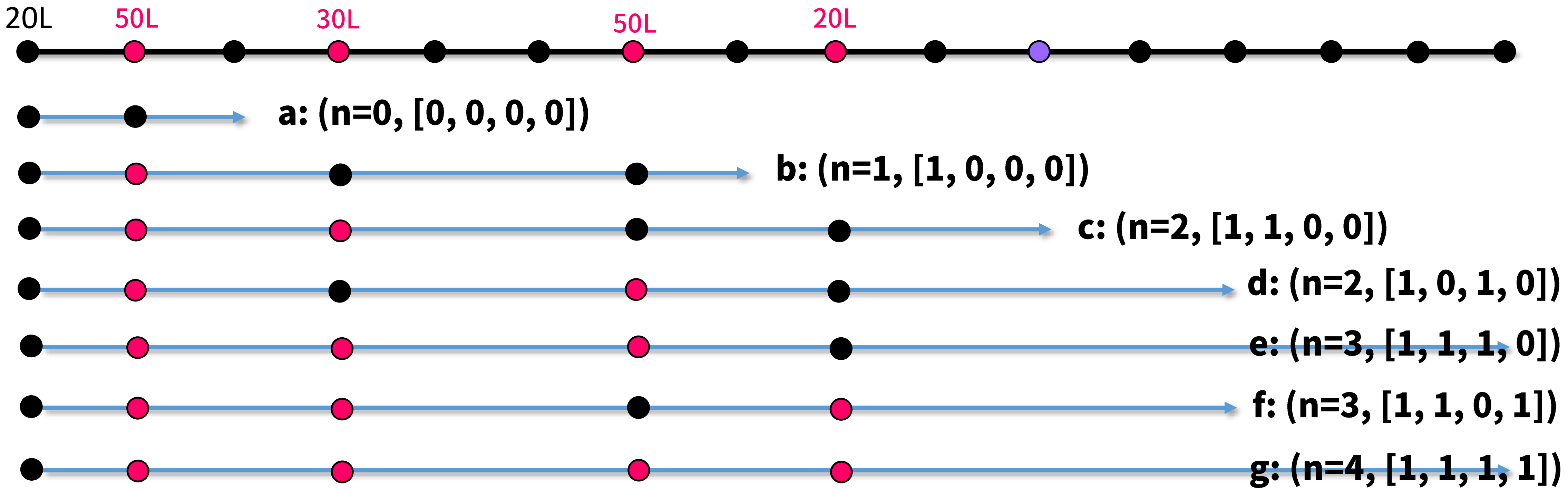
Gas Station Problem (2)

all possibilities of refueling



Gas Station Problem (3)

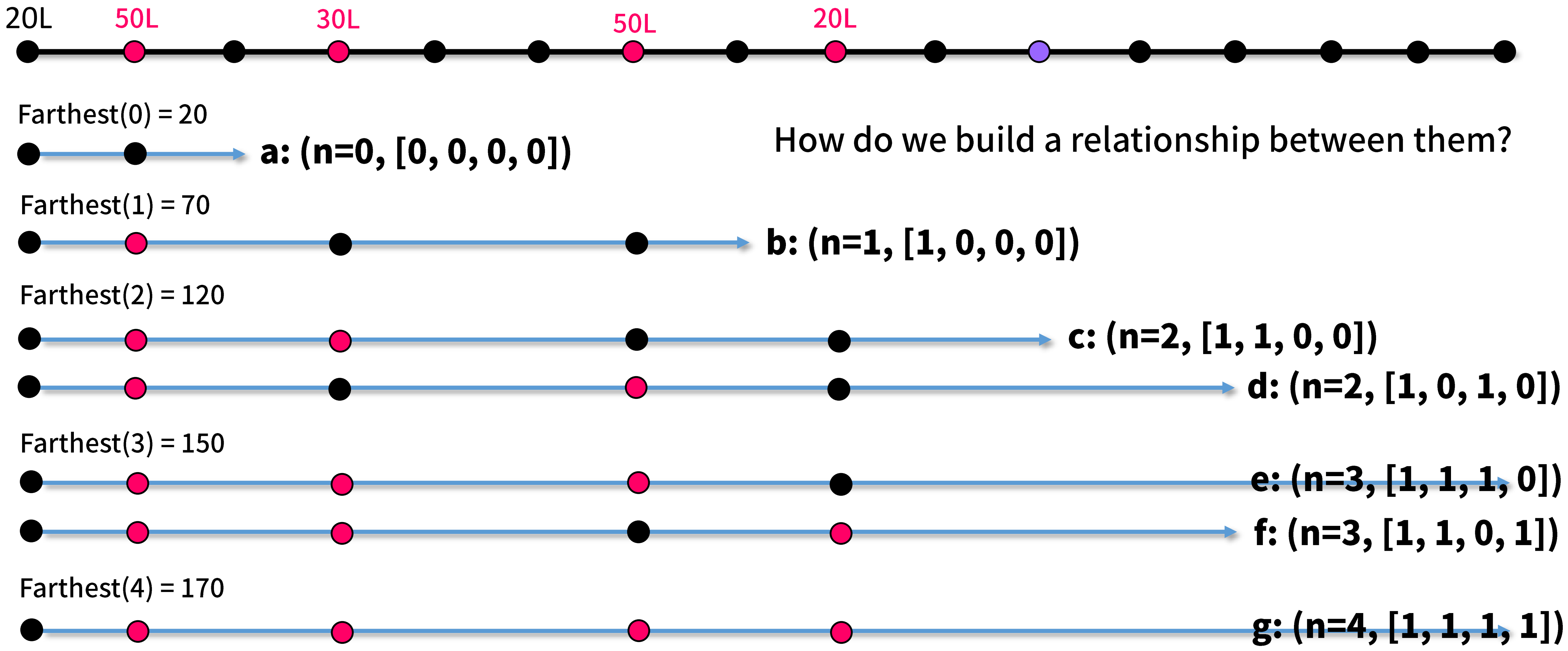
all possibilities of refueling



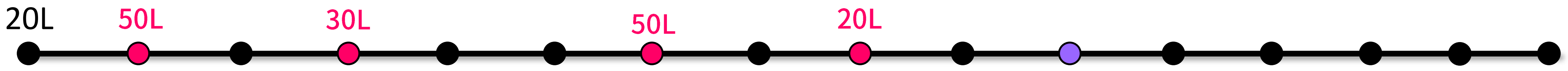
- FarthestP(b) = FarthestP(a) + capacity(s=1)
- FarthestP(c) = FarthestP(b) + capacity(s=2)
- FarthestP(d) = FarthestP(b) + capacity(s=3)
- FarthestP(e) = FarthestP(c) + capacity(s=3)
- FarthestP(f) = FarthestP(c) + capacity(s=4)
- FarthestP(g) = FarthestP(f) + capacity(s=3)

Gas Station Problem (4)

How do we build a relationship between them?



Gas Station Problem (4)



Farthest(0) = 20 = FarthestP(a)

a: (n=0, [0, 0, 0, 0])

How do we build a relationship between them?

Farthest(1) = 70 = FarthestP(b)

b: (n=1, [1, 0, 0, 0])

Farthest(2) = 120 = max(FarthestP(c), FarthestP(d))

c: (n=2, [1, 1, 0, 0])

d: (n=2, [1, 0, 1, 0])

Farthest(3) = 150 = max(FarthestP(e), FarthestP(f))

e: (n=3, [1, 1, 1, 0])

f: (n=3, [1, 1, 0, 1])

Farthest(4) = 170 =FarthestP(g)

g: (n=4, [1, 1, 1, 1])

Gas Station Problem (5)

All Possibilities

- a: (n=0, [0, 0, 0, 0])
- b: (n=1, [1, 0, 0, 0])
- c: (n=2, [1, 1, 0, 0])
- d: (n=2, [1, 0, 1, 0])
- e: (n=3, [1, 1, 1, 0])
- f: (n=3, [1, 1, 0, 1])
- g: (n=4, [1, 1, 1, 1])

- FarthestP(a) = 20
- FarthestP(b) = FarthestP(a) + capacity(s=1)
- FarthestP(c) = FarthestP(b) + capacity(s=2)
- FarthestP(d) = FarthestP(b) + capacity(s=3)
- FarthestP(e) = FarthestP(c) + capacity(s=3)
- FarthestP(f) = FarthestP(c) + capacity(s=4)
- FarthestP(g) = FarthestP(f) + capacity(s=3)

Goals

- Farthest(0) = 20 = FarthestP(a)
- Farthest(1) = 70 = FarthestP(b)
- Farthest(2) = 120 = max(FarthestP(c), FarthestP(d))
- Farthest(3) = 150 = max(FarthestP(e), FarthestP(f))
- Farthest(4) = 170 =FarthestP(g)

Gas Station Problem (6)

All Possibilities

- a: (n=0, [0, 0, 0, 0])
- b: (n=1, [1, 0, 0, 0])
- c: (n=2, [1, 1, 0, 0])
- d: (n=2, [1, 0, 1, 0])
- e: (n=3, [1, 1, 1, 0])
- f: (n=3, [1, 1, 0, 1])
- g: (n=4, [1, 1, 1, 1])

- FarthestP(a) = 20
- FarthestP(b) = FarthestP(a) + capacity(s=1)
- FarthestP(c) = FarthestP(b) + capacity(s=2)
- FarthestP(d) = FarthestP(b) + capacity(s=3)
- FarthestP(e) = FarthestP(c) + capacity(s=3)
- FarthestP(f) = FarthestP(c) + capacity(s=4)
- FarthestP(g) = FarthestP(f) + capacity(s=3)

Goals: Farthest(n) = max({FarthestP(x) | x.n = n})

- Farthest(0) = 20 = max(FarthestP(a))
- Farthest(1) = 70 = max(FarthestP(b))
- Farthest(2) = 120 = max(FarthestP(c), FarthestP(d))
- Farthest(3) = 150 = max(FarthestP(e), FarthestP(f))
- Farthest(4) = 170 = max(FarthestP(g))

Gas Station Problem (7)

All Possibilities

- a: (n=0, [0, 0, 0, 0])
- b: (n=1, [1, 0, 0, 0])
- c: (n=2, [1, 1, 0, 0])
- d: (n=2, [1, 0, 1, 0])
- e: (n=3, [1, 1, 1, 0])
- f: (n=3, [1, 1, 0, 1])
- g: (n=4, [1, 1, 1, 1])

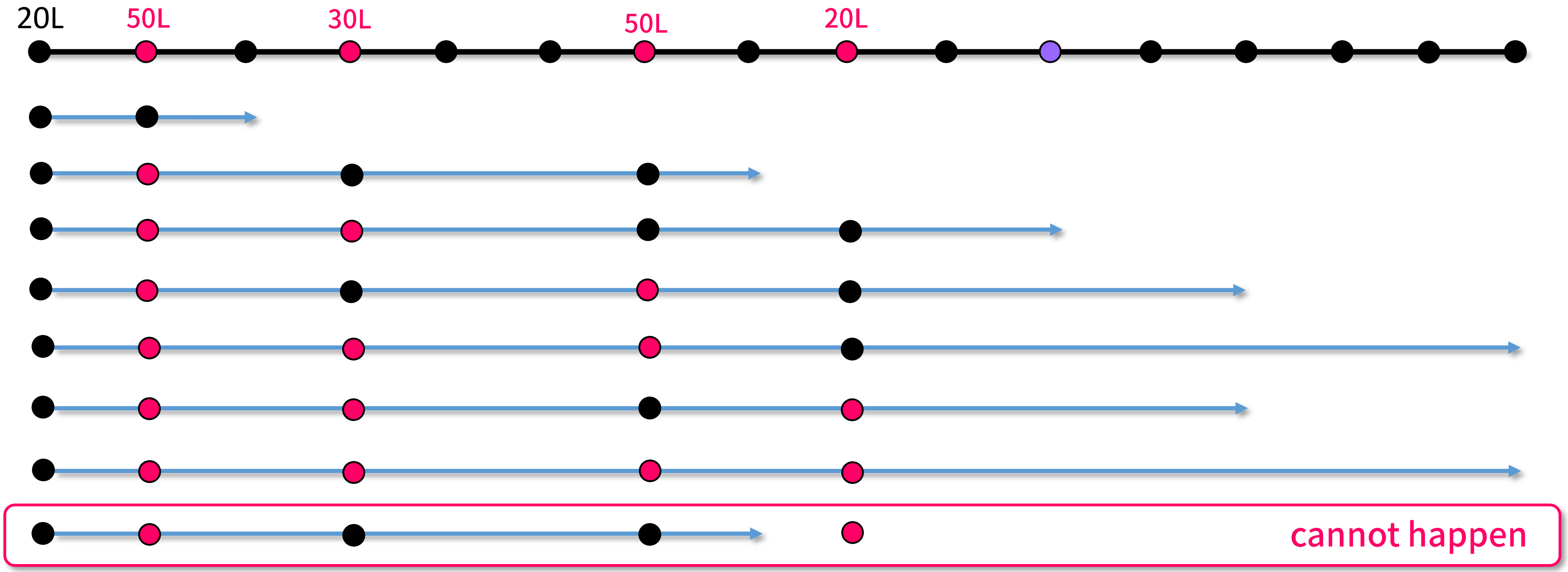
- FarthestP(a) = 20
- FarthestP(b) = FarthestP(a) + capacity(s=1)
- FarthestP(c) = FarthestP(b) + capacity(s=2)
- FarthestP(d) = FarthestP(b) + capacity(s=3)
- FarthestP(e) = FarthestP(c) + capacity(s=3)
- FarthestP(f) = FarthestP(c) + capacity(s=4)
- FarthestP(g) = FarthestP(f) + capacity(s=3)

Goals: max({FarthestP(y) + capacity(s) | x.n = n, y.n=n-1, s = where(x.fuels – y.fuels)})

- Farthest(0) = 20 = max(FarthestP(a))
- Farthest(1) = 70 = max(FarthestP(b))
- Farthest(2) = 120 = max(FarthestP(c), FarthestP(d))
- Farthest(3) = 150 = max(FarthestP(e), FarthestP(f))
- Farthest(4) = 170 = max(FarthestP(g))

Gas Station Problem (8)

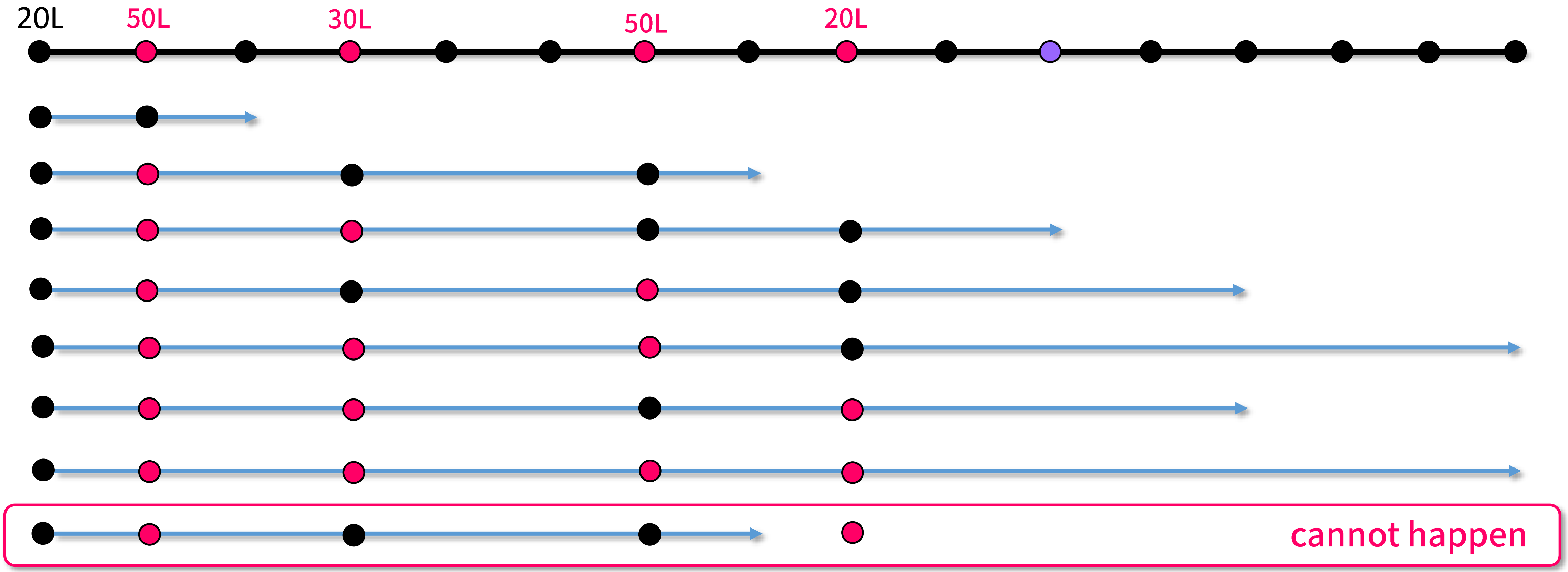
all possibilities of refueling



Goal: max({FarthestP(y) + capacity(s) | x.n = n, y.n=n-1, s = where(x.fuels – y.fuels)})

Gas Station Problem (9)

all possibilities of refueling



Goal: max({FarthestP(y) + capacity(s) | x.n = n, y.n=n-1, s = where(x.fuels – y.fuels), distance(s) < FarthestP(y)})

Exercises

- Minimum Number of Refueling Stops (Leetcode 871)
- Stone Games (Leetcode 877)

Thanks