Priority Queues: Introduction

Alexander S. Kulikov

Steklov Institute of Mathematics at St. Petersburg
Russian Academy of Sciences

Data Structures
Data Structures and Algorithms

Outline

1 Overview

2 Naive Implementations

Learning objectives

You will be able to:

- Implement a priority queue
- Explain what is going on inside built-in implementations:
  - C++: `priority_queue`
  - Java: `PriorityQueue`
  - Python: `heapq`

Queue

A queue is an abstract data type supporting the following main operations:

- `PushBack(`*e*`)` adds an element to the back of the queue;
- `PopFront()` extracts an element from the front of the queue.

Priority Queue (Informally)

A priority queue is a generalization of a queue where each element is assigned a priority and elements come out in order by priority.

Priority Queues: Typical Use Case

Scheduling jobs

- Want to process jobs one by one in order of decreasing priority. While the current job is processed, new jobs may arrive.

Priority Queues: Typical Use Case

Scheduling jobs

- Want to process jobs one by one in order of decreasing priority. While the current job is processed, new jobs may arrive.
- To add a job to the set of scheduled jobs, call `Insert`(*job*).

Priority Queues: Typical Use Case

Scheduling jobs

- Want to process jobs one by one in order of decreasing priority. While the current job is processed, new jobs may arrive.
- To add a job to the set of scheduled jobs, call `Insert`(*job*).
- To process a job with the highest priority, get it by calling `ExtractMax`().

Priority Queue (Formally)

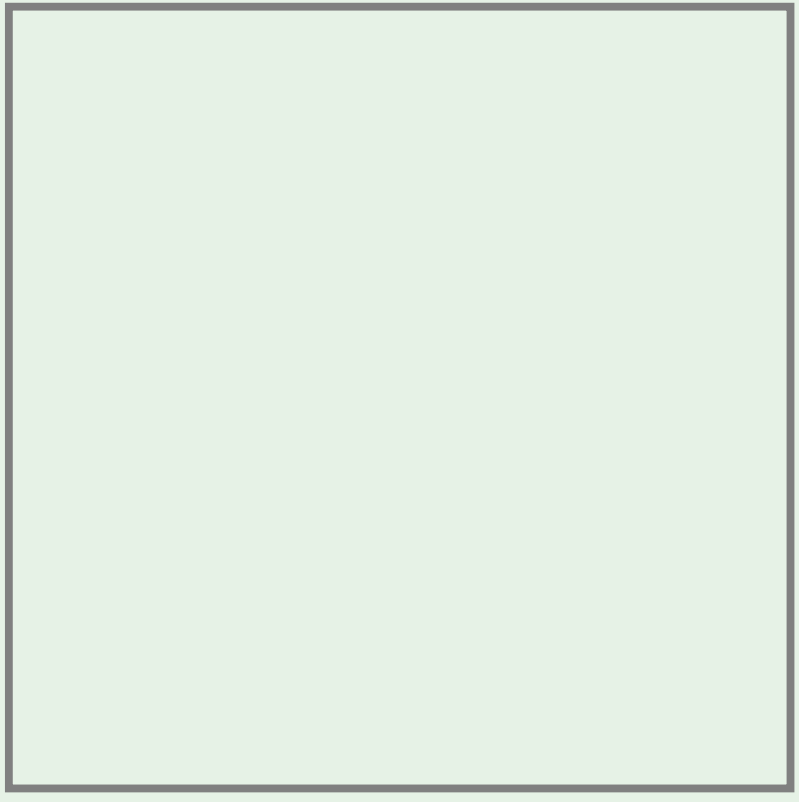
Definition

Priority queue is an abstract data type supporting the following main operations:

- `Insert`(p) adds a new element with priority p
- `ExtractMax`() extracts an element with maximum priority

Example

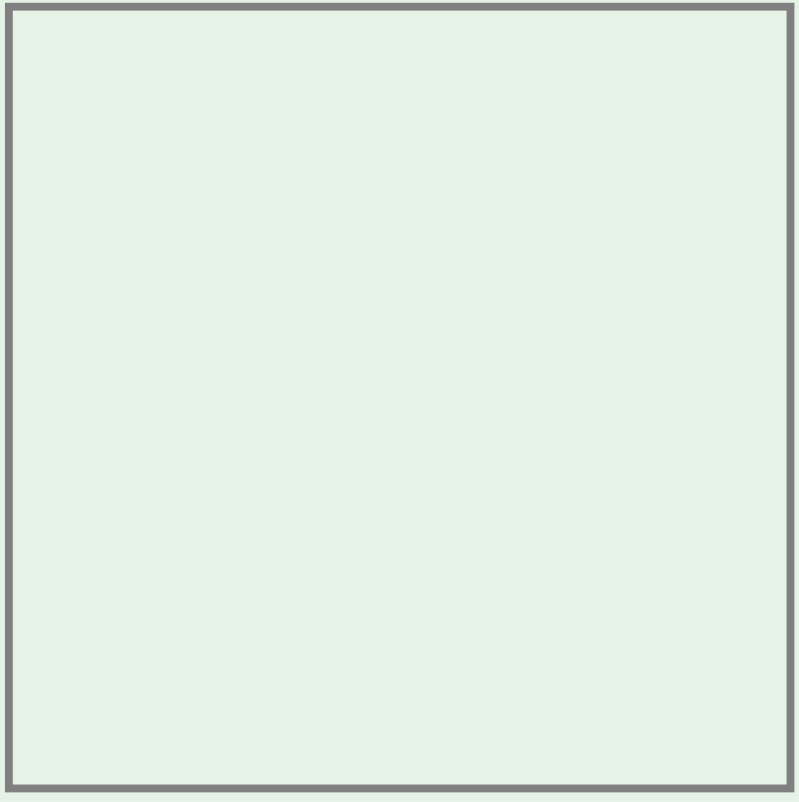
Contents:



Queries:

Example

Contents:

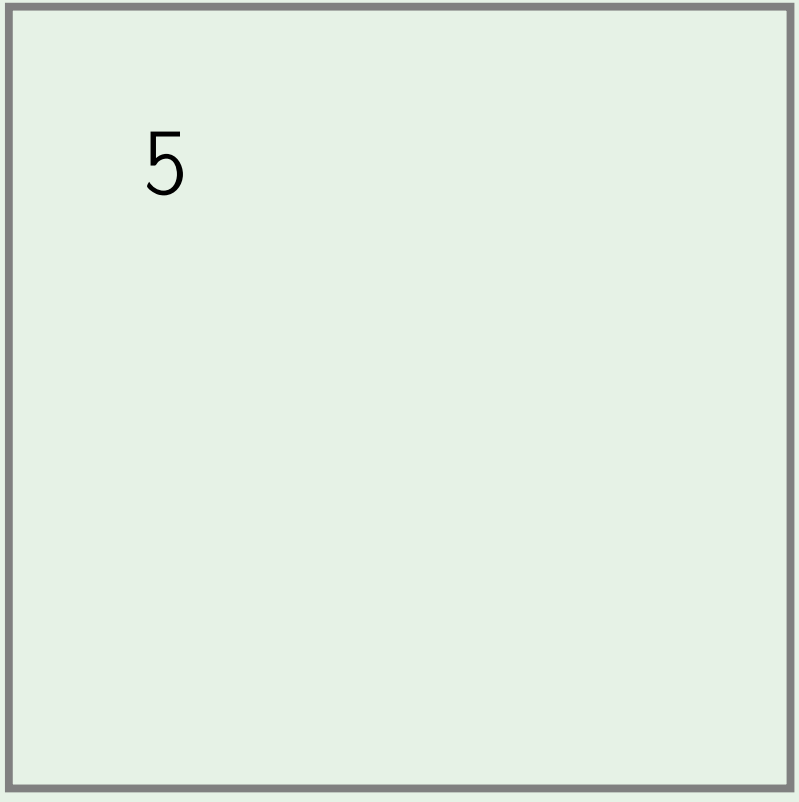


Queries:

`Insert(5)`

Example

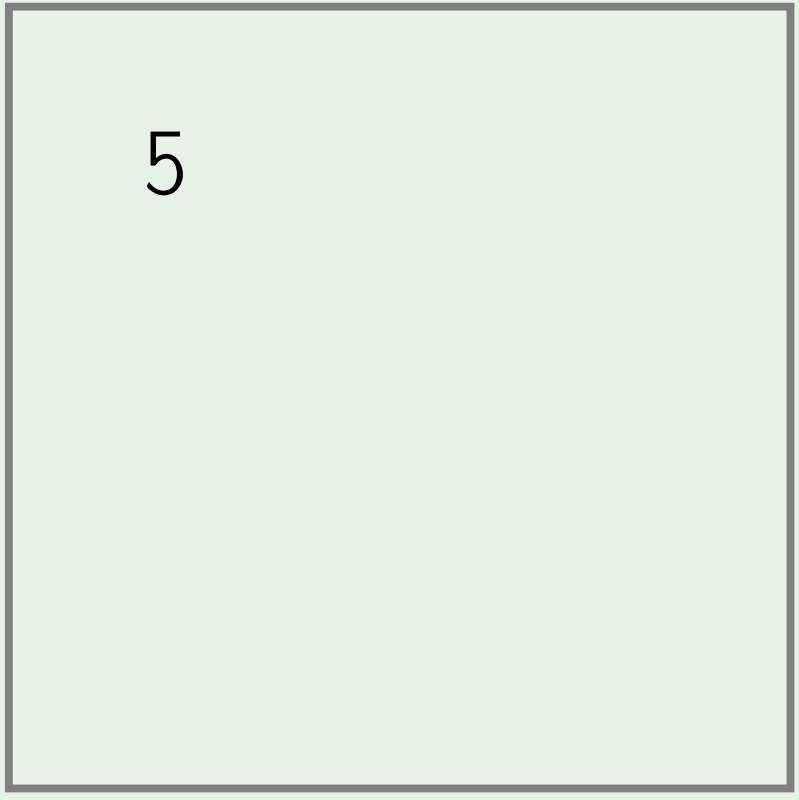
Contents:



Queries:

Example

Contents:

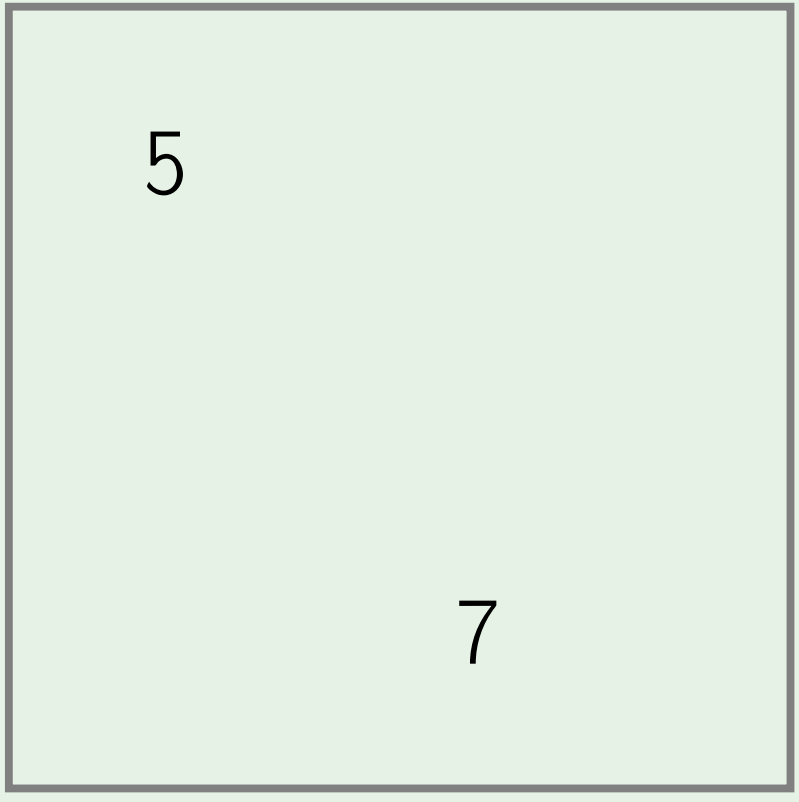


Queries:

`Insert(7)`

Example

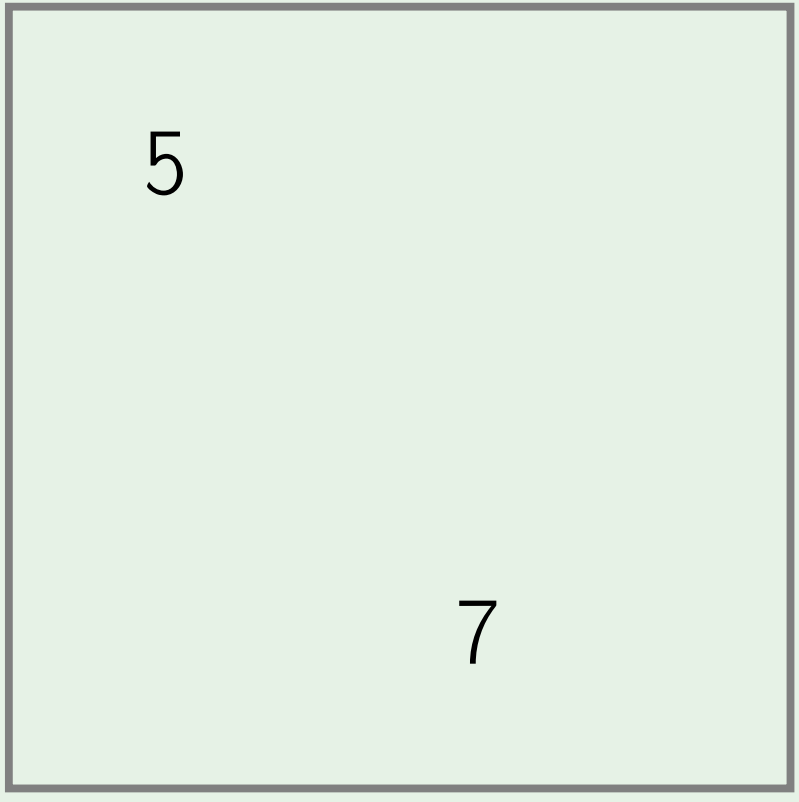
Contents:



Queries:

Example

Contents:

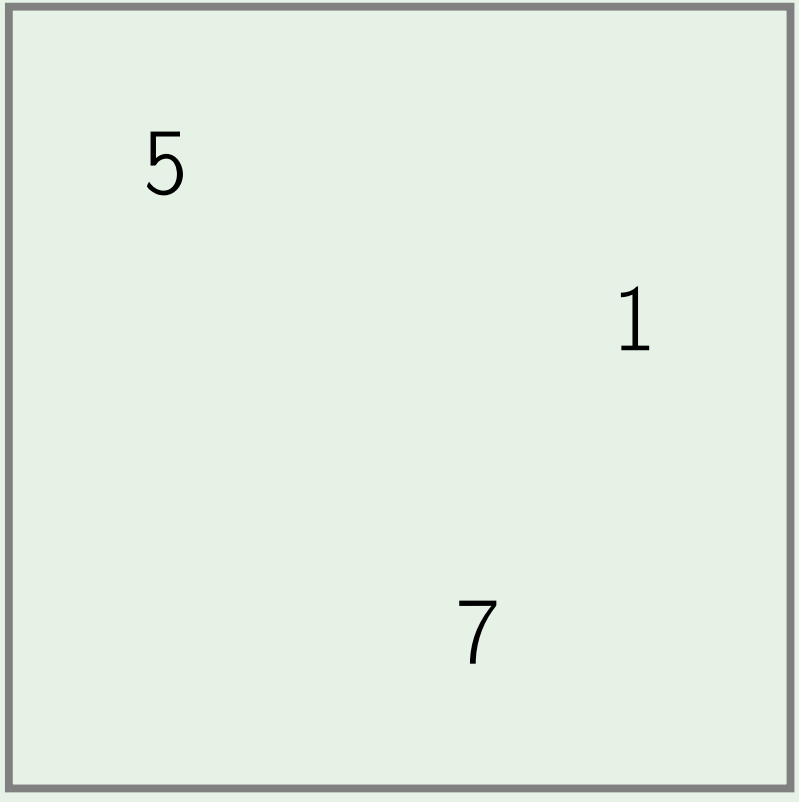


Queries:

`Insert(1)`

Example

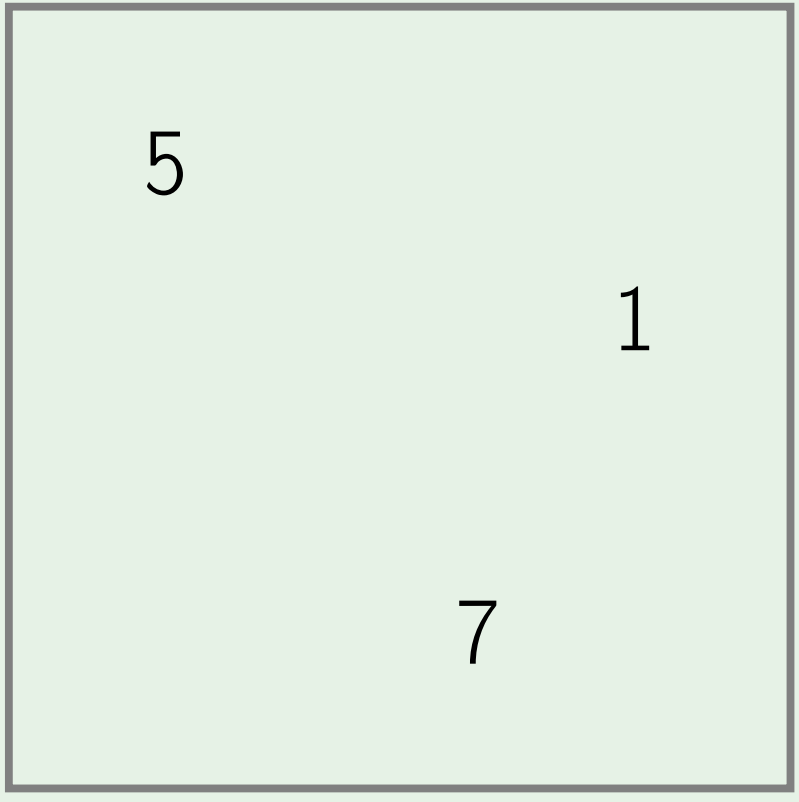
Contents:



Queries:

Example

Contents:

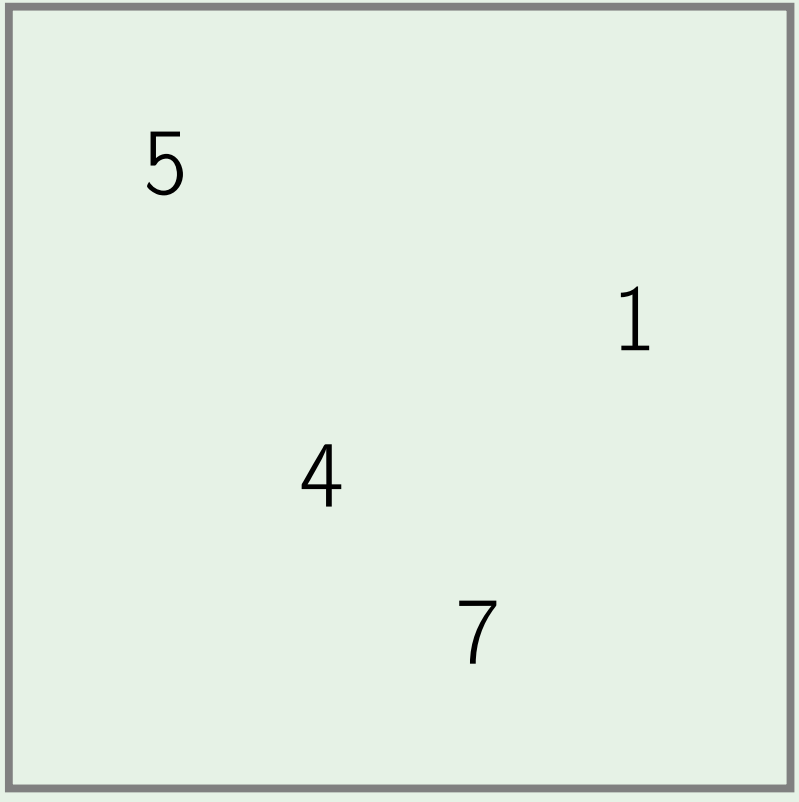


Queries:

`Insert(4)`

Example

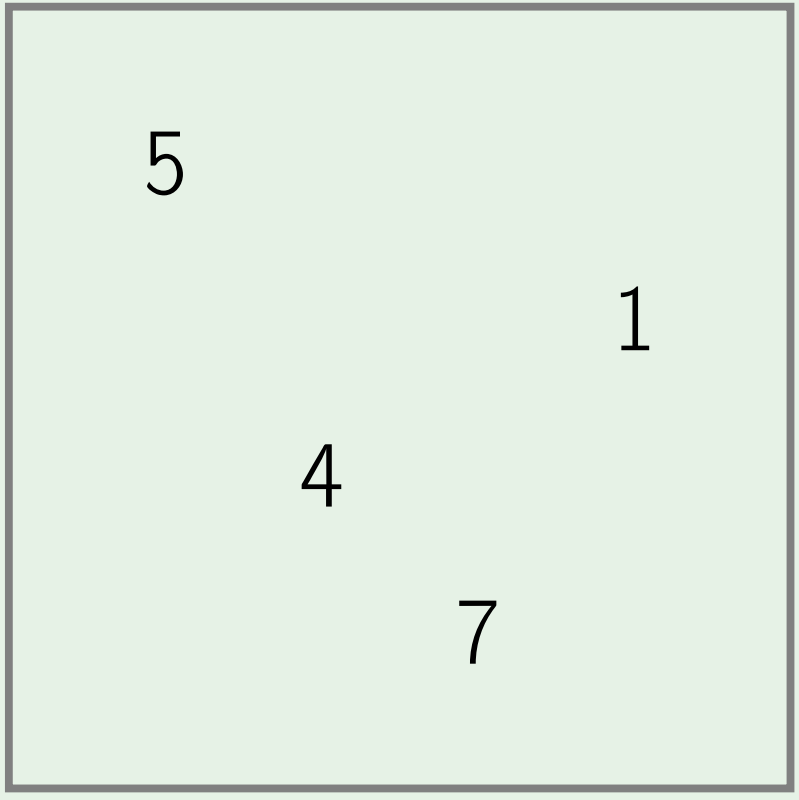
Contents:



Queries:

Example

Contents:

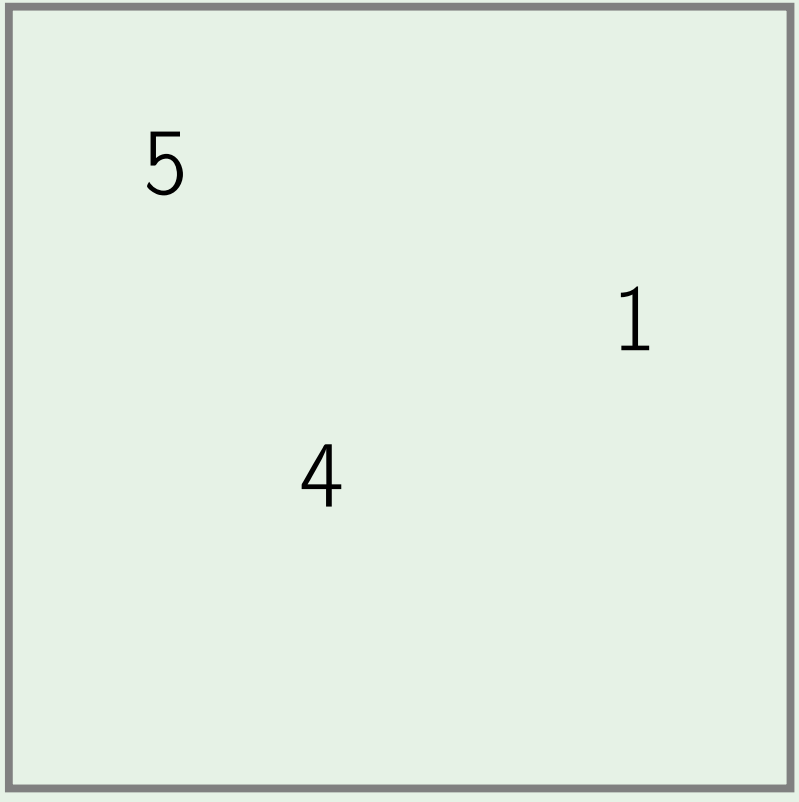


Queries:

`ExtractMax`() $\rightarrow$ 7

Example

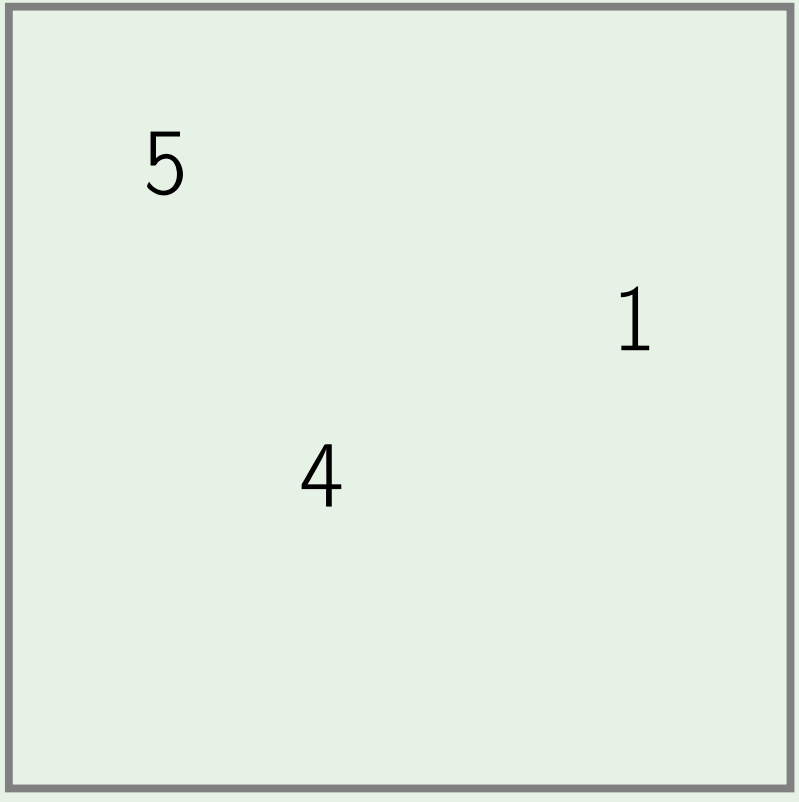
Contents:



Queries:

Example

Contents:

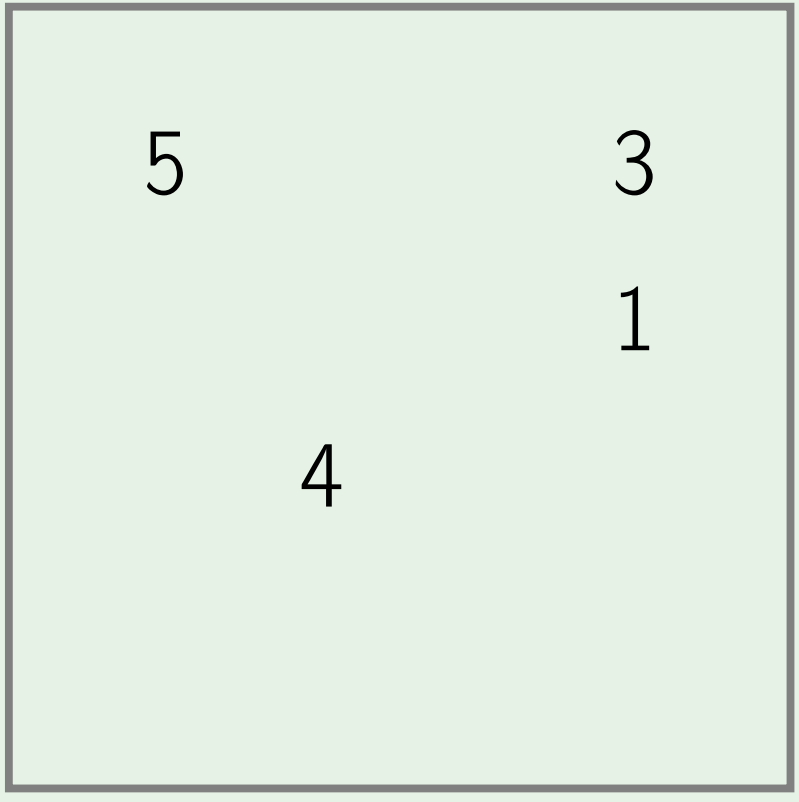


Queries:

`Insert(3)`

Example

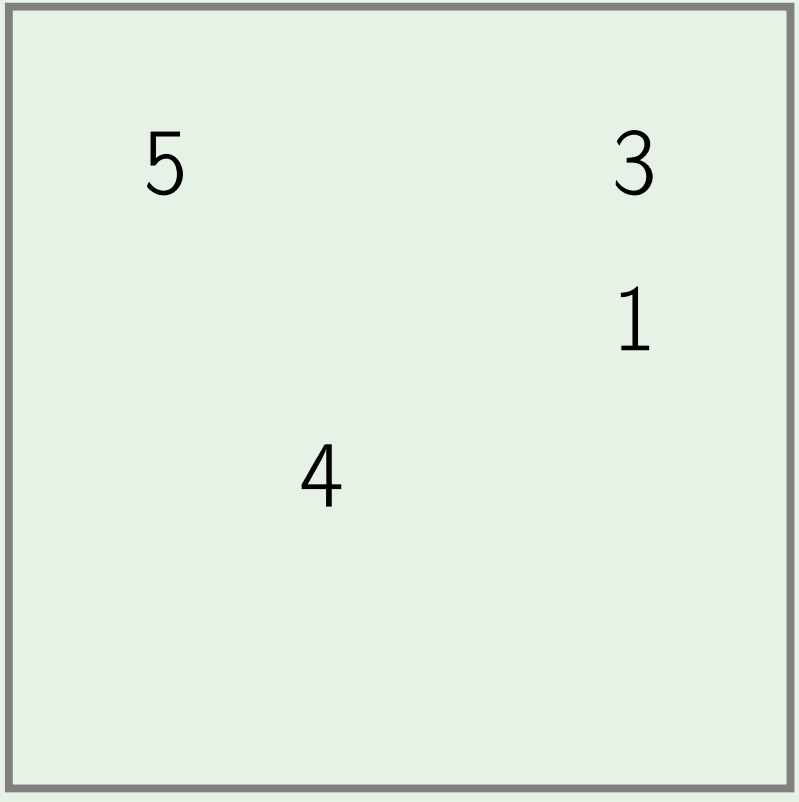
Contents:



Queries:

Example

Contents:

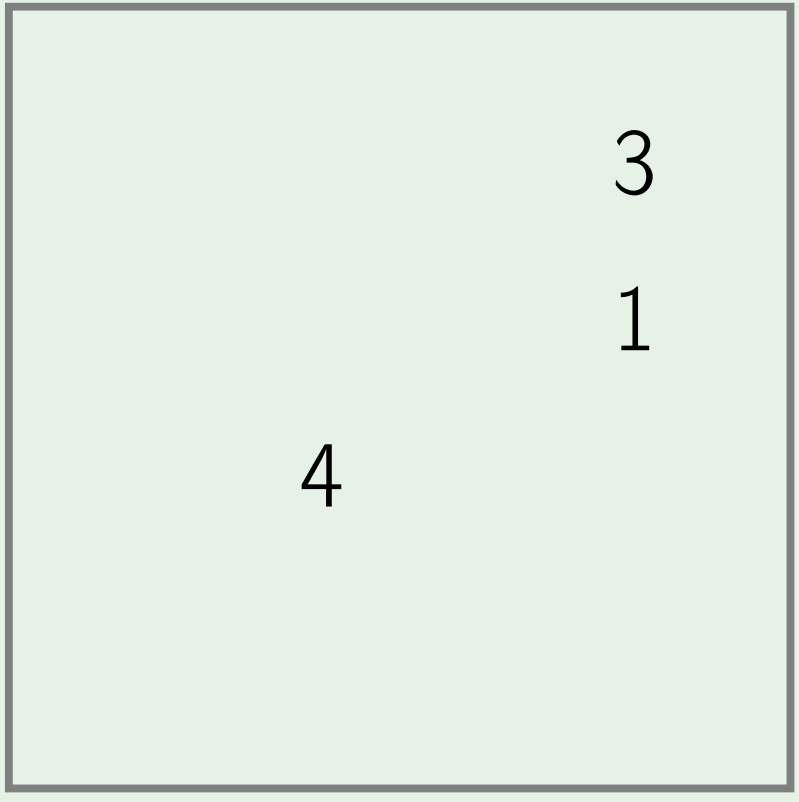


Queries:

`ExtractMax()` $\rightarrow 5$

Example

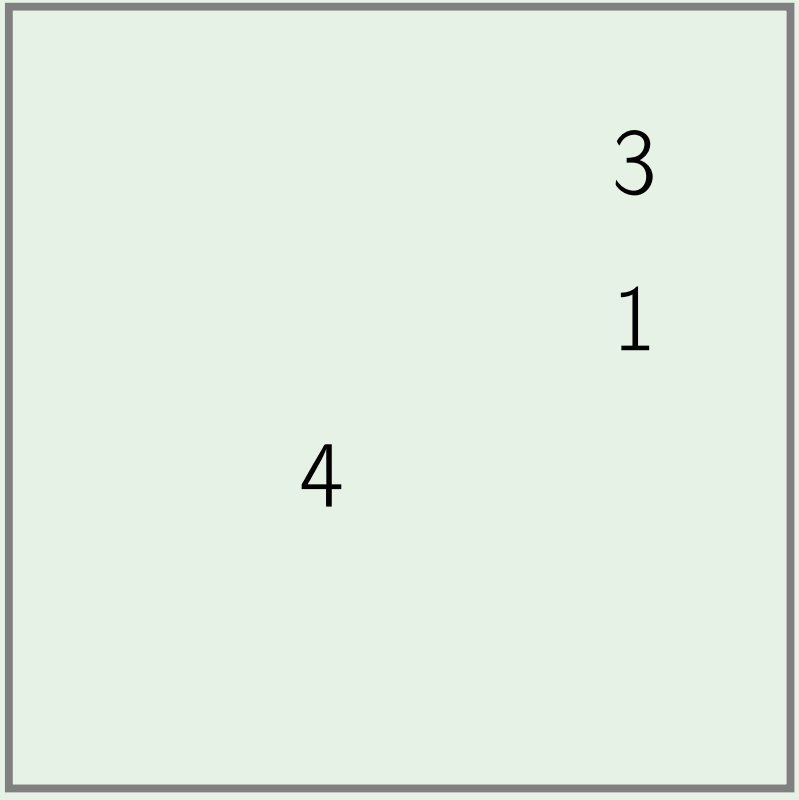
Contents:



Queries:

Example

Contents:

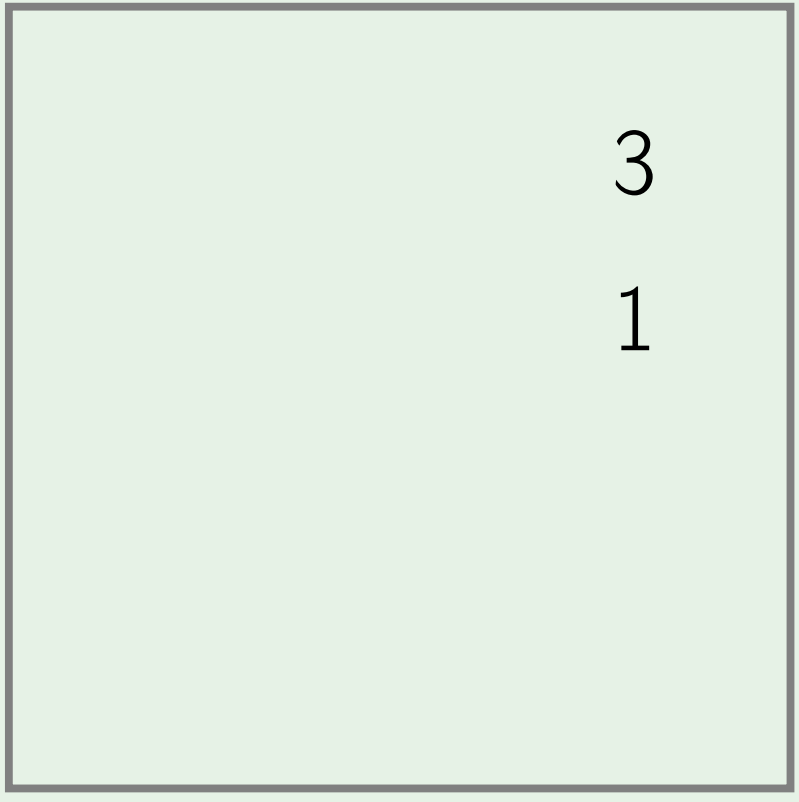


Queries:

$\texttt{ExtractMax}() \rightarrow 4$

Example

Contents:



Queries:

Additional Operations

- `Remove`(it) removes an element pointed by an iterator it
- `GetMax`() returns an element with maximum priority (without changing the set of elements)
- `ChangePriority`(it, p) changes the priority of an element pointed by it to p

Algorithms that Use Priority Queues

- Dijkstra's algorithm: finding a shortest path in a graph

Algorithms that Use Priority Queues

- Dijkstra's algorithm: finding a shortest path in a graph
- Prim's algorithm: constructing a minimum spanning tree of a graph

Algorithms that Use Priority Queues

- Dijkstra's algorithm: finding a shortest path in a graph
- Prim's algorithm: constructing a minimum spanning tree of a graph
- Huffman's algorithm: constructing an optimum prefix-free encoding of a string

Algorithms that Use Priority Queues

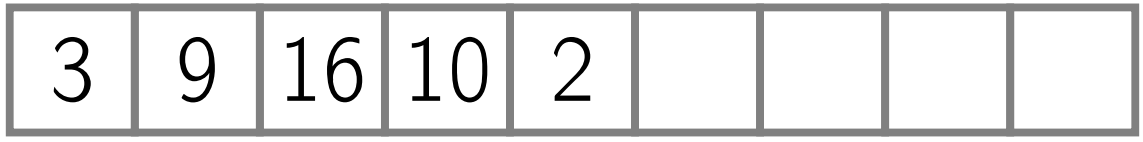
- Dijkstra's algorithm: finding a shortest path in a graph
- Prim's algorithm: constructing a minimum spanning tree of a graph
- Huffman's algorithm: constructing an optimum prefix-free encoding of a string
- Heap sort: sorting a given sequence

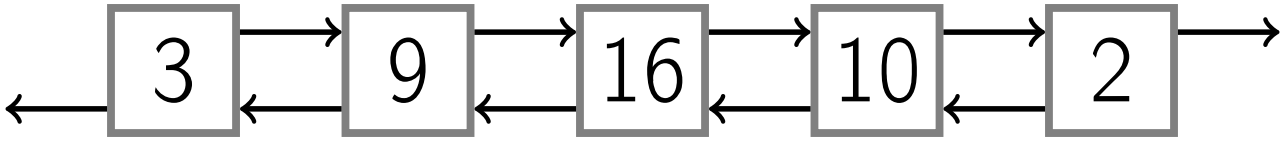
Outline

1 Overview

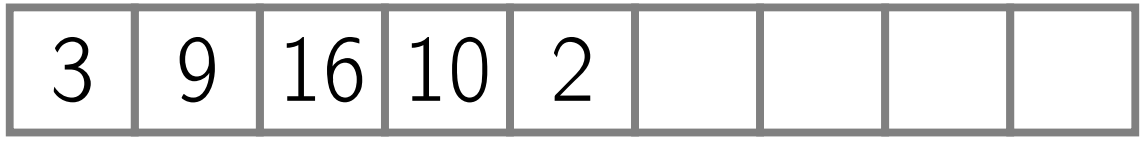
2 Naive Implementations

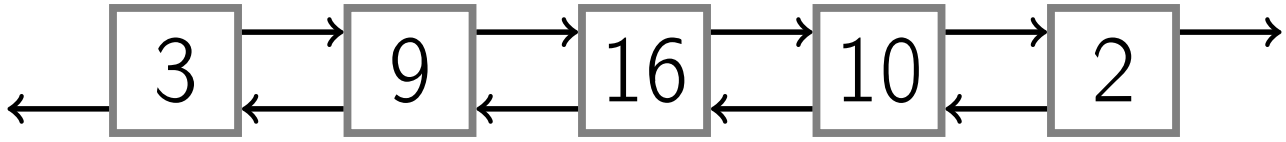
Unsorted Array/List





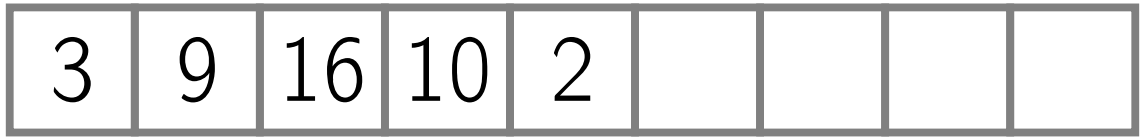
Unsorted Array/List

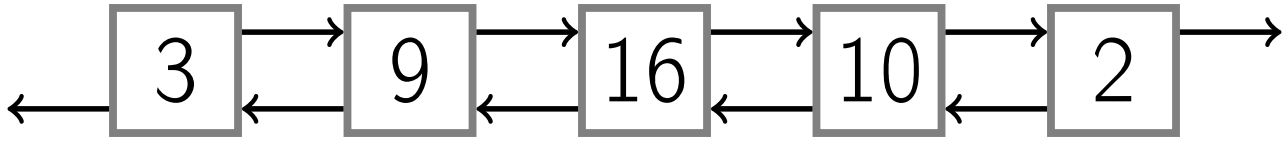




- `Insert(`e`)`
  - add e to the end
  - running time: $O(1)$

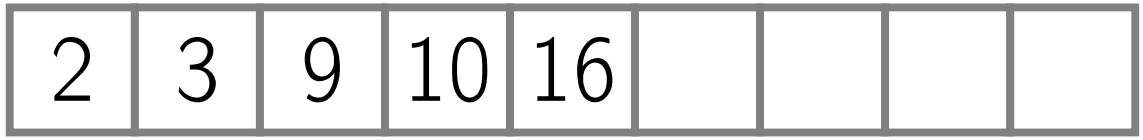
Unsorted Array/List



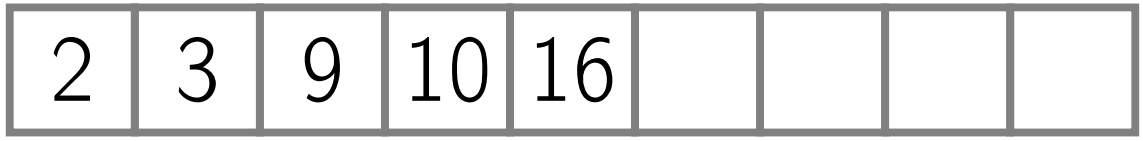


- `Insert(e)`
  - add e to the end
  - running time: $O(1)$
- `ExtractMax()`
  - scan the array/list
  - running time: $O(n)$

Sorted Array

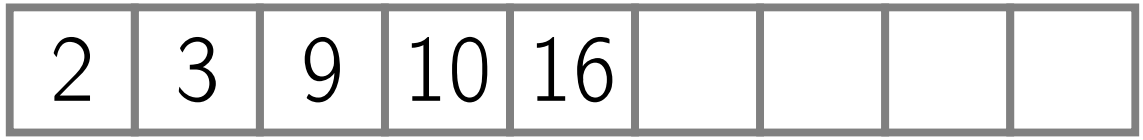


Sorted Array



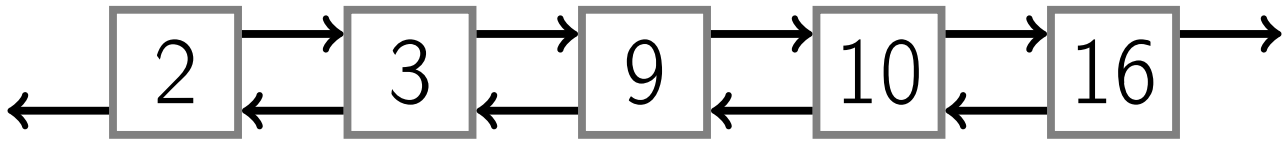
- `ExtractMax()`
  - extract the last element
  - running time: $O(1)$

Sorted Array

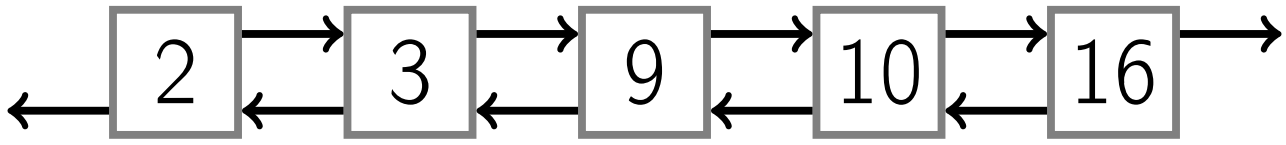


- `ExtractMax()`
  - extract the last element
  - running time: $O(1)$
- `Insert(`e`)`
  - find a position for e ($O(\log n)$ by using binary search), shift all elements to the right of it by 1 ($O(n)$), insert e ($O(1)$)
  - running time: $O(n)$

Sorted List

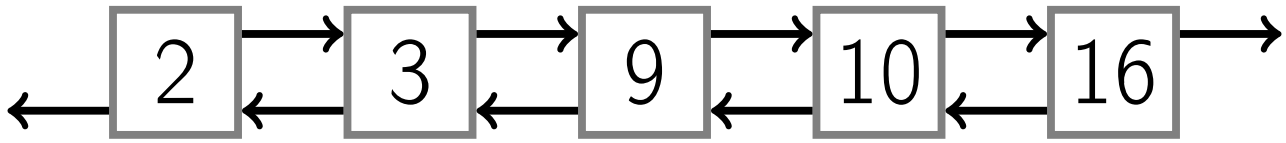


Sorted List



- `ExtractMax()`
  - extract the last element
  - running time: $O(1)$

Sorted List



- `ExtractMax()`
  - extract the last element
  - running time: $O(1)$
- `Insert(`e`)`
  - find a position for e ($O(n)$; note: cannot use binary search), insert e ($O(1)$)
  - running time: $O(n)$

Summary

	`Insert`	`ExtractMax`
Unsorted array/list	$O(1)$	$O(n)$
Sorted array/list	$O(n)$	$O(1)$

Summary

	`Insert`	`ExtractMax`
Unsorted array/list	$O(1)$	$O(n)$
Sorted array/list	$O(n)$	$O(1)$
Binary heap	$O(\log n)$	$O(\log n)$