Data Structures and Algorithms
Big-O Notation, Stack and Queue

2022-07-01

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

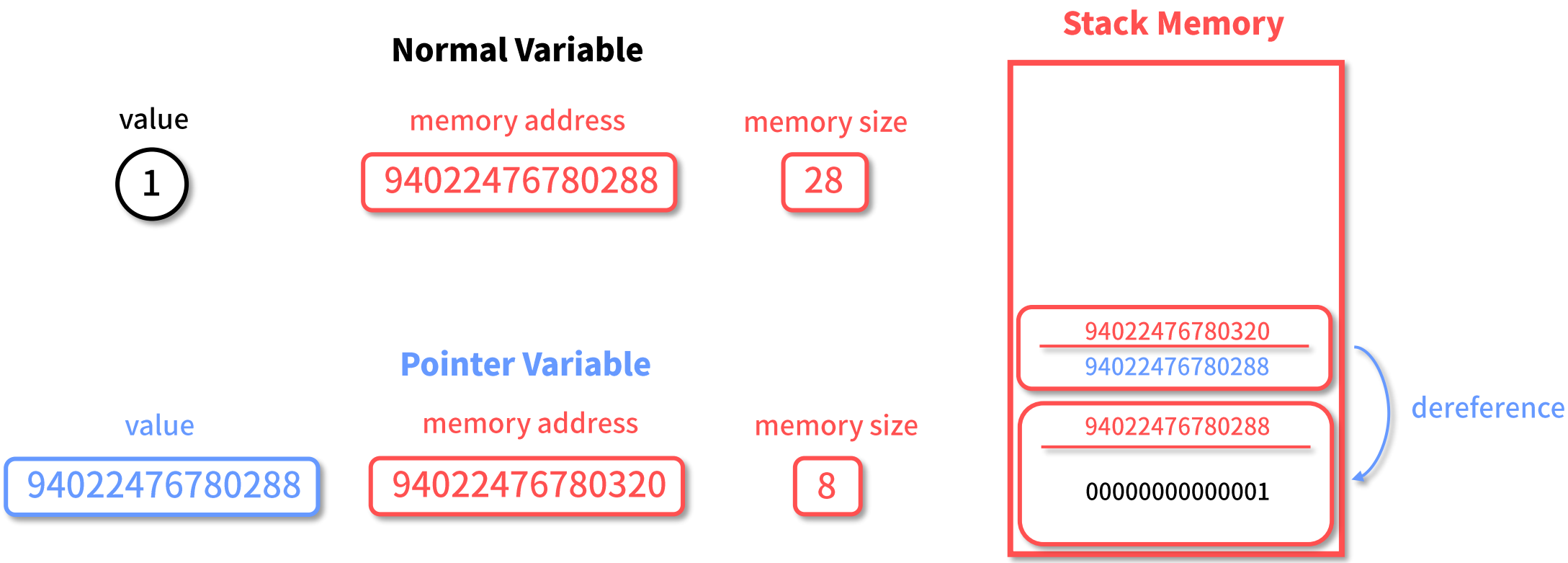
Overview

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

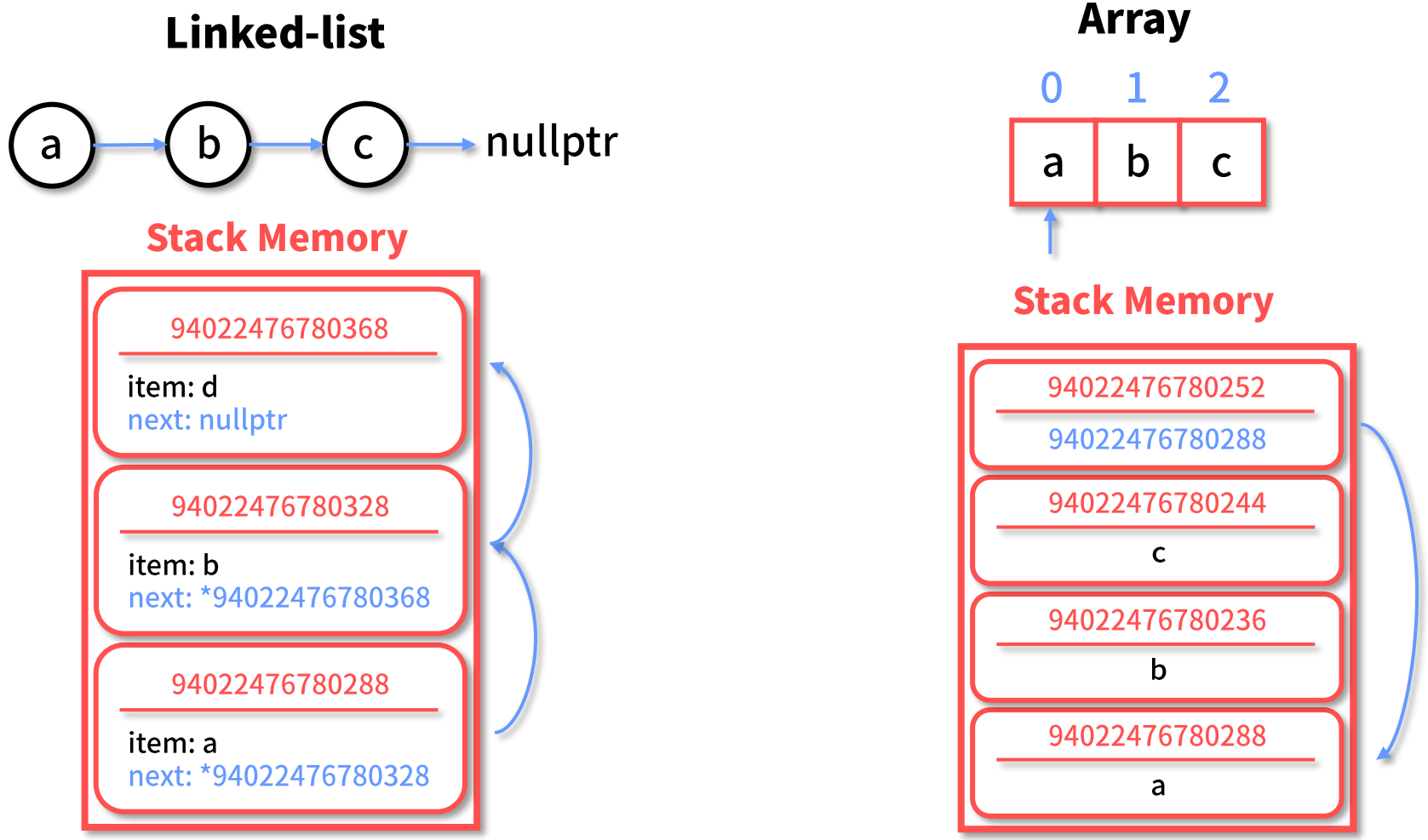
- Algorithms
  - Big-O Notation
  - Sorting
  - Graph Algorithms
  - Dynamic Programming

Variables, Linked-list and Array

Variables

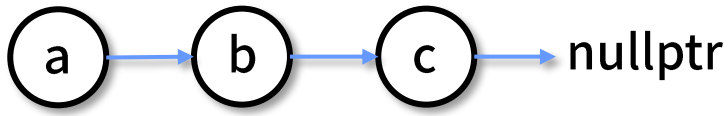

Normal Variable
value
1
memory address
94022476780288
memory size
28
Pointer Variable
value
94022476780288
memory address
94022476780320
memory size
8
Stack Memory
94022476780320
94022476780288
94022476780288
00000000000001
dereference

Linked-list and Array



Linked-list

Data structure



Initialization

```python
from __future__ import annotations
from typing import Any, Optional

...
class Node:
    def __init__(self, item: Any, next_node: Optional[Node] = None):
        self.item: Any = item
        self.next_node: Node = next_node
```

Big-O Notation

Big-O Notation (1)

- How do we estimate the running time without executing it.

```
a = 0
for i in range(0, 5):
  tmp = a + 1
  a = tmp
```

```
a = 0
for i in range(0, 5):
  a += 1
```

Big-O Notation (1)

- How do we estimate the running time without executing it.

```
a = 0
for i in range(0, 5):
  tmp = a + 1
  a = tmp
```

time complexity: 10T + 1

```
a = 0
for i in range(0, 5):
  a += 1
```

time complexity: 5T + 1

- How do we deal with different execution time of different command.
  - We assume they are the same (T) if they do not scale with the input.
  - But the example shown here should have the same time complexity.

Big-O Notation (2)

Formulation

$f(x) = O(g(x)) \text{ for } x \to \infty$

Finding Big-O notation

find positive integer M and n_0 such that
$f(n) \leq Mg(n)$
for all $n > n_0$

Example (1)

$f(x) = 2x$
if $M = 3, n_0 = 1, g(x) = x$
then
$f(n) = 2n \leq 3 \times g(x) = 3n$ for all $n \geq 1$

$\to f(n) = 2n = O(n)$

Example (2)

$f(x) = x + t, \quad t \geq 0$
if $M = 1, n_0 = 1 + \lceil t \rceil, g(x) = x$
then
$f(n) = n + t \leq 1 \times g(n) = n + \lceil t \rceil$ for all $n \geq 1$
$\to f(n) = O(n)$

Big-O Notation (3)

Comparison

NP-Complete NP-Hard

$$O(1) < O(\log^* n) < O(\log n) < O(n) < O(n \log^* n) < O(n \log n) < O(n^x) < O(x^n) < O(n!)$$

constant | logarithmic | n log-star n | polynomial | factorial

iterated logarithmic | linear | loglinear | exponential

$^*x > 1$

Iterated logarithmic

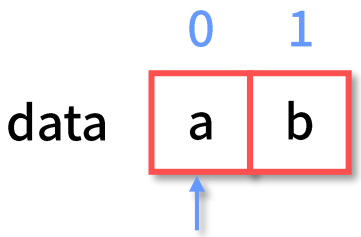
$$\log^* n := \begin{cases} 0 & \text{if } n \leq 1; \\ 1 + \log^*(\log n) & \text{if } n > 1 \end{cases}$$

Resize Array (1)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size
  for i in range(0, self.n):
    duplicate[i] = self.data[i]

  self.data = duplicate
  self.n = size
  return
```

Visualization



Command

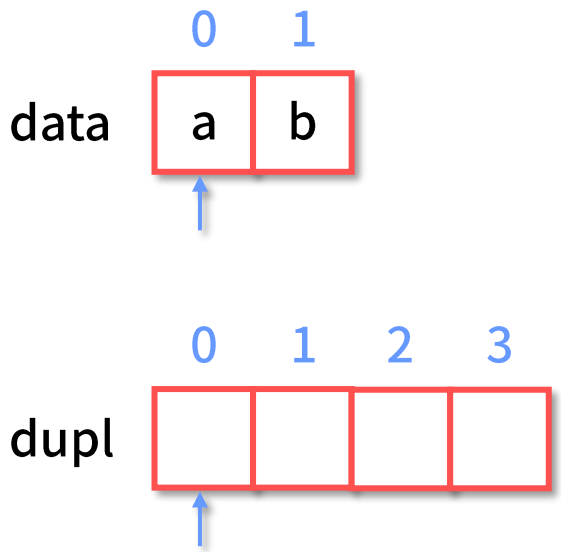
```
array.resize(4)
```

Resize Array (2)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size                    Size × T
  for i in range(0, self.n):
    duplicate[i] = self.data[i]

  self.data = duplicate
  self.n = size
  return
```

Visualization

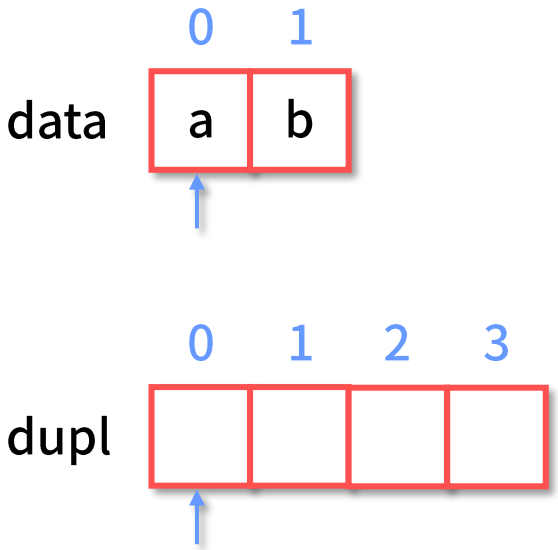


Resize Array (3)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size                    Size × T
  for i in range(0, self.n):                          T
    duplicate[i] = self.data[i]

  self.data = duplicate
  self.n = size
  return
```

Visualization



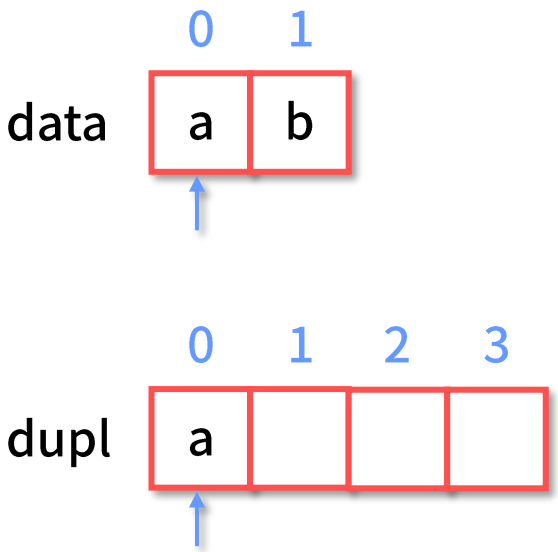
i = 0

Resize Array (3)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size                    Size × T
  for i in range(0, self.n):                          T
    duplicate[i] = self.data[i]                       T

  self.data = duplicate
  self.n = size
  return
```

Visualization



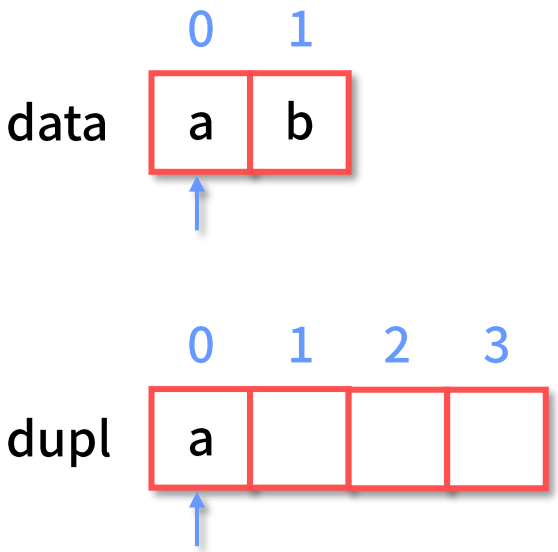
i = 0

Resize Array (4)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size                    Size × T
  for i in range(0, self.n):                          T
    duplicate[i] = self.data[i]                       T

  self.data = duplicate
  self.n = size
  return
```

Visualization



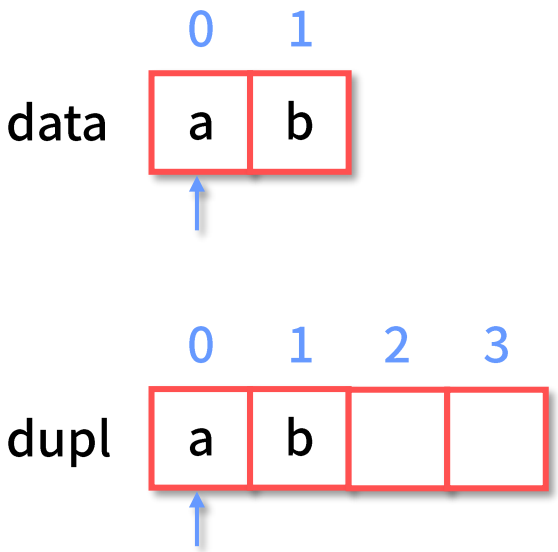
i = 1

Resize Array (5)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size                    Size × T
  for i in range(0, self.n):                          T
    duplicate[i] = self.data[i]                       T

  self.data = duplicate
  self.n = size
  return
```

Visualization



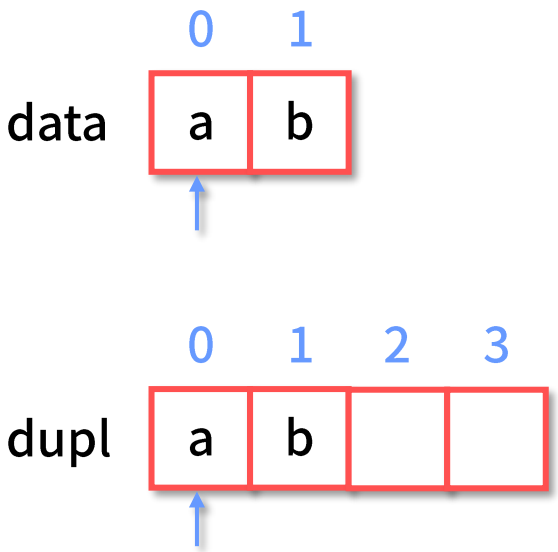
i = 1

Resize Array (6)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size                    Size × T
  for i in range(0, self.n):
    duplicate[i] = self.data[i]                Old Size × T

  self.data = duplicate
  self.n = size
  return
```

Visualization



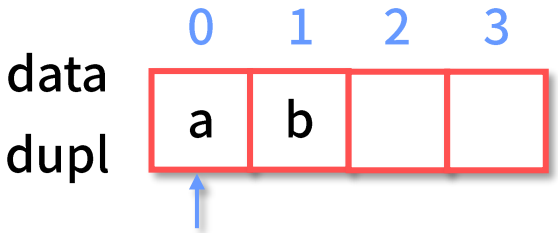
i = 1

Resize Array (7)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size                    Size × T
  for i in range(0, self.n):                          T
    duplicate[i] = self.data[i]                       T

  self.data = duplicate                               T
  self.n = size
  return
```

Visualization

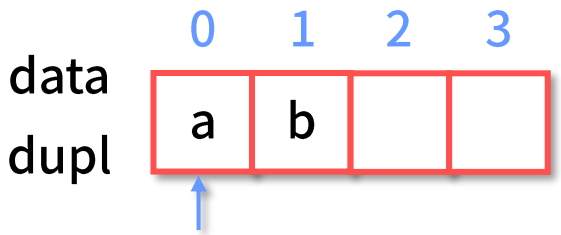


Resize Array (8)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size                    Size × T
  for i in range(0, self.n):                   Old Size × T
    duplicate[i] = self.data[i]

  self.data = duplicate                        T
  self.n = size                                T
  return
```

Visualization

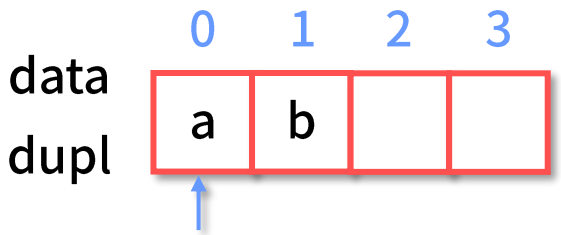


Resize Array (9)

Initialization

```
def resize(self, size: int = 0):
  duplicate = [None] * size                    Size × T
  for i in range(0, self.n):                   Old Size × T
    duplicate[i] = self.data[i]

  self.data = duplicate                        T
  self.n = size                                T
  return
```

Visualization



Time Complexity

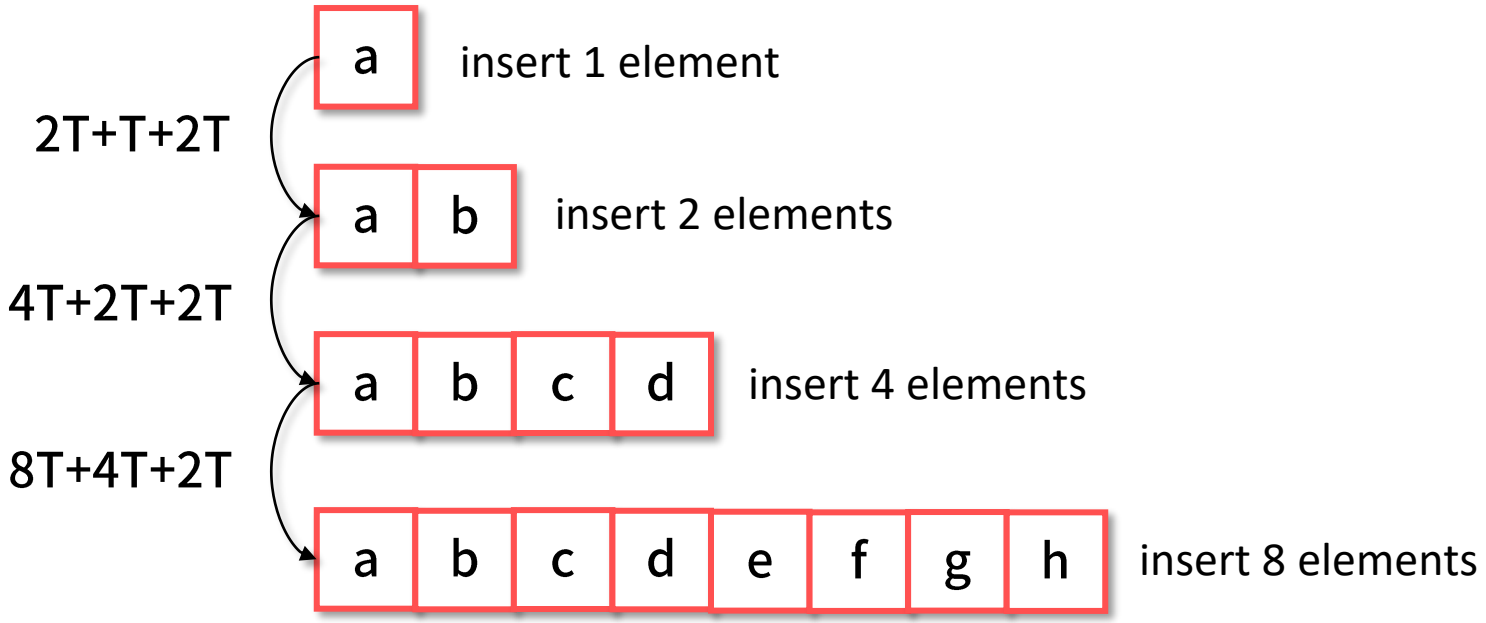
$$
\begin{aligned}
f(N_{new}) &= N_{new}T + N_{old}T + 2T \\
&= \frac{3}{2}N_{new}T + 2T \qquad (\text{let } N_{old} = \frac{1}{2}N_{new}) \\
&= O(N_{new})
\end{aligned}
$$

Amortized Time Complexity for Insertion

- If we resize the array twice the original size when it is full

Running Time (Resize)

Visualization



Total Time (Insert 2^k Element)

$$
\begin{aligned}
f(n=2^k) &= (2+\ldots+2^k)T + (2^0+\ldots+2^{k-1}) + 2kT \\
&= 2 \times 2^{k-1} + 1 \times 2^{k-2} + 2kT \\
&= 2^k + 2^{k-2} + 2kT \\
&= \frac{5}{4}2^k + 2kT
\end{aligned}
$$

Amortized Time (Insert 2^k Element)

$$\frac{f(n=2^k)}{2^k} = 5/4 + \frac{2kT}{2^k} \sim O(1)$$

Binary Search (1)

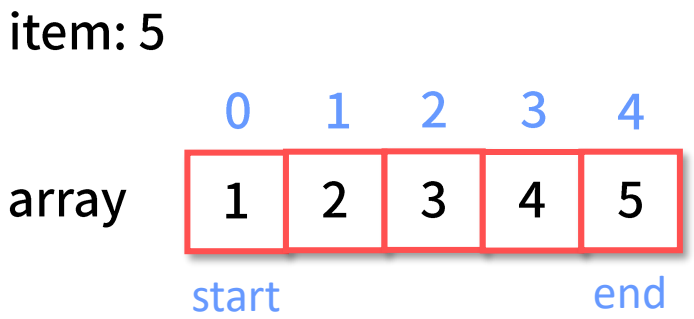
Function

```python
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:
    return -1
  middle = (start + end) // 2
  if sorted_array[middle] == item:
    return middle
  elif sorted_array[middle] > item:
    return binary_search(sorted_array, item, start, middle-1)
  elif sorted_array[middle] < item:
    return binary_search(sorted_array, item, middle+1, end)
```

Command

```python
array = [1, 2, 3, 4, 5]
binary_search(array, 5, 0, 4)
```

Visualization



Binary Search (2)

Function

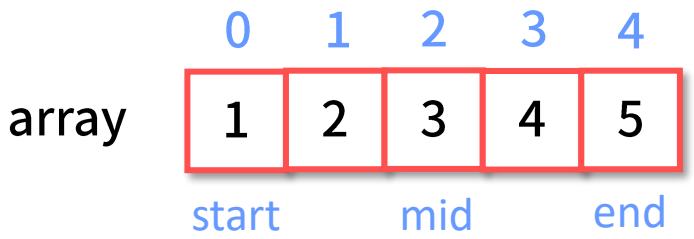
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                          T
    return -1                                              T
  middle = (start + end) // 2                              T
  if sorted_array[middle] == item:
    return middle
  elif sorted_array[middle] > item:
    return binary_search(sorted_array, item, start, middle-1)
  elif sorted_array[middle] < item:
    return binary_search(sorted_array, item, middle+1, end)
```

Function Call Stack

```
binary_search(array, 5, 0, 4)
```

Visualization

item: 5



Binary Search (2)

Function

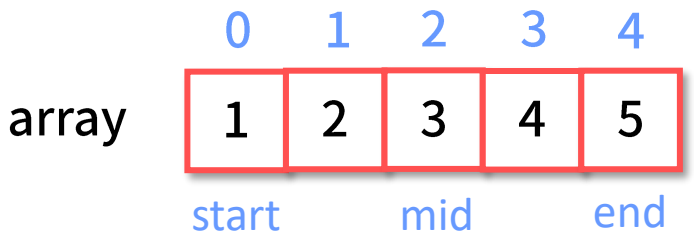
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                          T
    return -1                                              T
  middle = (start + end) // 2                              T
  if sorted_array[middle] == item:                         T
    return middle                                          T
  elif sorted_array[middle] > item:                        T
    return binary_search(sorted_array, item, start, middle-1)
  elif sorted_array[middle] < item:                        T
    return binary_search(sorted_array, item, middle+1, end)
```

Function Call Stack

```
binary_search(array, 5, 0, 4)
```

Visualization

item: 5



Binary Search (3)

Function

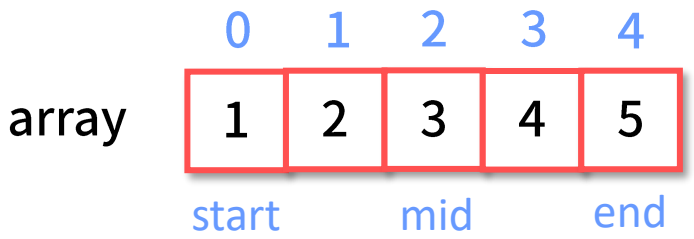
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                              T
    return -1                                                  T
  middle = (start + end) // 2                                  T
  if sorted_array[middle] == item:                             T
    return middle                                              T
  elif sorted_array[middle] > item:                            T
    return binary_search(sorted_array, item, start, middle-1)  T
  elif sorted_array[middle] < item:                            T
    return binary_search(sorted_array, item, middle+1, end)    T
```

Function Call Stack

```
binary_search(array, 5, 3, 4)
binary_search(array, 5, 0, 4)                                  at most 6T
```

Visualization

item: 5



Binary Search (4)

Function

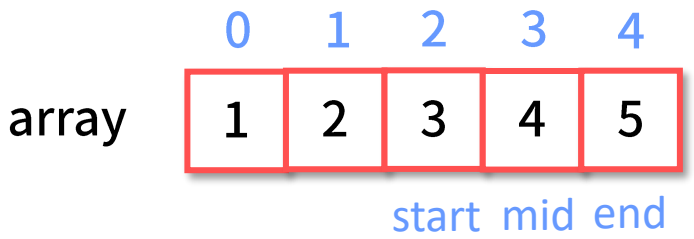
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                          T
    return -1                                              T
  middle = (start + end) // 2                              T
  if sorted_array[middle] == item:
    return middle
  elif sorted_array[middle] > item:
    return binary_search(sorted_array, item, start, middle-1)
  elif sorted_array[middle] < item:
    return binary_search(sorted_array, item, middle+1, end)
```

Function Call Stack

```
binary_search(array, 5, 3, 4)
binary_search(array, 5, 0, 4)                    at most 6T
```

Visualization

item: 5



Binary Search (5)

Function

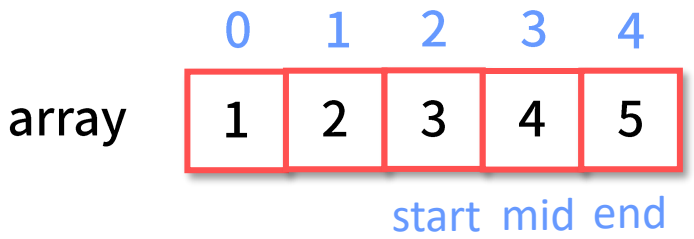
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                              T
    return -1                                                  T
  middle = (start + end) // 2                                  T
  if sorted_array[middle] == item:                             T
    return middle                                              T
  elif sorted_array[middle] > item:                            T
    return binary_search(sorted_array, item, start, middle-1)  T
  elif sorted_array[middle] < item:                            T
    return binary_search(sorted_array, item, middle+1, end)    T
```

Function Call Stack

```
binary_search(array, 5, 3, 4)    at most 6T
binary_search(array, 5, 0, 4)    at most 6T
```

Visualization

item: 5



Binary Search (6)

Function

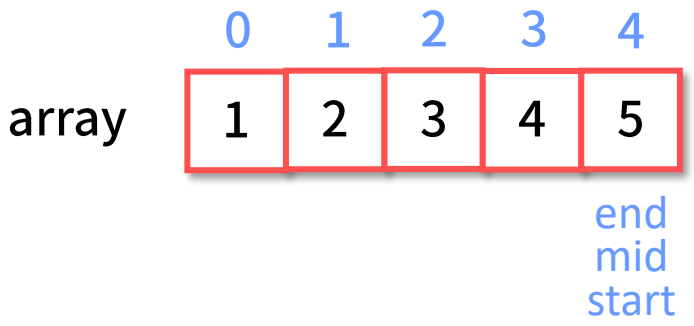
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                              T
    return -1                                                  T
  middle = (start + end) // 2                                  T
  if sorted_array[middle] == item:                             T
    return middle                                              T
  elif sorted_array[middle] > item:                            T
    return binary_search(sorted_array, item, start, middle-1)  T
  elif sorted_array[middle] < item:                            T
    return binary_search(sorted_array, item, middle+1, end)    T
```

Function Call Stack

```
binary_search(array, 5, 4, 4)    at most 9T
binary_search(array, 5, 3, 4)    at most 9T
binary_search(array, 5, 0, 4)    at most 9T
```

Visualization

item: 5



Binary Search (7)

Function

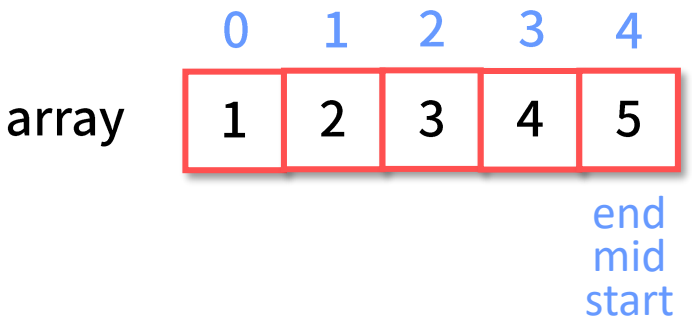
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                              T
    return -1                                                  T
  middle = (start + end) // 2                                  T
  if sorted_array[middle] == item:                             T
    return middle                                              T
  elif sorted_array[middle] > item:                            T
    return binary_search(sorted_array, item, start, middle-1)  T
  elif sorted_array[middle] < item:                            T
    return binary_search(sorted_array, item, middle+1, end)    T
```

Function Call Stack

```
binary_search(array, 5, 4, 4) return 4    at most 6T
binary_search(array, 5, 3, 4)             at most 6T
binary_search(array, 5, 0, 4)             at most 6T
```

Visualization

item: 5



Binary Search (8)

Function

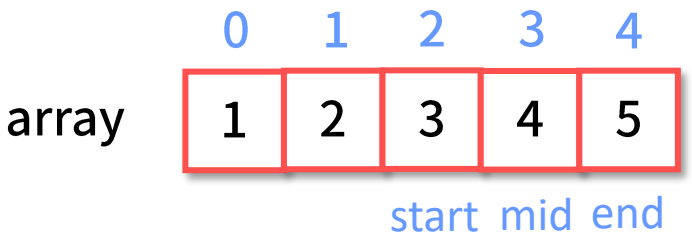
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                              T
    return -1                                                  T
  middle = (start + end) // 2                                  T
  if sorted_array[middle] == item:                             T
    return middle                                              T
  elif sorted_array[middle] > item:                            T
    return binary_search(sorted_array, item, start, middle-1)  T
  elif sorted_array[middle] < item:                            T
    return binary_search(sorted_array, item, middle+1, end)    T
```

Function Call Stack

```
binary_search(array, 5, 3, 4) return 4          at most 6T
binary_search(array, 5, 0, 4)                   at most 6T
```

Visualization

item: 5



Binary Search (9)

Function

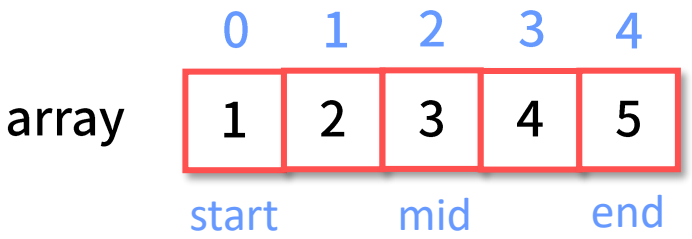
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                              T
    return -1                                                  T
  middle = (start + end) // 2                                  T
  if sorted_array[middle] == item:                             T
    return middle                                              T
  elif sorted_array[middle] > item:                            T
    return binary_search(sorted_array, item, start, middle-1)  T
  elif sorted_array[middle] < item:                            T
    return binary_search(sorted_array, item, middle+1, end)    T
```

Function Call Stack

```
binary_search(array, 5, 0, 4)  return 4                    at most 6T
```

Visualization

item: 5



Binary Search (10)

Function

```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                              T
    return -1                                                  T
  middle = (start + end) // 2                                  T
  if sorted_array[middle] == item:                             T
    return middle                                              T
  elif sorted_array[middle] > item:                            T
    return binary_search(sorted_array, item, start, middle-1)  T
  elif sorted_array[middle] < item:                            T
    return binary_search(sorted_array, item, middle+1, end)    T
```

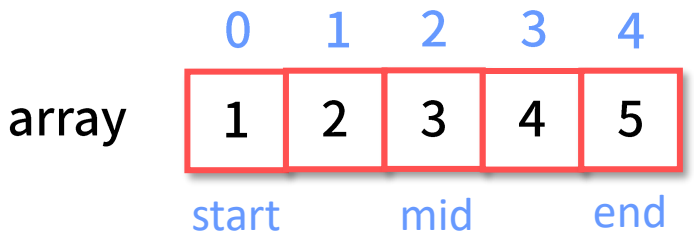
Function Call Stack

```
binary_search(array, 5, 0, 4)                                  at most 6T
```

Each binary search call take at most 6T time
How many time the function needs to be called

Visualization

item: 5



Binary Search (11)

Function

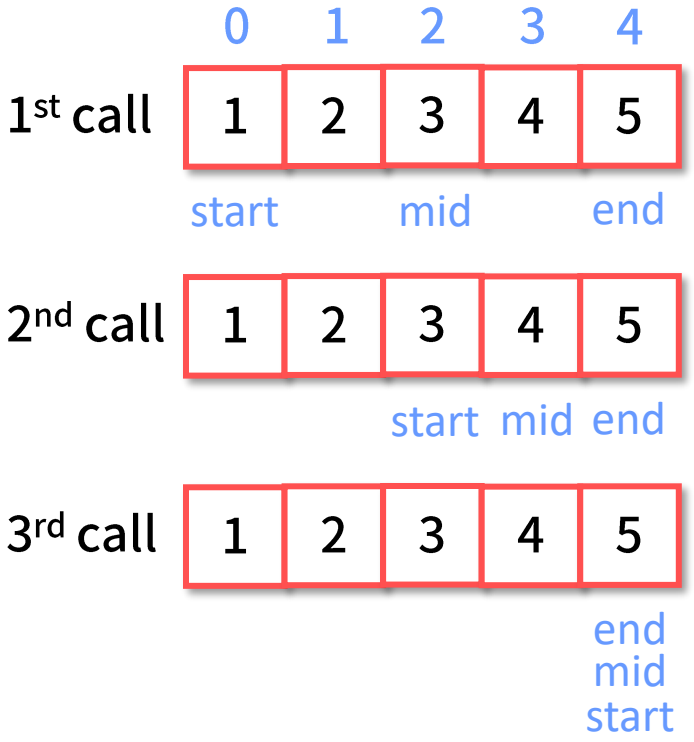
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                              T
    return -1                                                  T
  middle = (start + end) // 2                                  T
  if sorted_array[middle] == item:                             T
    return middle                                              T
  elif sorted_array[middle] > item:                            T
    return binary_search(sorted_array, item, start, middle-1)  T
  elif sorted_array[middle] < item:                            T
    return binary_search(sorted_array, item, middle+1, end)    T
```

Function Call Stack

```
binary_search(array, 5, 0, 4)                                  at most 6T
```

Each binary search call take at most 6T time
How many time the function needs to be called

Visualization



Binary Search (12)

Function

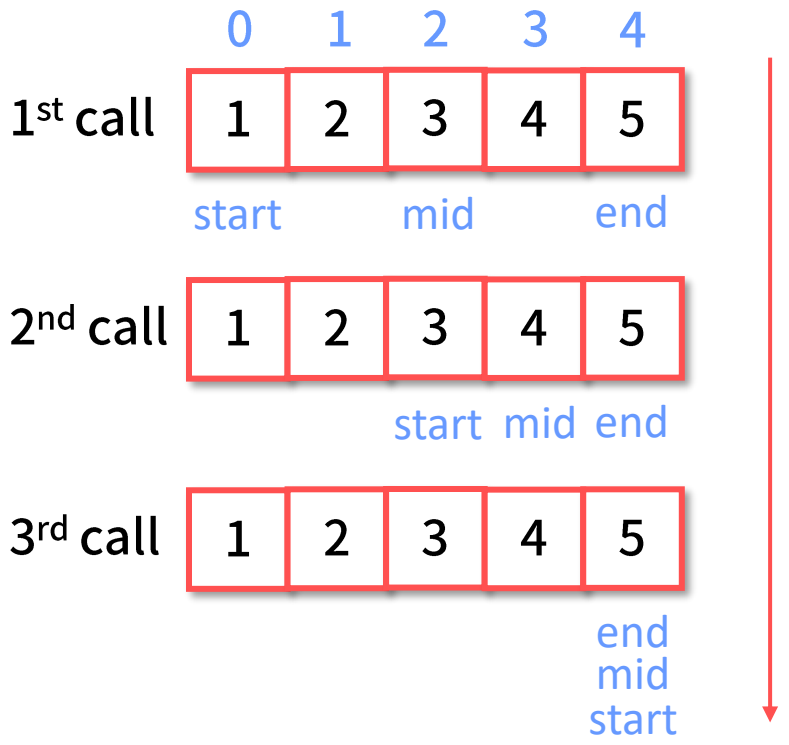
```
def binary_search(sorted_array, item, start: int, end: int):
  if end < start:                                              T
    return -1                                                  T
  middle = (start + end) // 2                                  T
  if sorted_array[middle] == item:                             T
    return middle                                              T
  elif sorted_array[middle] > item:                            T
    return binary_search(sorted_array, item, start, middle-1)  T
  elif sorted_array[middle] < item:                            T
    return binary_search(sorted_array, item, middle+1, end)    T
```

Function Call Stack

```
binary_search(array, 5, 0, 4)                                  at most 6T
```

Each binary search call take at most 6T time
How many time the function needs to be called

Visualization



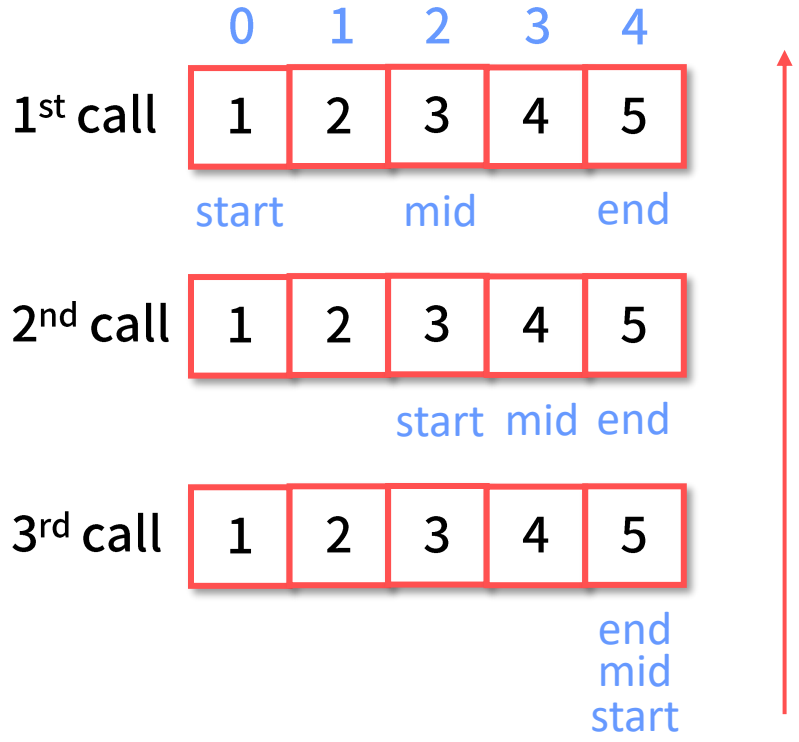
end subtract start will be
halved until end equals start

Binary Search (13)

- From the 1st function call to the last:
  - $end - start$ will be halved until end equals start.
- From the last function call to the 1st:
  - $end - start$ will expand by a factor of 2 until $end - start = n$
- The function needs to be called $\log_2 n$ times.
- Total time complexity:

$$f(n) = 6T log_2(n) = O(log_2(n))$$

Visualization



Time Complexity for Recursive Functions (1)

Theorem (Master Method) Consider the recurrence

$$T(n) = aT(n/b) + f(n), \tag{1}$$

where a, b are constants. Then

(A) If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = O(n^{\log_b a})$.

(B) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.

(C) If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if f satisfies the smoothness condition $af(n/b) \leq cf(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$.

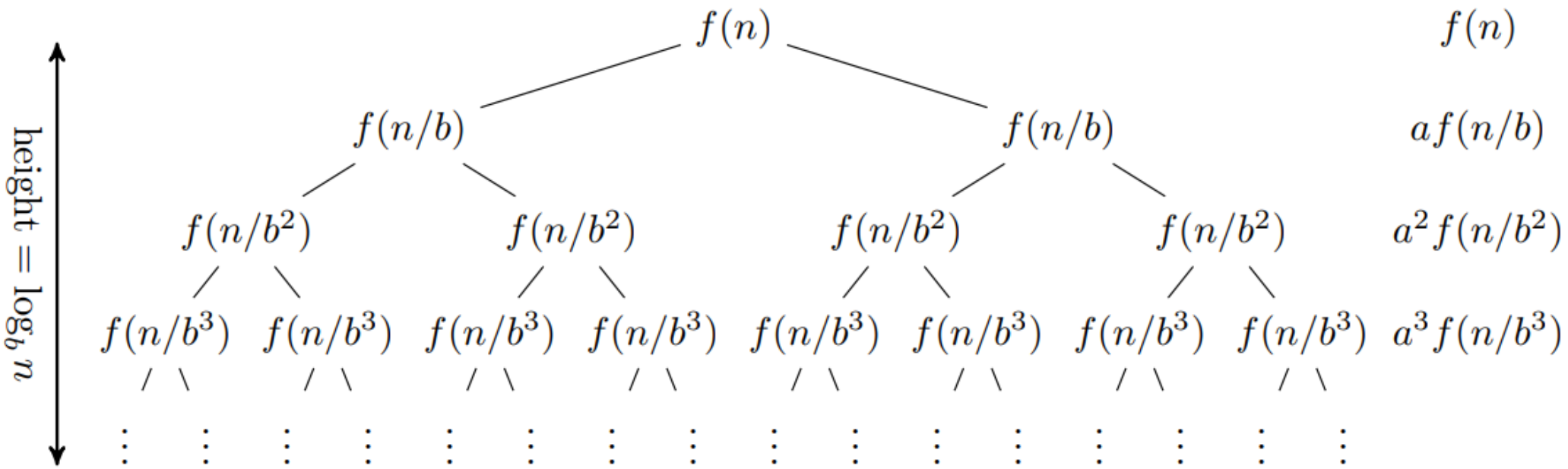
a : number of functions called in recursive function

b : scale factor for turning function into subproblems

University of Cornell CS3110

Time Complexity for Recursive Functions (2)

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}). \qquad (2)$$



Time Complexity for Recursive Functions (3)

Case (A). From (2), we have

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) \leq \sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} + O(n^{\log_b a}), \quad (3)$$

and

$$\begin{aligned}
\sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} &= n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i b^{-i \log_b a} b^{i\varepsilon} = n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i a^{-i} b^{i\varepsilon} \\
&= n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} b^{\varepsilon i} = n^{\log_b a - \varepsilon} \frac{b^{\varepsilon(\log_b n + 1)} - 1}{b^\varepsilon - 1} \\
&= n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon - 1}{b^\varepsilon - 1} \leq n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon}{b^\varepsilon - 1} = n^{\log_b a} \frac{b^\varepsilon}{b^\varepsilon - 1} \\
&= O(n^{\log_b a}).
\end{aligned}$$

Time Complexity for Recursive Functions (4)

Case (A). From (2), we have

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) \leq \sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} + O(n^{\log_b a}), \quad (3)$$

and

$$\begin{aligned}
\sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} &= n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i b^{-i \log_b a} b^{i\varepsilon} = n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i a^{-i} b^{i\varepsilon} \\
&= n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} b^{\varepsilon i} = n^{\log_b a - \varepsilon} \frac{b^{\varepsilon(\log_b n + 1)} - 1}{b^\varepsilon - 1} \\
&= n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon - 1}{b^\varepsilon - 1} \leq n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon}{b^\varepsilon - 1} = n^{\log_b a} \frac{b^\varepsilon}{b^\varepsilon - 1} \\
&= O(n^{\log_b a}).
\end{aligned}$$

Time Complexity for Recursive Functions (5)

Case (B). Here we have

$$\sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a} = n^{\log_b a} \sum_{i=0}^{\log_b n} a^i b^{-i \log_b a} = n^{\log_b a} \sum_{i=0}^{\log_b n} a^i a^{-i}$$
$$= n^{\log_b a} (\log_b n + 1) = \Theta(n^{\log_b a} \log n),$$

Combining this with (2) and the assumption of (B), to within constant factor bounds we have

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) = \sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a} + O(n^{\log_b a})$$
$$= \Theta(n^{\log_b a} \log n) + O(n^{\log_b a}) = \Theta(n^{\log_b a} \log n).$$

Time Complexity for Recursive Functions (6)

Case (C). The lower bound is immediate, because $f(n)$ is a term of the sum (2). For the upper bound, we will use the smoothness condition. This condition is satisfied by $f(n) = n^{\log_b a + \varepsilon}$ for any $\varepsilon > 0$ with $c = b^{-\varepsilon} < 1$:

$$af(n/b) = a(n/b)^{\log_b a + \varepsilon} = an^{\log_b a + \varepsilon} b^{-\log_b a} b^{-\varepsilon} = f(n)b^{-\varepsilon}.$$

In this case, we have $a^i f(n/b^i) \le c^i f(n)$ (easy induction on i using the smoothness condition), therefore

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) \le \sum_{i=0}^{\log_b n} c^i f(n) + O(n^{\log_b a})$$

$$\le f(n) \sum_{i=0}^{\infty} c^i + O(n^{\log_b a}) = f(n)\frac{1}{1-c} + O(n^{\log_b a}) = O(f(n)).$$

Time Complexity for Recursive Functions (7)

Binary Search

let $a = 1, b = 2, f(n) = 6T$
then $\log_2 1 = 0$,
since $f(n) = 6T = \Theta(n^{log_2 1}) = \Theta(1)$
use Case 2 of Master Theorem

$$\begin{aligned} T(n) &= \Theta(n^0 \log n) \\ &= \Theta(\log n) \\ &= O(\log n) \\ &= \Omega(\log n) \end{aligned}$$

Master Theorem

Theorem (Master Method) Consider the recurrence

$$T(n) = aT(n/b) + f(n), \tag{1}$$

where a, b are constants. Then

(A) If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = O(n^{\log_b a})$.

(B) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.

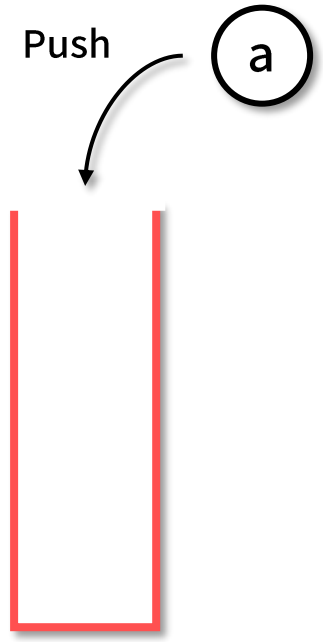
(C) If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if f satisfies the smoothness condition $af(n/b) \leq cf(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$.

Stack

Stack (1)

Insert

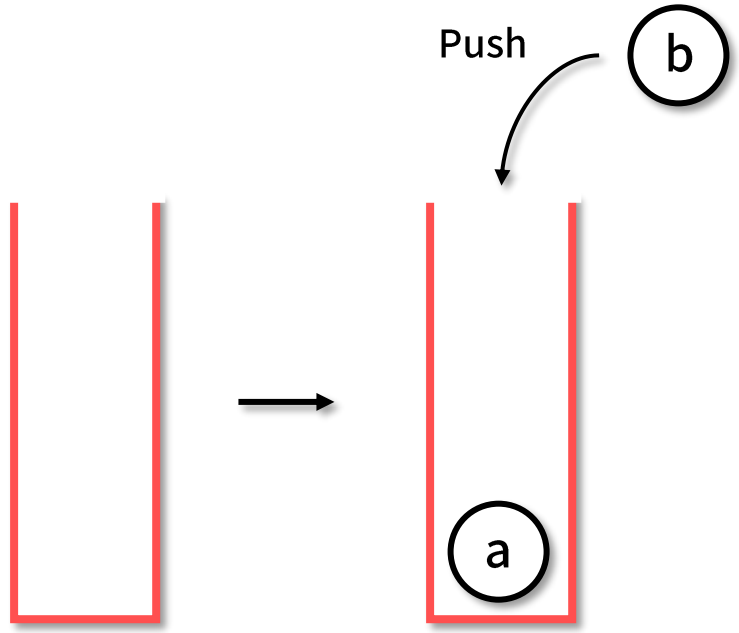
- First In, Last-Out (FILO) Data Structure



Stack (2)

Insert

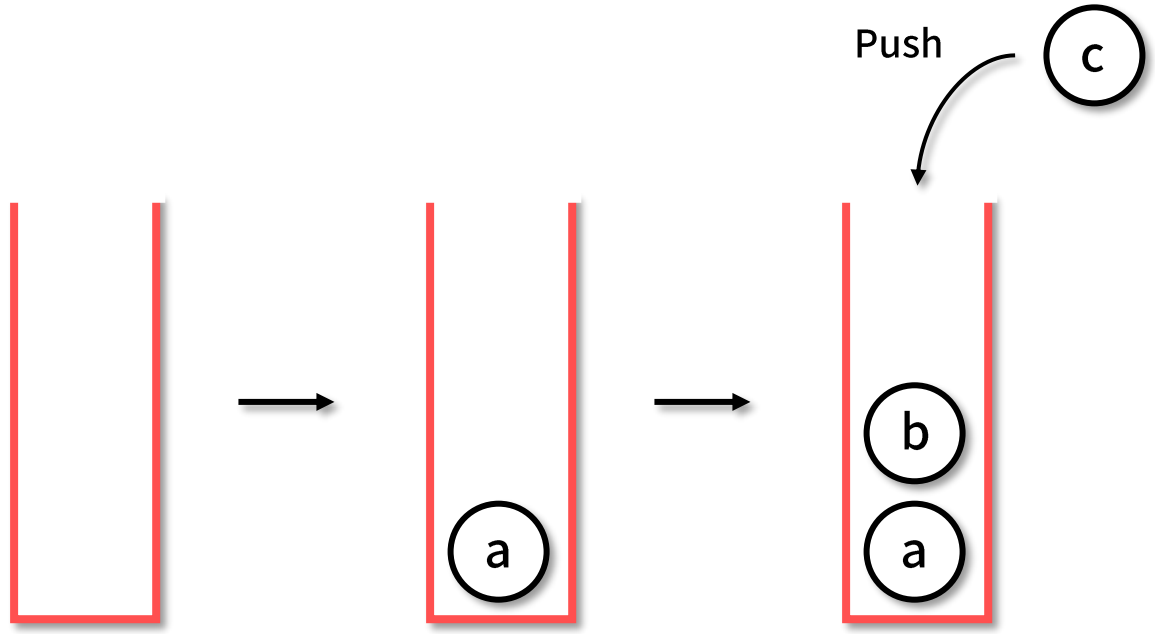
- First In, Last-Out (FILO) Data Structure



Stack (3)

Insert

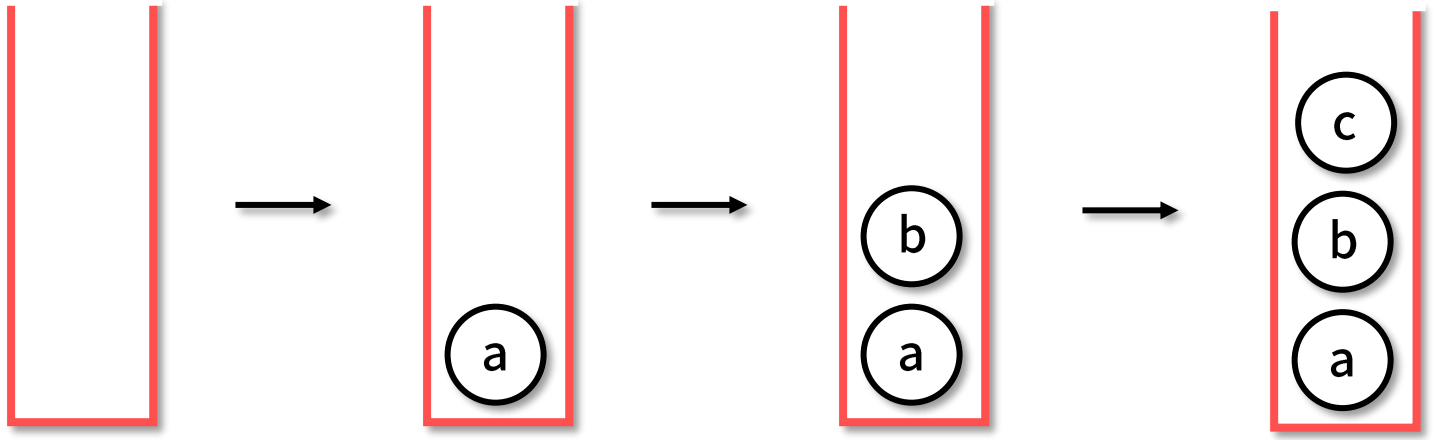
- First In, Last-Out (FILO) Data Structure



Stack (4)

Insert

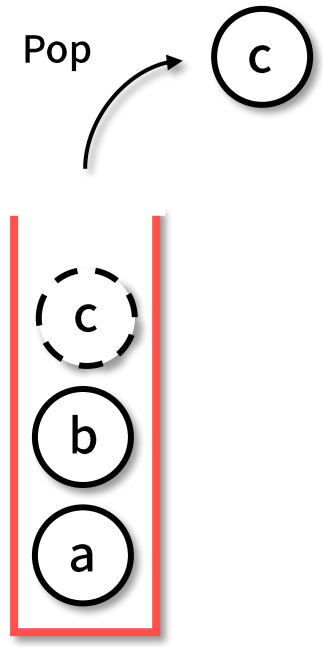
- First In, Last-Out (FILO) Data Structure



Stack (5)

Remove

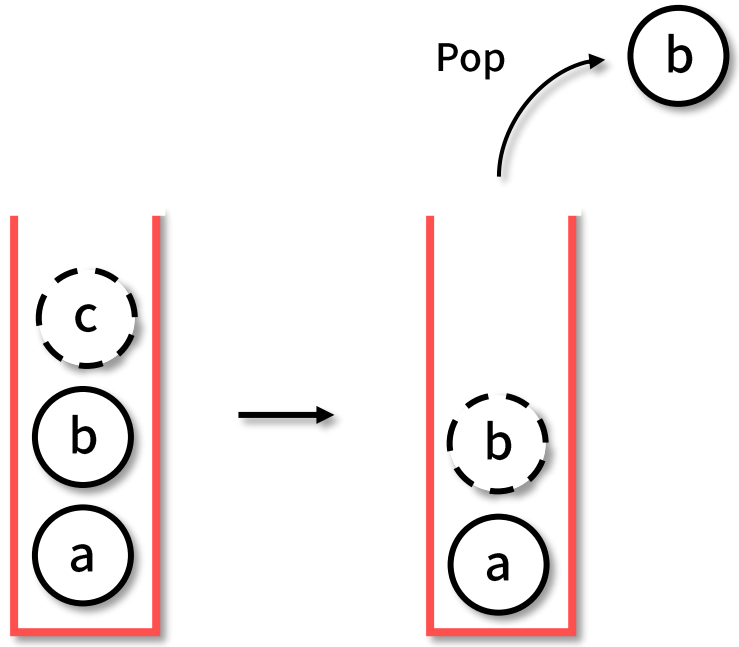
- First In, Last-Out (FILO) Data Structure



Stack (6)

Remove

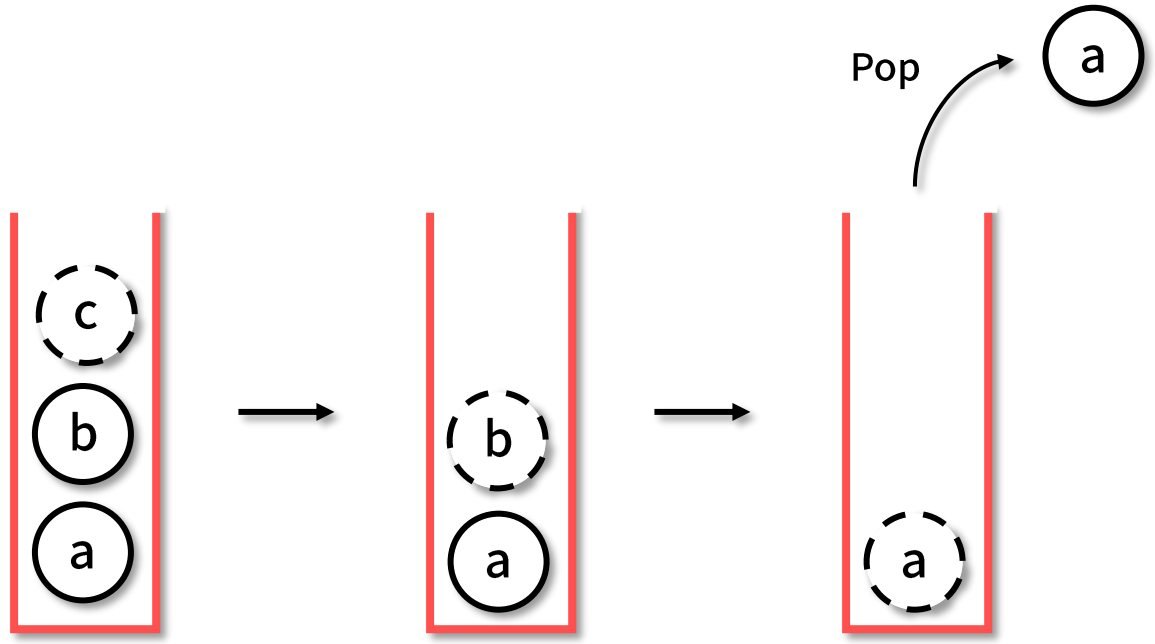
- First In, Last-Out (FILO) Data Structure



Stack (7)

Remove

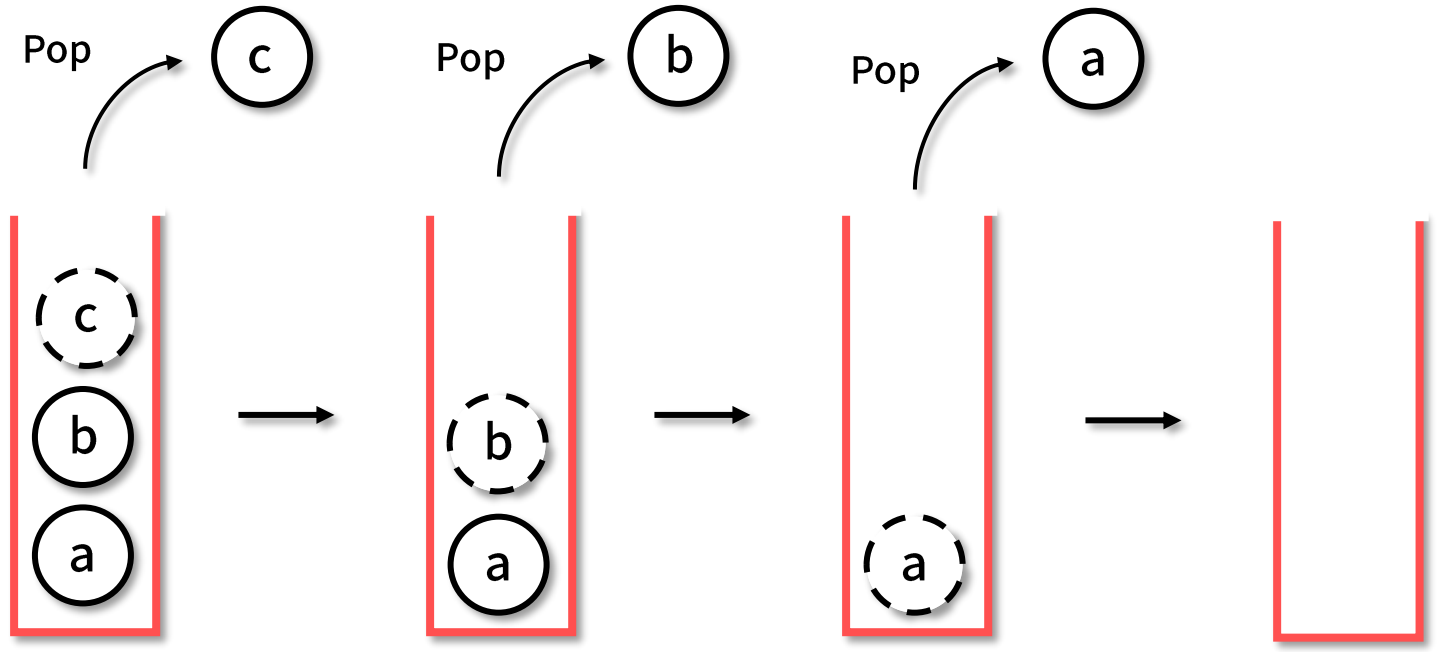
- First In, Last-Out (FILO) Data Structure



Stack (8)

Remove

- First In, Last-Out (FILO) Data Structure



Stack (using Linked-list)

Initialization

```
class LinkedStack(Iterable):
  def __init__(self):
    self.n: int = 0
    self.last: Optional[Node] = None

  def __iter__(self) -> Iterator[LinkedStack]: ···

  def __len__(self) -> int: ···

  def __str__(self) -> str: ···

  def is_empty(self) -> bool: ···          check if the Stack is empty

  def peek(self) -> Any: ···                  check last inserted item

  def pop(self) -> Any: ···    remove and return the last inserted item

  def push(self, item: Any) -> None: ···                  insert item
```

Visualization

n = 0

last → nullptr

Stack (using Linked-list)

Insert (1)

Insert

```
def push(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.last = node
    else:
        node.next_node = self.last
        self.last = node
    self.n += 1
    return
```

Command

```
stack = LinkedStack()
stack.push('a')
```

Visualization

n = 0

last → nullptr

node (a) → nullptr

item = a

Stack (using Linked-list)

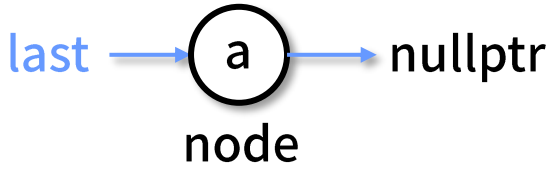
Insert (2)

Insert

```
def push(self, item: Any) -> None:
  node = Node(item)
  if self.is_empty():
    self.last = node
  else:
    node.next_node = self.last
    self.last = node
  self.n += 1
  return
```

Visualization (empty)

n = 0



item = a

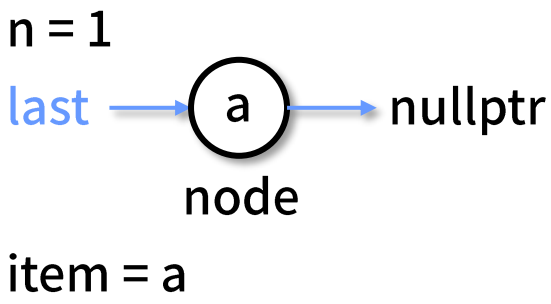
Stack (using Linked-list)

Insert (3)

Insert

```
def push(self, item: Any) -> None:
  node = Node(item)
  if self.is_empty():
    self.last = node
  else:
    node.next_node = self.last
    self.last = node
  self.n += 1
  return
```

Visualization (empty)



Stack (using Linked-list)

Insert (4)

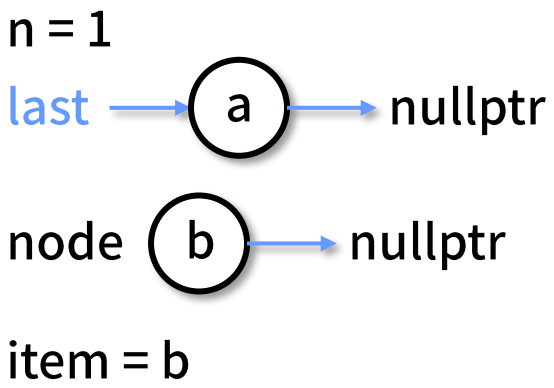
Insert

```
def push(self, item: Any) -> None:
  node = Node(item)
  if self.is_empty():
    self.last = node
  else:
    node.next_node = self.last
    self.last = node
  self.n += 1
  return
```

Command

```
stack.push('b')
```

Visualization (not empty)



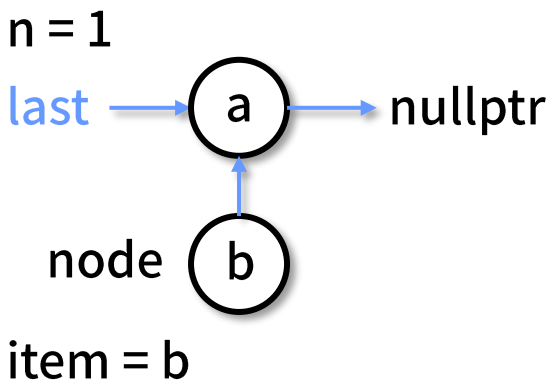
Stack (using Linked-list)

Insert (5)

Insert

```
def push(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.last = node
    else:
        node.next_node = self.last
        self.last = node
    self.n += 1
    return
```

Visualization (not empty)



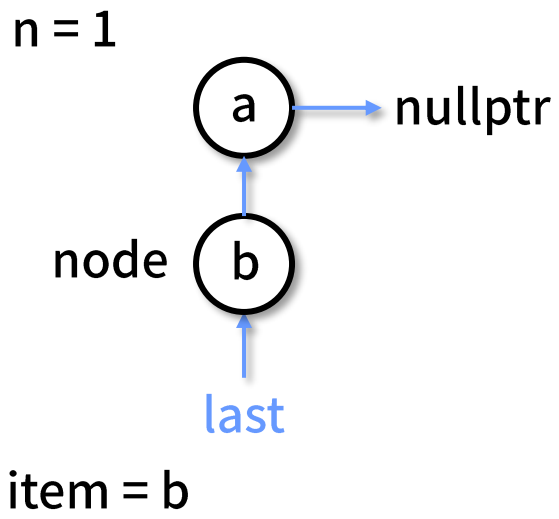
Stack (using Linked-list)

Insert (6)

Insert

```
def push(self, item: Any) -> None:
  node = Node(item)
  if self.is_empty():
    self.last = node
  else:
    node.next_node = self.last
    self.last = node
  self.n += 1
  return
```

Visualization (not empty)



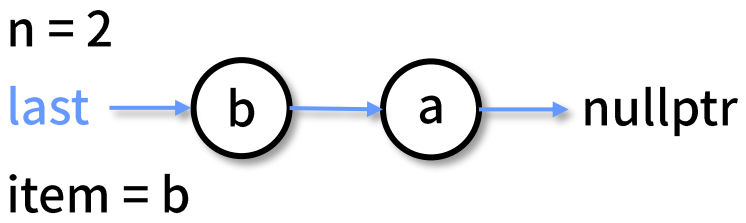
Stack (using Linked-list)

Insert (7)

Insert

```
def push(self, item: Any) -> None:
  node = Node(item)
  if self.is_empty():
    self.last = node
  else:
    node.next_node = self.last
    self.last = node
  self.n += 1
  return
```

Visualization (not empty)



Stack (using Linked-list)

Remove (1)

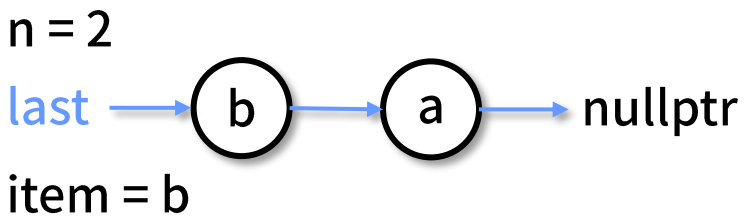
Remove

```
def pop(self) -> Any:
    item = self.last.item
    self.last = self.last.next_node
    self.n -= 1
    return item
```

Command

```
stack.pop()
```

Visualization (not empty)



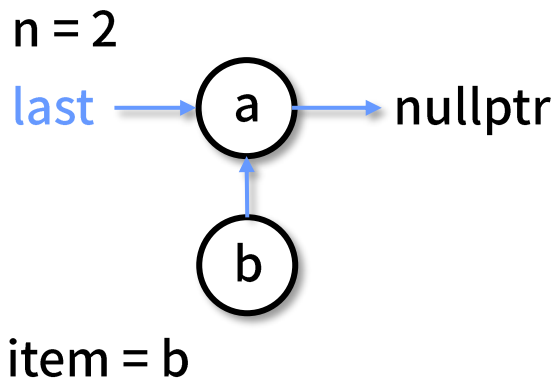
Stack (using Linked-list)

Remove (2)

Remove

```
def pop(self) -> Any:
    item = self.last.item
    self.last = self.last.next_node
    self.n -= 1
    return item
```

Visualization (not empty)



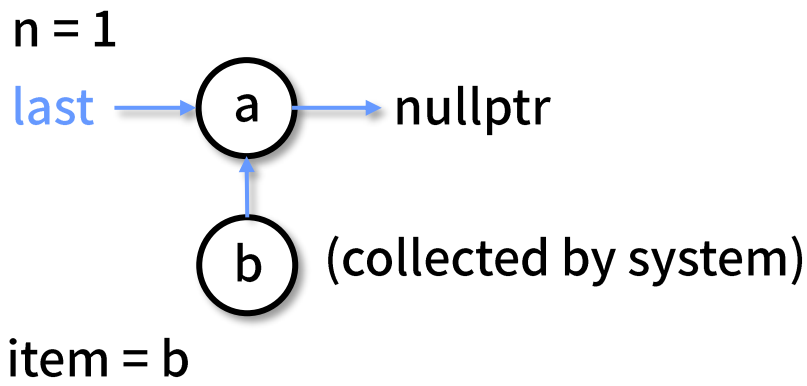
Stack (using Linked-list)

Remove (3)

Remove

```
def pop(self) -> Any:
    item = self.last.item
    self.last = self.last.next_node
    self.n -= 1
    return item
```

Visualization (not empty)



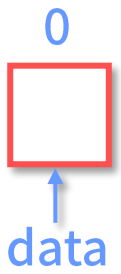
Stack (using Array)

Initialization

```
class Stack(Iterable):
  def __init__(self):
    self.n: int = 0
    self.data: Array = Array(1)

  def __len__(self) -> int: ···

  def __str__(self) -> str: ···

  def __iter__(self) -> Iterator[Stack]: ···

  def is_empty(self) -> bool: ···                check if the Stack is empty

  def peek(self) -> Any: ···                     check last inserted item

  def pop(self) -> Any: ···          remove and return the last inserted item

  def push(self, item: Any) -> None: ···                    insert item
```

Visualization

n = 0



Stack (using Array)

Insert (1)

Insert

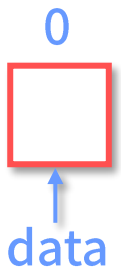
```
def push(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.data[self.n] = item
    self.n += 1
    return
```

Command

```
stack = Stack()
stack.push('a')
```

Visualization (w/o Resize)

n = 0 item = a



Stack (using Array)

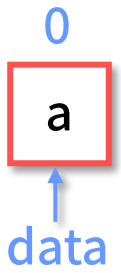
Insert (2)

Insert

```
def push(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.data[self.n] = item
    self.n += 1
    return
```

Visualization (w/o Resize)

n = 0 item = a



Stack (using Array)

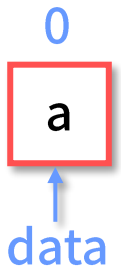
Insert (3)

Insert

```
def push(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.data[self.n] = item
    self.n += 1
    return
```

Visualization (w/o Resize)

n = 1 item = a



Stack (using Array)

Insert (4)

Insert

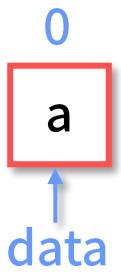
```
def push(self, item: Any) -> None:
  if self.n == len(self.data):
    self.data.resize(2 * self.n)
  self.data[self.n] = item
  self.n += 1
  return
```

Command

```
stack.push('b')
```

Visualization (w/ Resize)

n = 1 item = b



Stack (using Array)

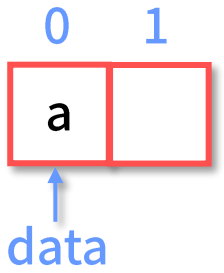
Insert (5)

Insert

```
def push(self, item: Any) -> None:
  if self.n == len(self.data):
    self.data.resize(2 * self.n)
  self.data[self.n] = item
  self.n += 1
  return
```

Visualization (w/ Resize)

n = 1 item = b



Stack (using Array)

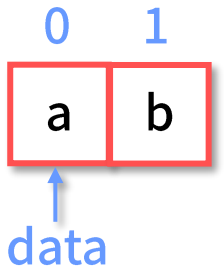
Insert (6)

Insert

```
def push(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.data[self.n] = item
    self.n += 1
    return
```

Visualization (w/ Resize)

n = 1 item = b



Stack (using Array)

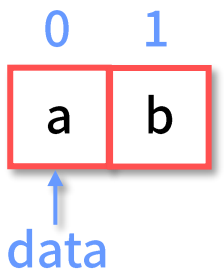
Insert (7)

Insert

```
def push(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.data[self.n] = item
    self.n += 1
    return
```

Visualization (w/ Resize)

n = 2



Stack (using Array)

Insert (8)

Insert

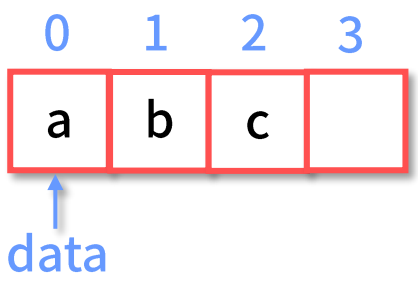
```
def push(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.data[self.n] = item
    self.n += 1
    return
```

Command

```
stack.push('c')
```

Visualization (w/ Resize)

n = 3



Stack (using Array)

Insert (9)

Insert

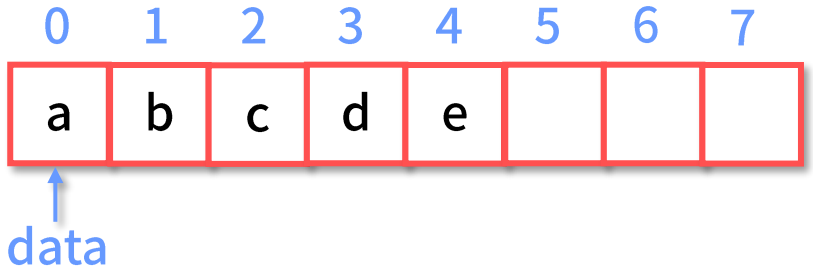
```
def push(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.data[self.n] = item
    self.n += 1
    return
```

Command

```
stack.push('d')
stack.push('e')
```

Visualization (w/ Resize)

n = 5



Stack (using Array)

Remove (1)

Remove

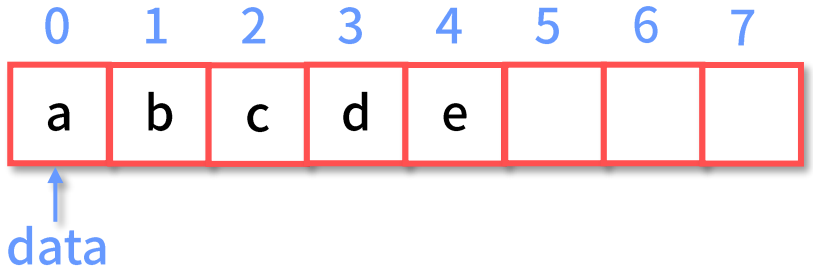
```
def pop(self) -> Any:
    item = self.data[self.n-1]
    self.data[self.n-1] = None
    self.n -= 1
    if self.n > 0 and self.n <= len(self.data) // 4:
        self.data.resize(len(self.data) // 2)
    return item
```

Command

```
stack.pop()
```

Visualization (w/o Resize)

n = 5



item = e

Stack (using Array)

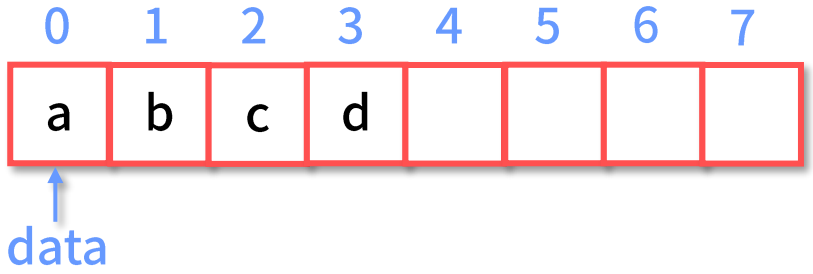
Remove (2)

Remove

```
def pop(self) -> Any:
    item = self.data[self.n-1]
    self.data[self.n-1] = None
    self.n -= 1
    if self.n > 0 and self.n <= len(self.data) // 4:
        self.data.resize(len(self.data) // 2)
    return item
```

Visualization (w/o Resize)

n = 5



item = e

Stack (using Array)

Remove (3)

Remove

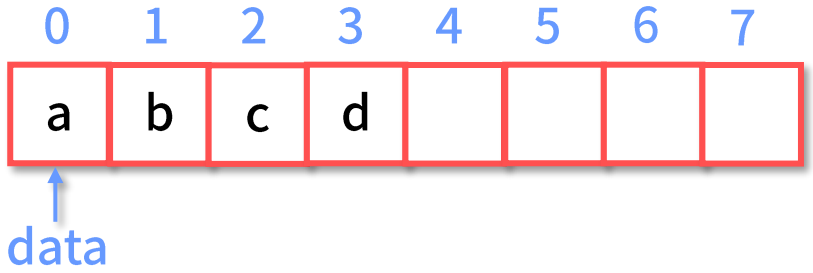
```
def pop(self) -> Any:
    item = self.data[self.n-1]
    self.data[self.n-1] = None
    self.n -= 1
    if self.n > 0 and self.n <= len(self.data) // 4:
        self.data.resize(len(self.data) // 2)
    return item
```

Command

```
stack.pop()
stack.pop()
```

Visualization (w/o Resize)

n = 4



item = e

Stack (using Array)

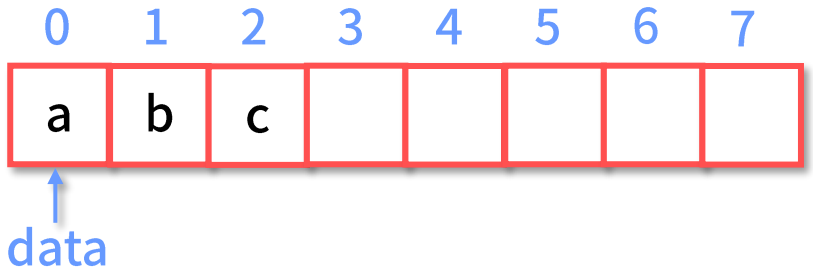
Remove (4)

Remove

```
def pop(self) -> Any:
    item = self.data[self.n-1]
    self.data[self.n-1] = None
    self.n -= 1
    if self.n > 0 and self.n <= len(self.data) // 4:
        self.data.resize(len(self.data) // 2)
    return item
```

Visualization (w/ Resize)

n = 3



item = c

Stack (using Array)

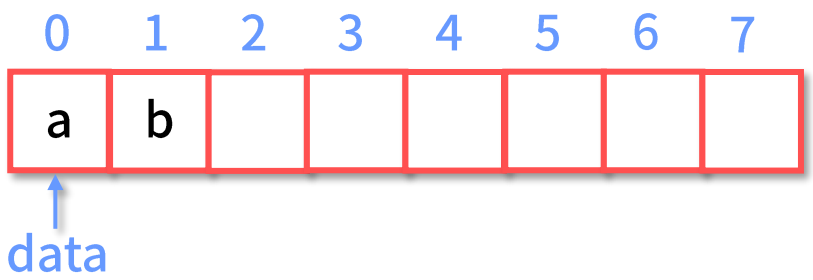
Remove (5)

Remove

```
def pop(self) -> Any:
    item = self.data[self.n-1]
    self.data[self.n-1] = None
    self.n -= 1
    if self.n > 0 and self.n <= len(self.data) // 4:
        self.data.resize(len(self.data) // 2)
    return item
```

Visualization (w/ Resize)

n = 3



item = c

Stack (using Array)

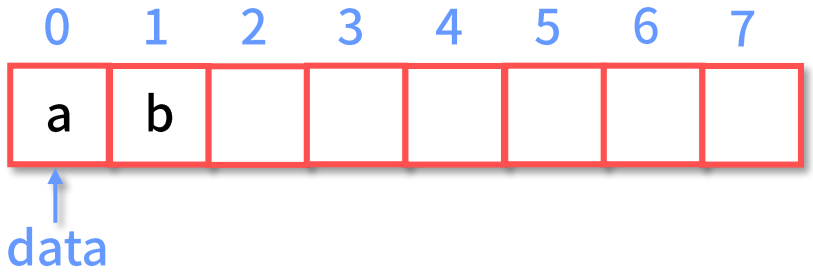
Remove (6)

Remove

```
def pop(self) -> Any:
    item = self.data[self.n-1]
    self.data[self.n-1] = None
    self.n -= 1
    if self.n > 0 and self.n <= len(self.data) // 4:
        self.data.resize(len(self.data) // 2)
    return item
```

Visualization (w/ Resize)

n = 2



item = c

Stack (using Array)

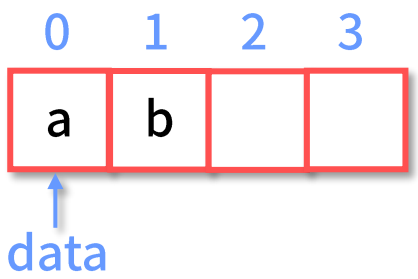
Remove (7)

Remove

```
def pop(self) -> Any:
  item = self.data[self.n-1]
  self.data[self.n-1] = None
  self.n -= 1
  if self.n > 0 and self.n <= len(self.data) // 4:
    self.data.resize(len(self.data) // 2)
  return item
```

Visualization (w/ Resize)

n = 2



item = c

Stack

Summary

Time Complexity

	Linked-list	Array
Initialization	O(1)	O(1)
Push	O(1)	O(1)*
Pop	O(1)	O(1)*
Peek	O(1)	O(1)

*amortized time complexity

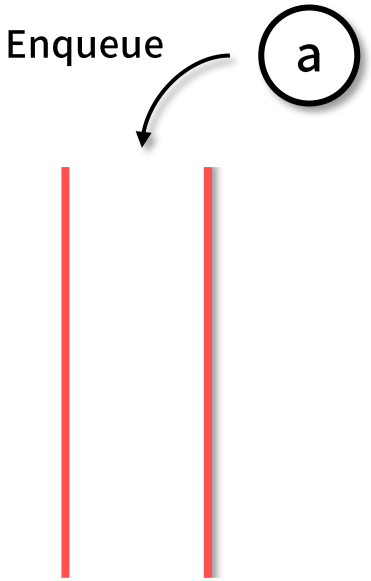
Stack (Practice)

1. Valid Parathesis (Leetcode Problem 20)
2. Min Stack (Leetcode Problem 155)

Queue

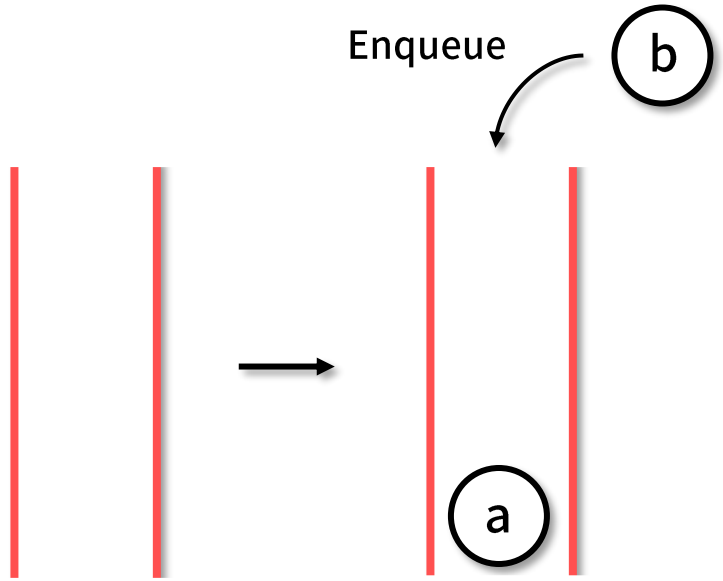
Queue

- First In, First-Out (FIFO) Data Structure



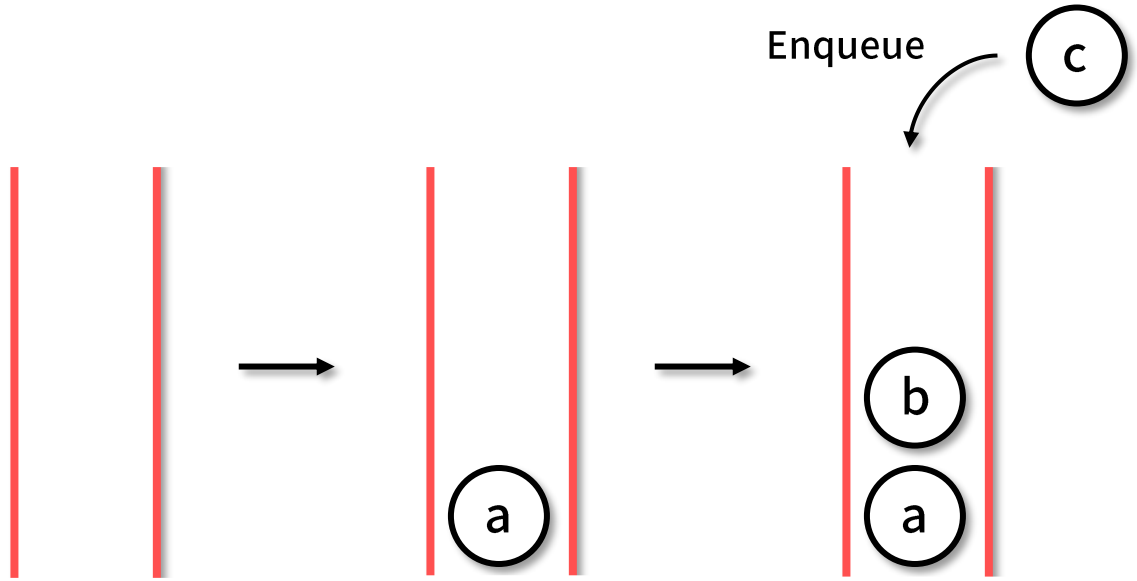
Queue

- First In, First-Out (FIFO) Data Structure



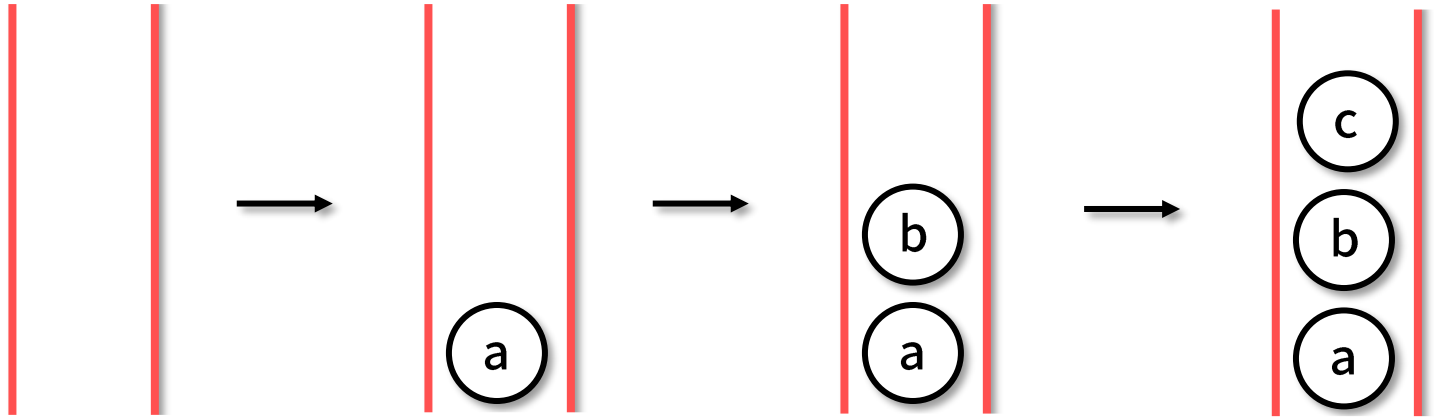
Queue

- First In, First-Out (FIFO) Data Structure



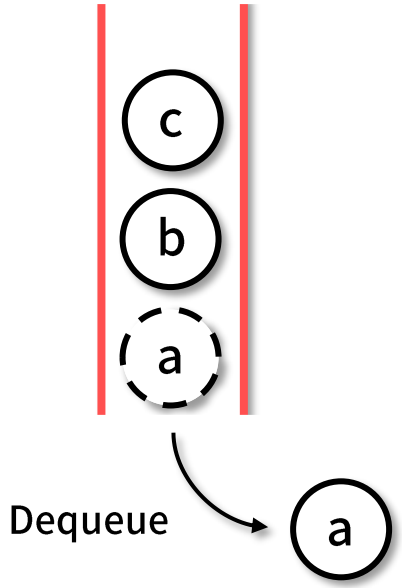
Queue

- First In, First-Out (FIFO) Data Structure



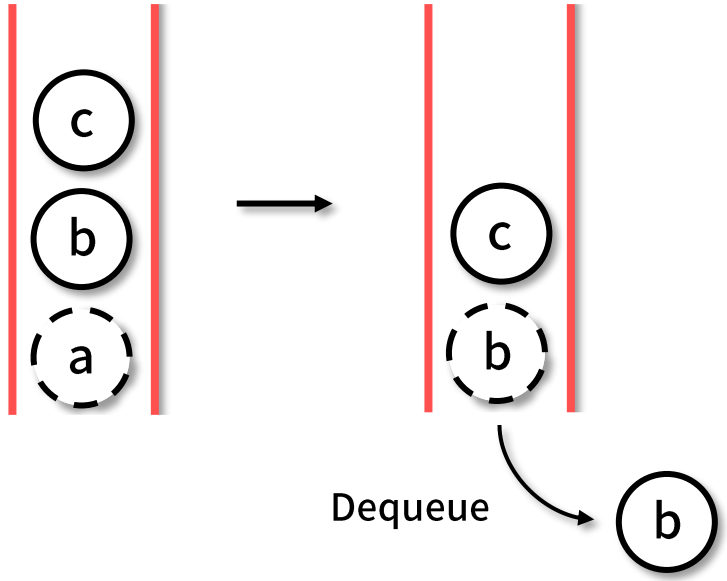
Queue

- First In, First-Out (FIFO) Data Structure



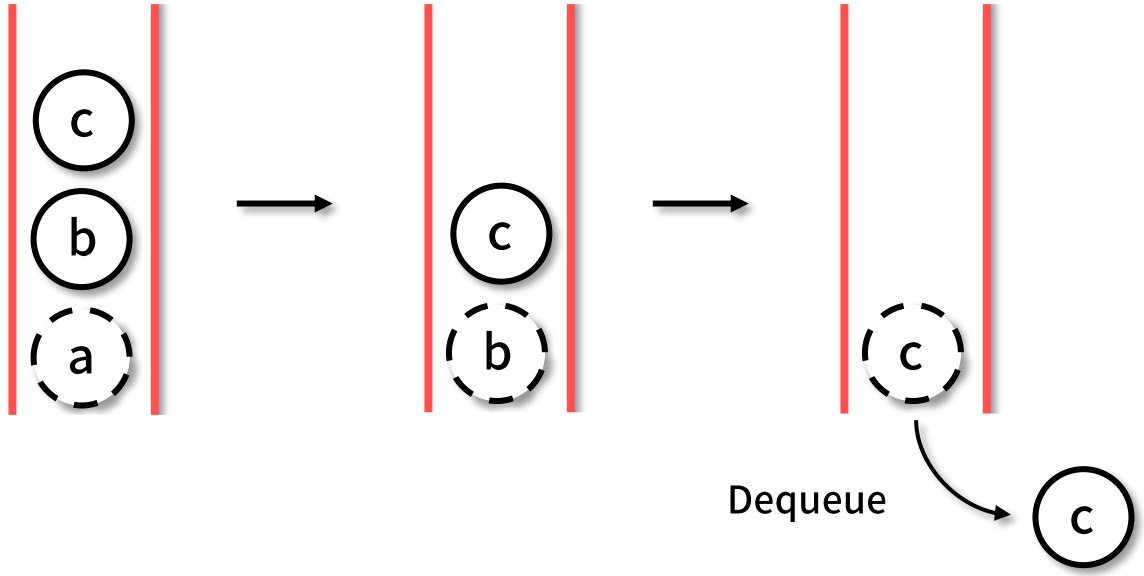
Queue

- First In, First-Out (FIFO) Data Structure



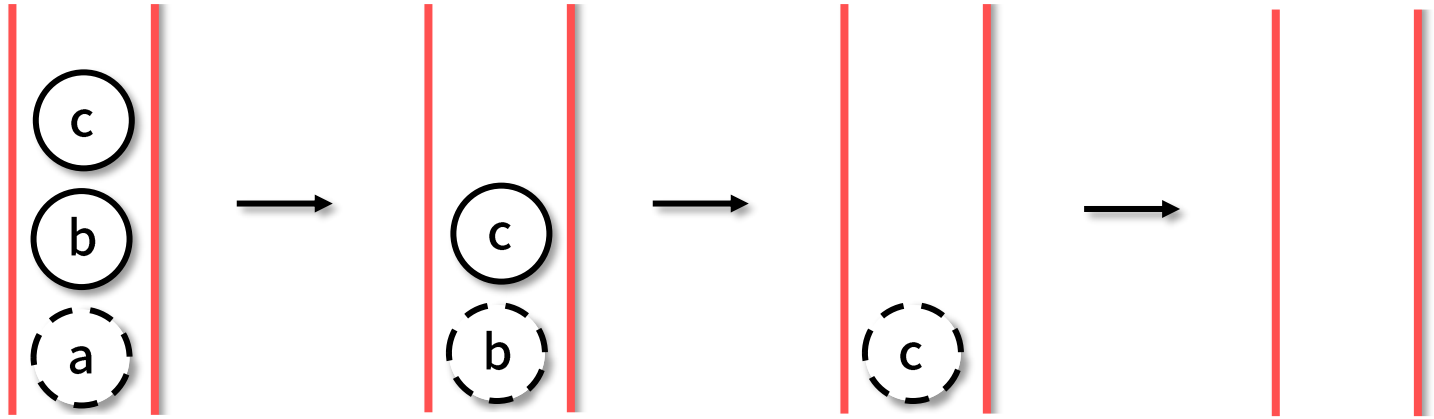
Queue

- First In, First-Out (FIFO) Data Structure



Queue

- First In, First-Out (FIFO) Data Structure



Queue (using Linked-list)

Initialization

```
class LinkedQueue(Iterable):
  def __init__(self):
    self.n: int = 0
    self.first: Optional[Node] = None
    self.last: Optional[Node] = None

  def __iter__(self) -> Iterator[LinkedQueue]: ···

  def __len__(self) -> int: ···

  def __str__(self) -> str: ···

  def is_empty(self) -> bool: ···                    check if the Queue is empty

  def peek(self) -> Any: ···                         check first inserted item

  def dequeue(self) -> Any: ···          remove and return the first inserted item

  def enqueue(self, item: Any) -> None: ···                    insert item
```

Visualization

n = 0

first → nullptr

last → nullptr

Queue (using Linked-list)

Insert (1)

Insert

```
def enqueue(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.first = node
    else:
        self.last.next_node = node
    self.last = node
    self.n += 1
    return
```

Command

```
queue = LinkedQueue()
queue.enqueue('a')
```

Visualization

n = 0

first → nullptr

last → nullptr

node (a) → nullptr

item = a

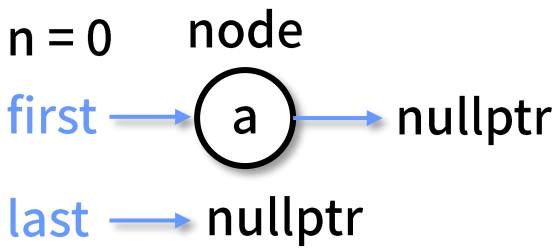
Queue (using Linked-list)

Insert (2)

Insert

```
def enqueue(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.first = node
    else:
        self.last.next_node = node
    self.last = node
    self.n += 1
    return
```

Visualization



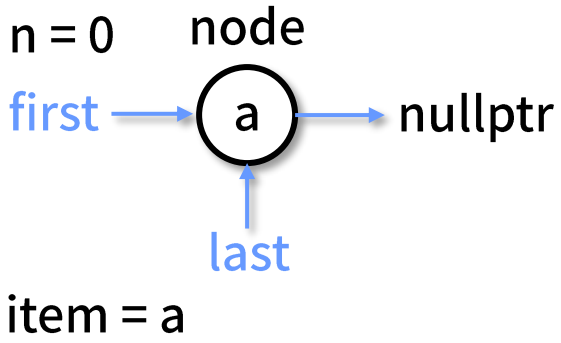
Queue (using Linked-list)

Insert (3)

Insert

```
def enqueue(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.first = node
    else:
        self.last.next_node = node
    self.last = node
    self.n += 1
    return
```

Visualization



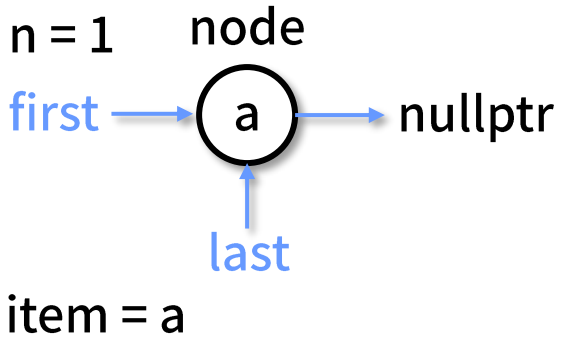
Queue (using Linked-list)

Insert (4)

Insert

```
def enqueue(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.first = node
    else:
        self.last.next_node = node
    self.last = node
    self.n += 1
    return
```

Visualization



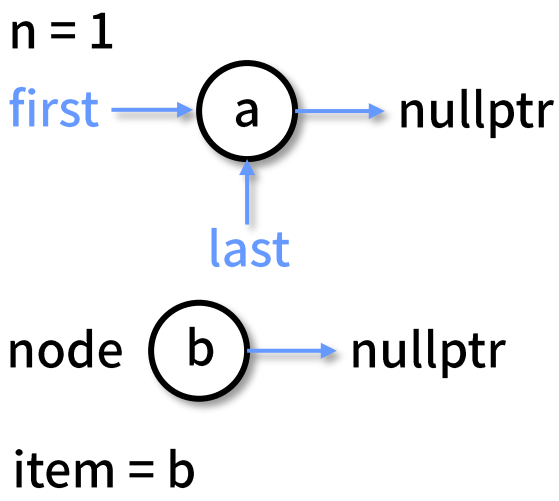
Queue (using Linked-list)

Insert (5)

Insert

```
def enqueue(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.first = node
    else:
        self.last.next_node = node
    self.last = node
    self.n += 1
    return
```

Visualization



Command

```
queue.enqueue('b')
```

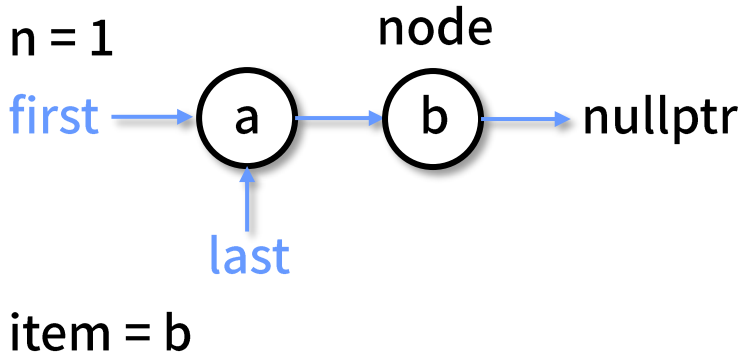
Queue (using Linked-list)

Insert (6)

Insert

```
def enqueue(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.first = node
    else:
        self.last.next_node = node
    self.last = node
    self.n += 1
    return
```

Visualization



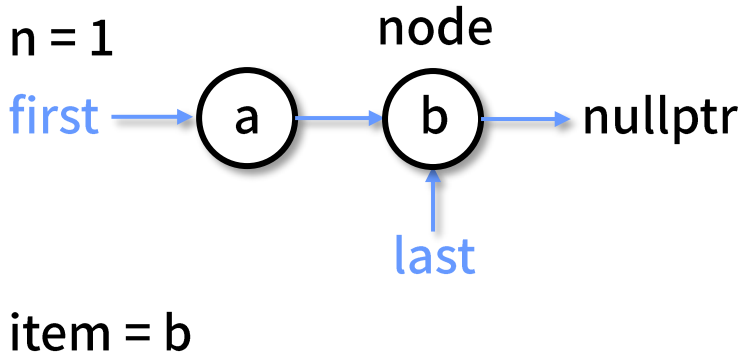
Queue (using Linked-list)

Insert (7)

Insert

```
def enqueue(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.first = node
    else:
        self.last.next_node = node
    self.last = node
    self.n += 1
    return
```

Visualization



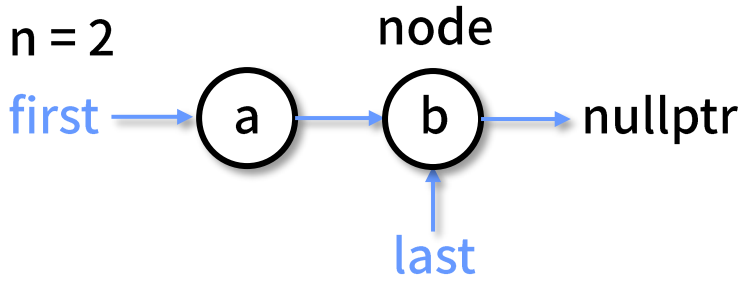
Queue (using Linked-list)

Insert (8)

Insert

```
def enqueue(self, item: Any) -> None:
    node = Node(item)
    if self.is_empty():
        self.first = node
    else:
        self.last.next_node = node
    self.last = node
    self.n += 1
    return
```

Visualization



Queue (using Linked-list)

Remove (1)

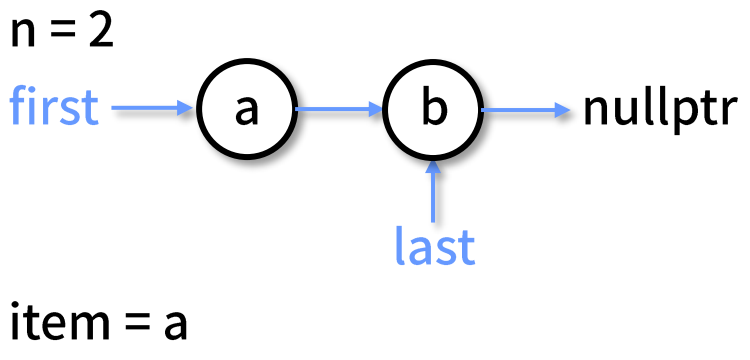
Remove

```
def dequeue(self) -> Any:
    item = self.first.item
    self.first = self.first.next_node
    self.n -= 1
    if self.n == 0:
        self.last = None
    return item
```

Command

```
queue.dequeue()
```

Visualization



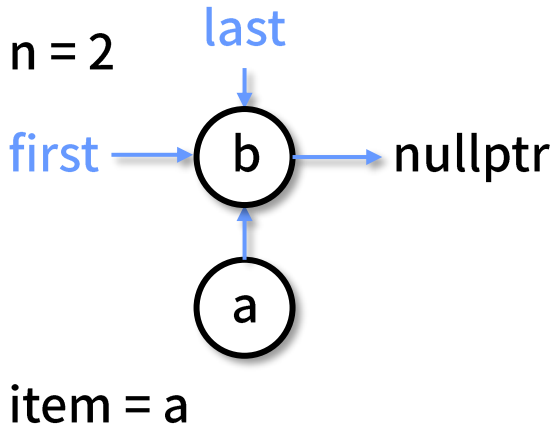
Queue (using Linked-list)

Remove (2)

Remove

```
def dequeue(self) -> Any:
    item = self.first.item
    self.first = self.first.next_node
    self.n -= 1
    if self.n == 0:
        self.last = None
    return item
```

Visualization



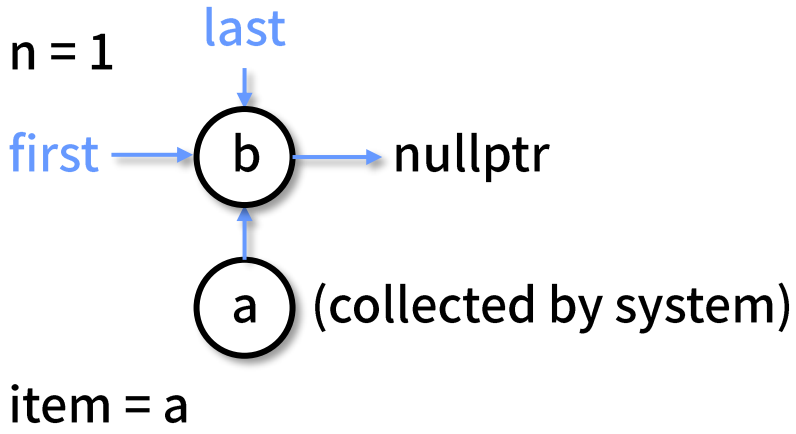
Queue (using Linked-list)

Remove (3)

Remove

```
def dequeue(self) -> Any:
    item = self.first.item
    self.first = self.first.next_node
    self.n -= 1
    if self.n == 0:
        self.last = None
    return item
```

Visualization



Queue (using Linked-list)

Remove (4)

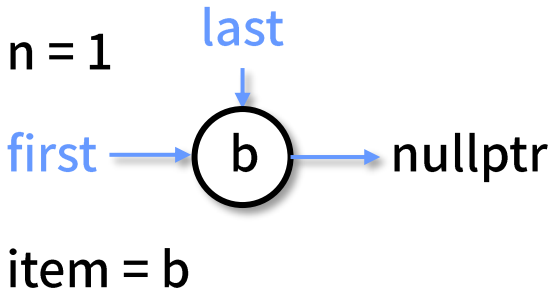
Remove

```
def dequeue(self) -> Any:
    item = self.first.item
    self.first = self.first.next_node
    self.n -= 1
    if self.n == 0:
        self.last = None
    return item
```

Command

```
queue.dequeue()
```

Visualization



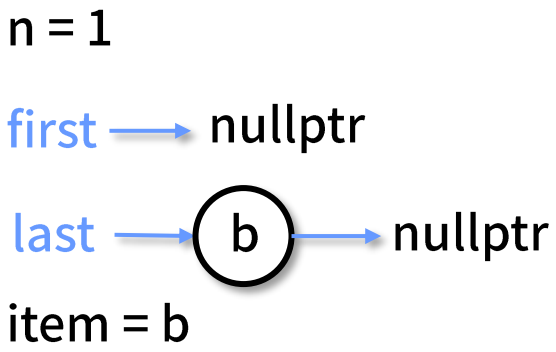
Queue (using Linked-list)

Remove (5)

Remove

```
def dequeue(self) -> Any:
    item = self.first.item
    self.first = self.first.next_node
    self.n -= 1
    if self.n == 0:
        self.last = None
    return item
```

Visualization



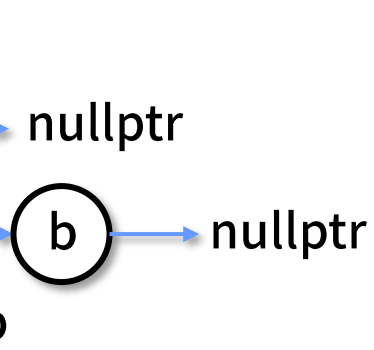
Queue (using Linked-list)

Remove (6)

Remove

```
def dequeue(self) -> Any:
    item = self.first.item
    self.first = self.first.next_node
    self.n -= 1
    if self.n == 0:
        self.last = None
    return item
```

Visualization



Queue (using Linked-list)

Remove (7)

Remove

```
def dequeue(self) -> Any:
    item = self.first.item
    self.first = self.first.next_node
    self.n -= 1
    if self.n == 0:
        self.last = None
    return item
```

Visualization

n = 0

first ⟶ nullptr

last ⟶ nullptr

item = b

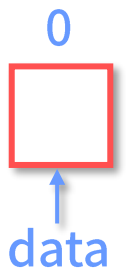
Queue (using Array)

Initialization

```
class Queue(Iterable):
  def __init__(self):
    self.n: int = 0
    self.first: int = 0
    self.last: int = -1
    self.data: Array = Array(1)

  def __len__(self) -> int: ···

  def __str__(self) -> str: ···

  def __iter__(self) -> Iterator[Queue]: ···

  def is_empty(self) -> bool: ···                check if the Queue is empty

  def peek(self) -> Any: ···                     check first inserted item

  def dequeue(self) -> Any: ···   remove and return the first inserted item

  def enqueue(self, item: Any) -> None: ···                  insert item
```

Visualization

n = 0 first = 0 last = -1



Queue (using Array)

Insert (1)

Insert

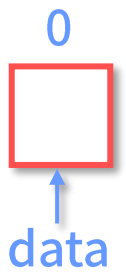
```
def enqueue(self, item: Any) -> None:
  if self.n == len(self.data):
    self.data.resize(2 * self.n)
  self.last += 1
  self.data[self.last] = item
  self.n += 1
  return
```

Command

```
queue = Queue()
queue.enqueue('a')
```

Visualization (w/o Resize)

n = 0 first = 0 last = 0



item = a

Queue (using Array)

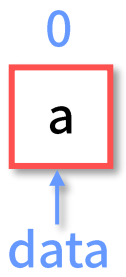
Insert (2)

Insert

```
def enqueue(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.last += 1
    self.data[self.last] = item
    self.n += 1
    return
```

Visualization (w/o Resize)

n = 0 first = 0 last = 0



item = a

Queue (using Array)

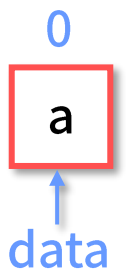
Insert (3)

Insert

```
def enqueue(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.last += 1
    self.data[self.last] = item
    self.n += 1
    return
```

Visualization (w/o Resize)

n = 1 first = 0 last = 0



item = a

Queue (using Array)

Insert (4)

Insert

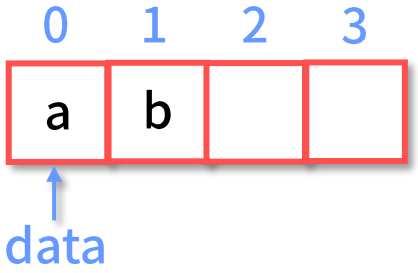
```
def enqueue(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.last += 1
    self.data[self.last] = item
    self.n += 1
    return
```

Command

```
queue.enqueue('b')
queue.enqueue('c')
```

Visualization (w/ Resize)

n = 2 first = 0 last = 1



item = c

Queue (using Array)

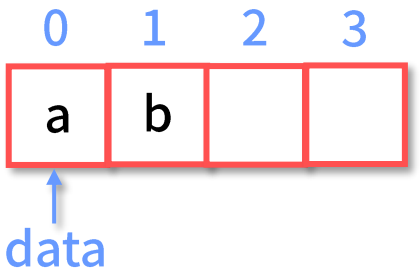
Insert (5)

Insert

```
def enqueue(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.last += 1
    self.data[self.last] = item
    self.n += 1
    return
```

Visualization (w/ Resize)

n = 2 first = 0 last = 2



item = c

Queue (using Array)

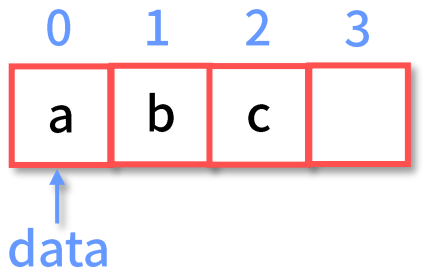
Insert (6)

Insert

```
def enqueue(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.last += 1
    self.data[self.last] = item
    self.n += 1
    return
```

Visualization (w/ Resize)

n = 2 first = 0 last = 2



item = c

Queue (using Array)

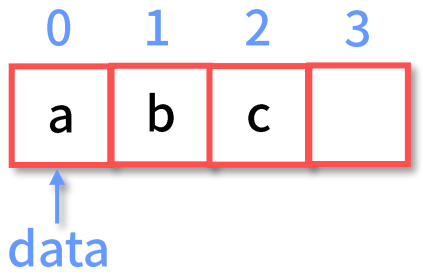
Insert (7)

Insert

```
def enqueue(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.last += 1
    self.data[self.last] = item
    self.n += 1
    return
```

Visualization (w/ Resize)

n = 3 first = 0 last = 2



item = c

Queue (using Array)

Insert (8)

Insert

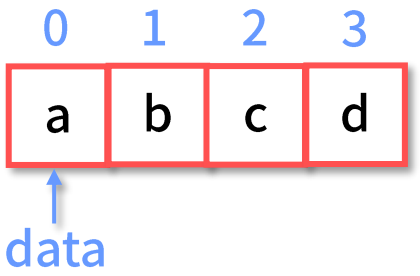
```
def enqueue(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.last += 1
    self.data[self.last] = item
    self.n += 1
    return
```

Command

```
queue.enqueue('d')
```

Visualization (w/ Resize)

n = 4 first = 0 last = 3



Queue (using Array)

Insert (9)

Insert

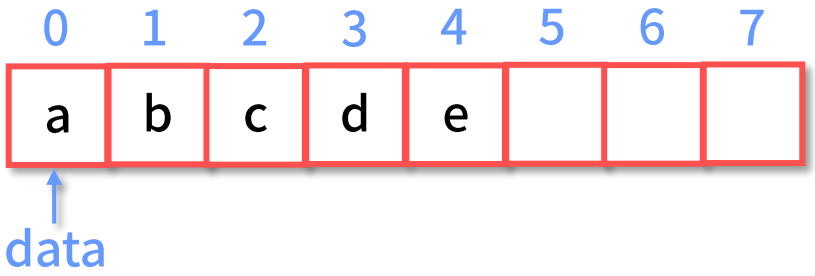
```
def enqueue(self, item: Any) -> None:
    if self.n == len(self.data):
        self.data.resize(2 * self.n)
    self.last += 1
    self.data[self.last] = item
    self.n += 1
    return
```

Command

```
queue.enqueue('e')
```

Visualization (w/ Resize)

n = 5 first = 0 last = 4



Queue (using Array)

Remove (1)

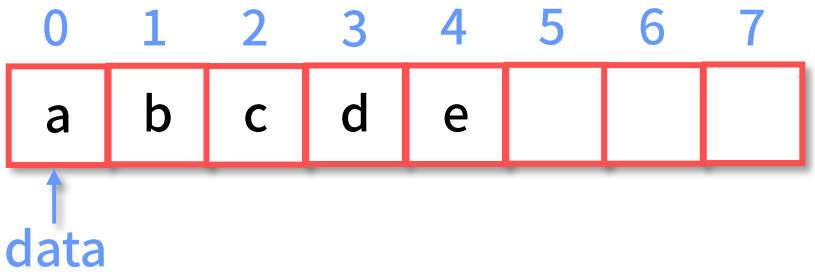
Remove

```
def dequeue(self) -> Any:
    item = self.data[self.first]
    self.data[self.first] = None
    self.first += 1
    self.n -= 1
    if self.n > 0 and self.n == len(self.data) // 4:
        self.data.resize(len(self.data) // 2, self.first)
        self.first = 0
        self.last = self.n - 1
    if self.n == 0:
        self.first = 0
        self.last = -1
    return item
```

```
queue.dequeue()
```

Visualization (w/o Resize)

n = 5 first = 0 last = 4



item = a

Queue (using Array)

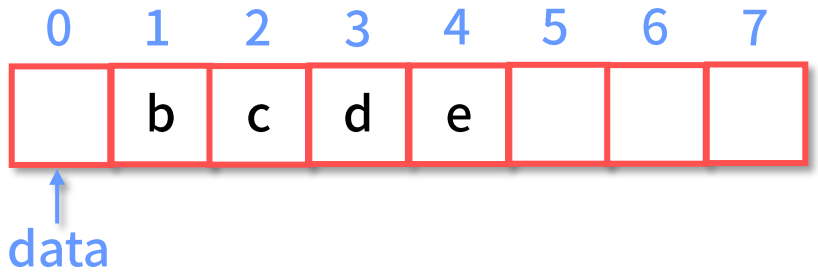
Remove (2)

Remove

```
def dequeue(self) -> Any:
  item = self.data[self.first]
  self.data[self.first] = None
  self.first += 1
  self.n -= 1
  if self.n > 0 and self.n == len(self.data) // 4:
    self.data.resize(len(self.data) // 2, self.first)
    self.first = 0
    self.last = self.n - 1
  if self.n == 0:
    self.first = 0
    self.last = -1
  return item
```

Visualization (w/o Resize)

n = 5 first = 0 last = 4



item = a

Queue (using Array)

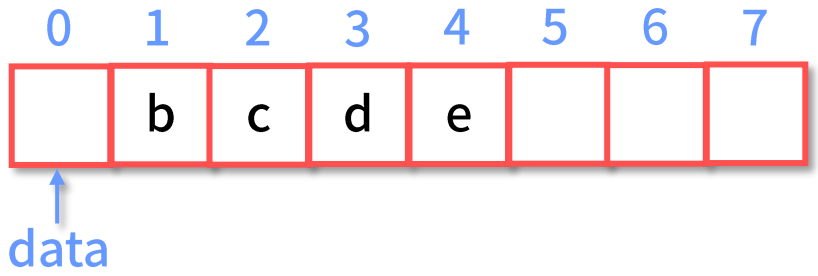
Remove (3)

Remove

```
def dequeue(self) -> Any:
    item = self.data[self.first]
    self.data[self.first] = None
    self.first += 1
    self.n -= 1
    if self.n > 0 and self.n == len(self.data) // 4:
        self.data.resize(len(self.data) // 2, self.first)
        self.first = 0
        self.last = self.n - 1
    if self.n == 0:
        self.first = 0
        self.last = -1
    return item
```

Visualization (w/o Resize)

n = 5 first = 1 last = 4



item = a

Queue (using Array)

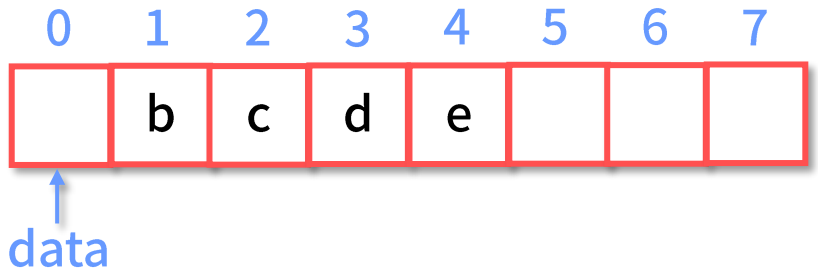
Remove (4)

Remove

```
def dequeue(self) -> Any:
    item = self.data[self.first]
    self.data[self.first] = None
    self.first += 1
    self.n -= 1
    if self.n > 0 and self.n == len(self.data) // 4:
        self.data.resize(len(self.data) // 2, self.first)
        self.first = 0
        self.last = self.n - 1
    if self.n == 0:
        self.first = 0
        self.last = -1
    return item
```

Visualization (w/o Resize)

n = 4 first = 1 last = 4



item = a

Queue (using Array)

Remove(5)

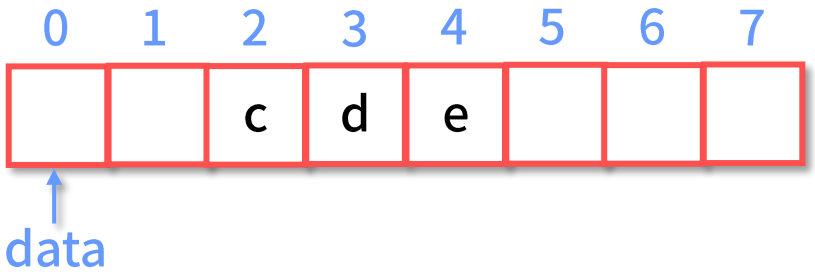
Remove

```
def dequeue(self) -> Any:
  item = self.data[self.first]
  self.data[self.first] = None
  self.first += 1
  self.n -= 1
  if self.n > 0 and self.n == len(self.data) // 4:
    self.data.resize(len(self.data) // 2, self.first)
    self.first = 0
    self.last = self.n - 1
  if self.n == 0:
    self.first = 0
    self.last = -1
  return item
```

```
queue.dequeue()
```

Visualization (w/o Resize)

n = 3 first = 2 last = 4



Queue (using Array)

Remove (6)

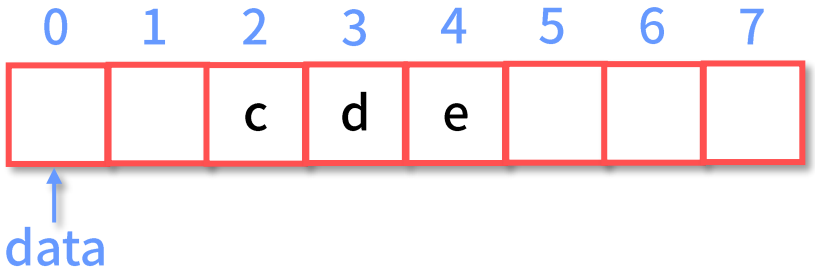
Remove

```
def dequeue(self) -> Any:
    item = self.data[self.first]
    self.data[self.first] = None
    self.first += 1
    self.n -= 1
    if self.n > 0 and self.n == len(self.data) // 4:
        self.data.resize(len(self.data) // 2, self.first)
        self.first = 0
        self.last = self.n - 1
    if self.n == 0:
        self.first = 0
        self.last = -1
    return item
```

```
queue.dequeue()
```

Visualization (w/ Resize)

n = 3 first = 2 last = 4



item = c

Queue (using Array)

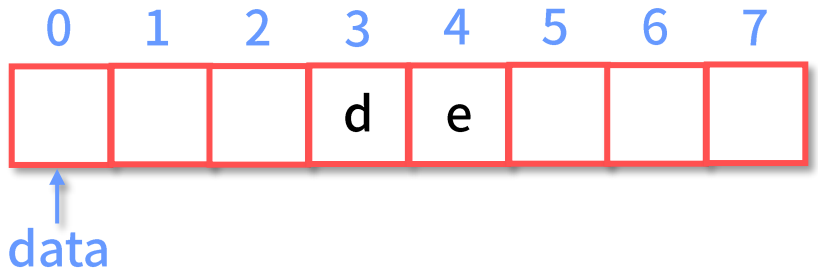
Remove (7)

Remove

```
def dequeue(self) -> Any:
    item = self.data[self.first]
    self.data[self.first] = None
    self.first += 1
    self.n -= 1
    if self.n > 0 and self.n == len(self.data) // 4:
        self.data.resize(len(self.data) // 2, self.first)
        self.first = 0
        self.last = self.n - 1
    if self.n == 0:
        self.first = 0
        self.last = -1
    return item
```

Visualization (w/ Resize)

n = 3 first = 2 last = 4



item = c

Queue (using Array)

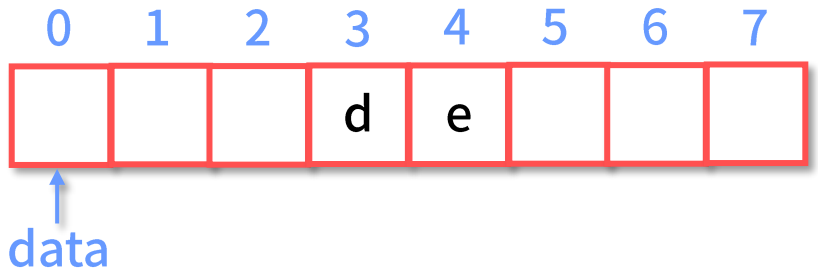
Remove (8)

Remove

```
def dequeue(self) -> Any:
  item = self.data[self.first]
  self.data[self.first] = None
  self.first += 1
  self.n -= 1
  if self.n > 0 and self.n == len(self.data) // 4:
    self.data.resize(len(self.data) // 2, self.first)
    self.first = 0
    self.last = self.n - 1
  if self.n == 0:
    self.first = 0
    self.last = -1
  return item
```

Visualization (w/ Resize)

n = 3 first = 3 last = 4



item = c

Queue (using Array)

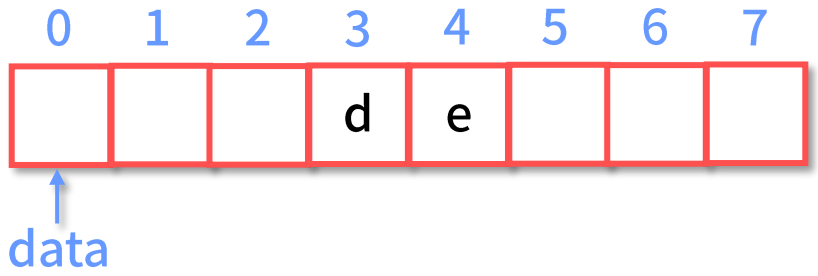
Remove (9)

Remove

```
def dequeue(self) -> Any:
    item = self.data[self.first]
    self.data[self.first] = None
    self.first += 1
    self.n -= 1
    if self.n > 0 and self.n == len(self.data) // 4:
        self.data.resize(len(self.data) // 2, self.first)
        self.first = 0
        self.last = self.n - 1
    if self.n == 0:
        self.first = 0
        self.last = -1
    return item
```

Visualization (w/ Resize)

n = 2 first = 3 last = 4



item = c

Queue (using Array)

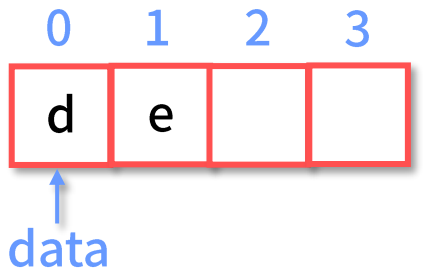
Remove (10)

Remove

```
def dequeue(self) -> Any:
    item = self.data[self.first]
    self.data[self.first] = None
    self.first += 1
    self.n -= 1
    if self.n > 0 and self.n == len(self.data) // 4:
        self.data.resize(len(self.data) // 2, self.first)
        self.first = 0
        self.last = self.n - 1
    if self.n == 0:
        self.first = 0
        self.last = -1
    return item
```

Visualization (w/ Resize)

n = 2 first = 3 last = 4



item = c

Queue (using Array)

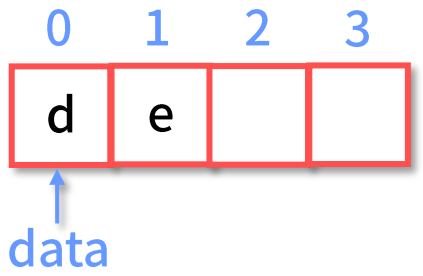
Remove (11)

Remove

```
def dequeue(self) -> Any:
  item = self.data[self.first]
  self.data[self.first] = None
  self.first += 1
  self.n -= 1
  if self.n > 0 and self.n == len(self.data) // 4:
    self.data.resize(len(self.data) // 2, self.first)
    self.first = 0
    self.last = self.n - 1
  if self.n == 0:
    self.first = 0
    self.last = -1
  return item
```

Visualization (w/ Resize)

n = 2 first = 0 last = 4



item = c

Queue (using Array)

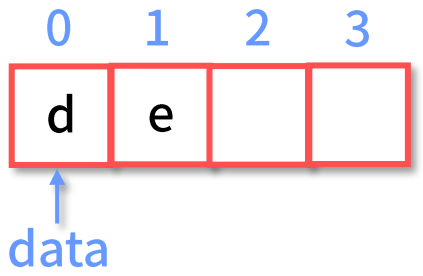
Remove (12)

Remove

```
def dequeue(self) -> Any:
  item = self.data[self.first]
  self.data[self.first] = None
  self.first += 1
  self.n -= 1
  if self.n > 0 and self.n == len(self.data) // 4:
    self.data.resize(len(self.data) // 2, self.first)
    self.first = 0
    self.last = self.n - 1
  if self.n == 0:
    self.first = 0
    self.last = -1
  return item
```

Visualization (w/ Resize)

n = 2 first = 0 last = 1



item = c

Queue (Practice)

1. Time Needed to Buy Tickets (Leetcode Problem 2073)
2. Number of Recent Calls (Leetcode Problem 933)

Queue

Summary

Time Complexity

	Linked-list	Array
Initialization	O(1)	O(1)
Push	O(1)	O(1)*
Pop	O(1)	O(1)*
Peek	O(1)	O(1)

*amortized time complexity