Carleton University
Alina Shaikhet

COMP 2402

Abstract Data Types & Algorithms

Welcome to COMP 2402

Alina Shaikhet – Instructor

Email: alina.shaikhet@carleton.ca

Office Hours:
Tuesdays 13:30 – 15:30 in HP 5137

Ishtiaque Hossain – Instructor

Email:
ishtiaquehossain@cunet.carleton.ca

Office Hours: Thursdays 13:30 – 15:30

Teaching Assistants:

Abby Ibrahim
Aekus Trehan
Alvina Han
Ansh Arora
Gabe Martell
Grant Li
James Yap
Jansen Khoe
John Lu
Khang Tran
Laura Jin
Lauris Petlah
Michael Ge
Nathaniel Lays
Omar Sha
Robert Babaev
Salman Aljanabi
Sam Israelstam
Shawn Shi
Tom Mai
Ujan Sen

Course expectations

What will you learn and why?

(code that is efficient, reliable, fast, and elegant)

You will learn how to write **better code** leading to software that runs **faster** and consumes **less memory**.

You will be able to weigh the pros and cons of various solutions to the given problem and be able to make educated decisions as to which code is best for the given situation.

This course is fundamental to computer science. When you are applying for software developing (or engineering) positions, I guarantee you will be tested on problems that involve the use of data structures and algorithms.

Topics

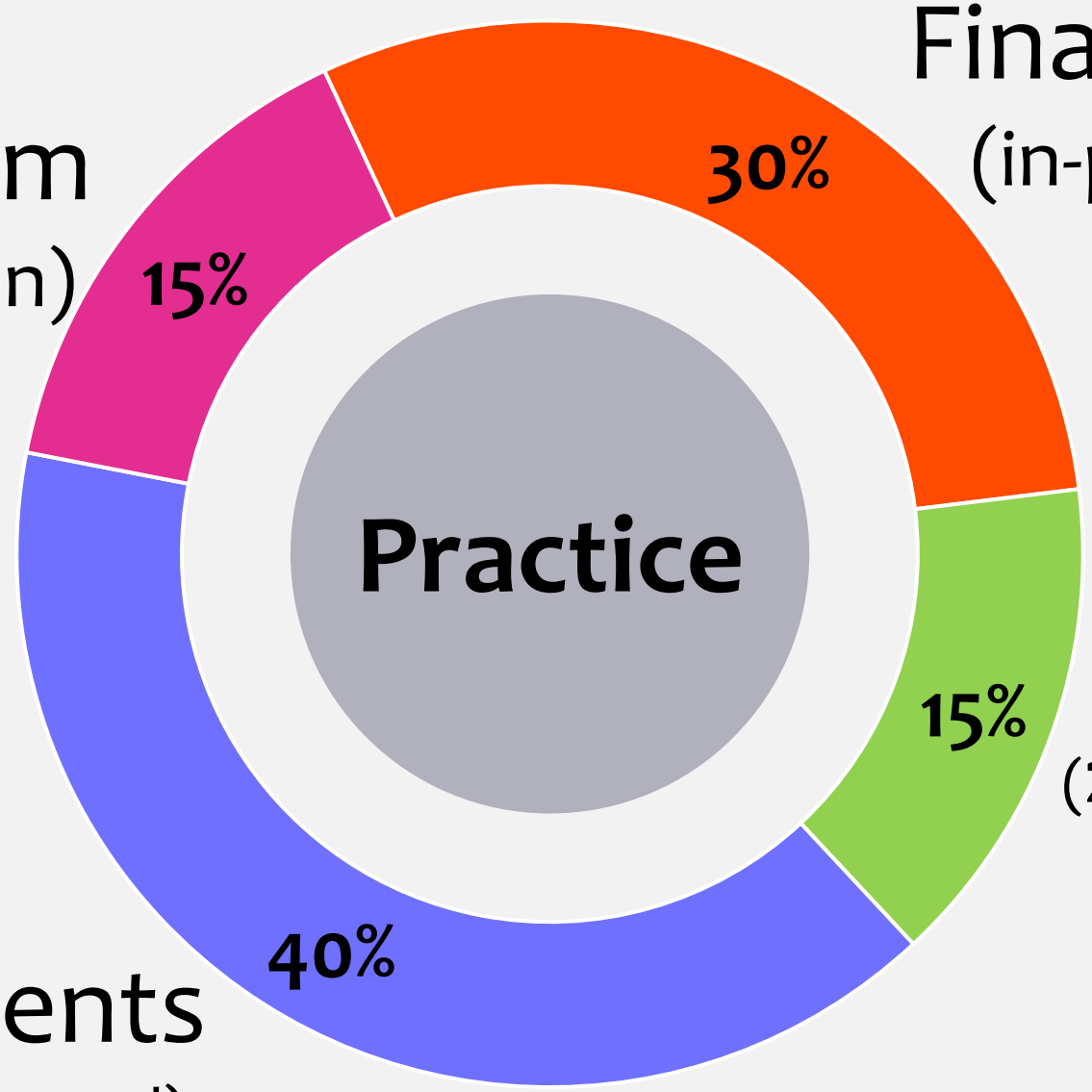
This course is about how to use, implement, and analyze data structures.

- The Java Collections Framework (JCF)
- Sequences: lists, stacks, queues, deques
- Array-based implementations of sequences
- Linked-list based implementations of sequences
- Unordered sets - hash tables
- Ordered sets - balanced search trees, skiplists
- Priority queues - heaps
- Sorting algorithms
- Graphs
- **Performance issues**

Evaluation

Midterm
(in-person)

Final Exam
(in-person)



11 Quizzes
(2 with the lowest mark will be dropped)

1 attempt; untimed within range

5 Assignments
(best 4 out of 5 counted)

Did you see the Outline?

- **A student may miss up to 1 assignment and 2 quizzes** for medical, compassionate, or other reasons without penalty.
  If you miss more than that, a mark of zero will be used for the missed items when the final grade is computed.

- Students with an **illness during the span of time a midterm** is offered might be granted an exemption. You need to **contact your instructor right away** and provide a copy of the Carleton University Self Declaration Form (https://carleton.ca/registrar/wp-content/uploads/self-declaration.pdf).
  The weight of the midterm will then be applied to the final exam mark.

What you will need for the course

Background:

- Review your Java programming
- Review discrete math (in particular, big-Oh notation and Sigma-notation for sums).

- Textbook (it is free). It was specifically written for this course.
- Install Java (command line compiler)
  If you do not have Java installed on your computer, you can download it for free at oracle.com
- Java Tutorials (optional) https://www.w3schools.com/java/default.asp
- Editor. The most popular choice is Visual Studio Code.

Assignments

Assignments are graded by an **autograder in Gradescope**.

- Instant feedback
- Submit often – your best grade is recorded
- TA time is allocated to helping you.

- No marks for trying.
- If your submission is missing files, or your code is not compiling for whatever reason, you will get a mark of **0**.

Assignment Workshops & Videos

Before each assignment, we will have two Assignment Workshops (online via Zoom) conducted by our TAs.

The workshops will be recorded, but please try to attend live workshops – this is an excellent opportunity to ask questions.

The goals of the workshops are to:

- Explain assignment specifications & requirements
- Give you some examples and answer your questions
- Show the base code and explain the purpose of the provided methods
- Show where to add your code, which methods to implement, and how to compile/run
- Show how to test your code locally and how to submit to the autograder

After each assignment deadline our TAs will post a Video explaining the solution to this assignment.

Assignments

- Start early and ensure you have significant time to work on your assignments – this will allow you to learn the concepts better
- Do not be late. The autograder cuts off – that's it. No late submissions can be accepted after that.

but you do get to drop the lowest assignment

- Collaboration/discussions are encouraged, but at a high level (general approach, useful tools, helpful resources)
- Do not share any of your work/code with others, do not copy someone else's work from any source (students, online, etc.)
  Plagiarism is a form of theft.

Academic Integrity

Electronic tools are in use to detect plagiarism

If your assignment is flagged, it will be sent to the Dean

Examples of Sanctions: **0 on assignment and reduction of final grade in the course**
F in the course
One-year suspension from program

Why you shouldn't plagiarize/cheat/etc…

- Employers don't really care that you have a degree – they care about the knowledge your degree represents.
- It isn't productive – what are you gaining?
- It isn't fair to others
- You are likely to get caught and penalized

Self-Study Spaces

The library and other lounge spaces on campus.

https://carleton.ca/studentsupport/2022/study-spaces-on-campus/

We have reserved supplementary space, and divided the rooms into two categories:

- Quiet study rooms (for viewing and independent work only);

  No need to book a seat.
  Use your own headsets.

- Interactive and group study rooms.

  The rooms may be used for group work and to meet/study/interact with your peers.
  You need to book the rooms: https://booking.carleton.ca

Quiet Study Space (830am-10pm Monday to Friday)	**Group Study Rooms (self service) (8:30 a.m. to 10:00 p.m. every day)**
TB 238 (cap 76)	CB 2103 (cap 7)
TB 240 (cap 78)	CB 3102 (cap 7)
ME 3190 (cap 36)	CB 3201 (cap 5)
	CB 2302 (cap 9)
PA 112 (cap 19)	LA B249 (cap 19)
PA 118 (cap 20)	SA 411 (cap 15)
LA A204 (cap 19)	PA 100A (cap 15)
LA B250 computer lab (cap 27)	PA 114 (cap 19)
	CB 3208
	SA 507
	UC 374

How to get good at this class

Our goal is not only to solve a problem, but to do it in the best way possible.

- Take notes
- Practice

  A rewarding aspect of programming is that once you have done a program correctly, you get to see it work.

- Discuss with other students, make friends
- Ask questions

  My goal #1 is to create a positive learning environment for everybody.

  I want you to succeed to the best of your ability. If there's something I can do to help you learn, please let me know.

- Support each other

By the end of this course, you will be a better problem solver!

CU Spirit Day Program

Starting **Friday** September 8, all students, staff and faculty are encouraged to wear CU Spirit Day shirt/sweater (or any Carleton gear) and continue to wear Carleton gear on every Friday.

For more information on the program, including some benefits, visit the website.

Let's celebrate Ravens spirit 

Abstract data types and algorithms

We will learn:

- how to be more careful and thoughtful about the way you program,
- how to store and manipulate data better, and how to choose the best way to do that for the situation that you're in.

You want programs that run

- Correctly
- Fast
- Reliably
- On huge inputs
- Space efficiently

3 Main Components

- **Data structures** is a collection of:
  - data values,
  - relationships among them,
  - functions/operations that can be applied to the data.

- **Algorithms**

An **algorithm** is a finite sequence of precise instructions (or steps) for solving a problem

1. Terminate?
2. Correct?
3. Efficient?
    1. Time complexity
    2. Space complexity

- **Abstraction** – hiding implementation details in order to reduce complexity and increase manageability

Data Structures

- What are they?

  Data Structures – store data,
  answer queries,
  (maybe) perform updates.

numbers, strings, documents, …

what is the data stored at position i? – get(i)
what is the "smallest" item? – first()

add/remove/change data
add the element x at position i – add(i,x)

- Why do we need them?

  Every part of a computer system has non-trivial data structures in it.

We interact with data structures all the time:

- Open a file
- Look up a contact on your phone
- Web search
- Log in
- GIS (geographic systems)

In this Course we will

- Define a new abstract data types (ADT), i.e an **interface** that provides methods that allow you to manipulate data in certain ways
- Discus various **implementations** of the ADTs using different structures and algorithms
- Discuss pros/cons.

Interface (ADT)

What does the data structure represent?

A collection, a set, a sequence,
a map, a graph, the world, …

What operations does it support?

adding, removing, finding elements;
membership testing;
range searching; …

Implementation

Performance: speed and memory

How long does each operation take?

How much space does it use?

example

Sets (or sorted sets)	Binary search trees
Sequences	Skiplists
Maps	Hash tables
Graphs	Adjacency Lists

You can have different implementations of the same interface

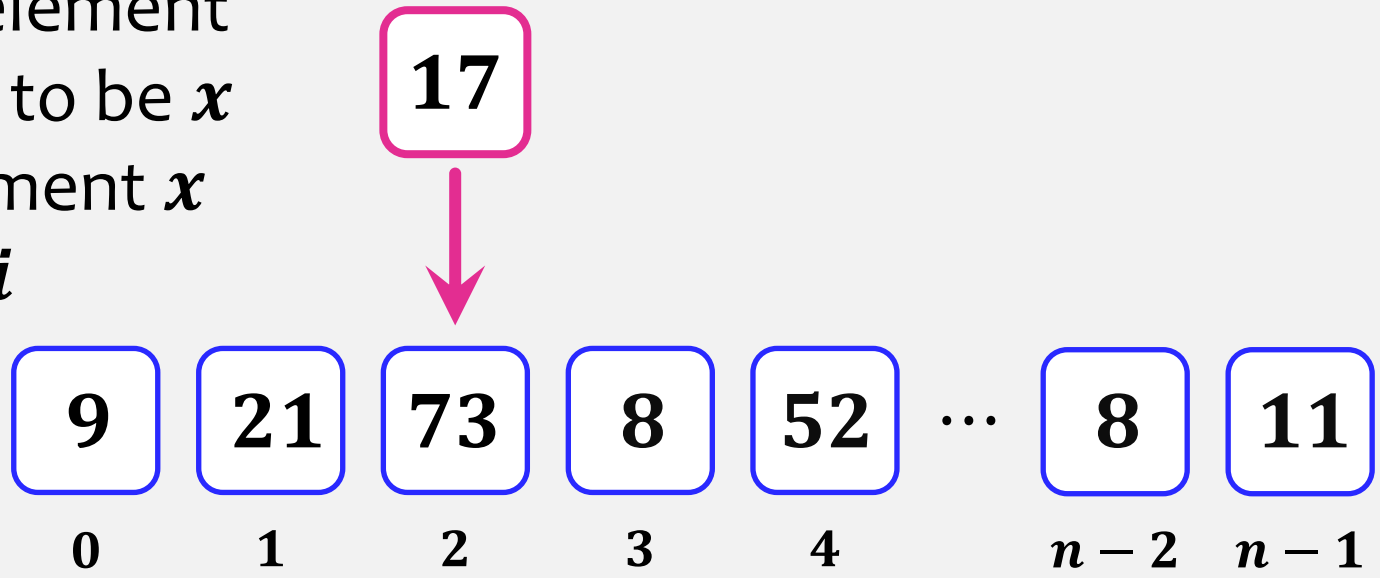
Abstract data types – List

Lists – store an indexed sequence of elements

Example: movie list(season 1,2 3,…)

Methods:

- size() – returns the number of elements on the list (n)
- get(i) – returns the element at position i
- set(i, x) – update the element at position i to be x
- add(i, x) – add the element x to position i



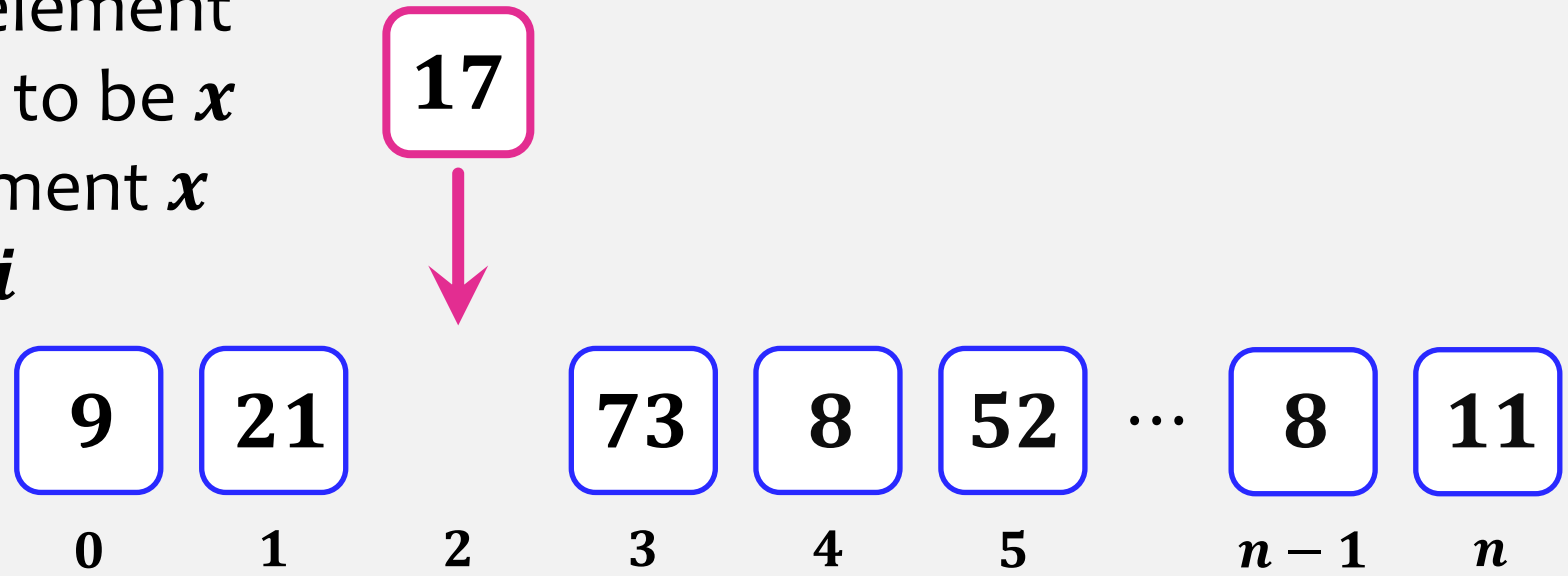
Abstract data types – List

Lists – store an indexed sequence of elements

Example: movie list(season 1,2 3,…)

Methods:

- size() – returns the number of elements on the list (n)
- get(i) – returns the element at position i
- set(i, x) – update the element at position i to be x
- add(i, x) – add the element x to position i
- remove(i) – remove element at position i

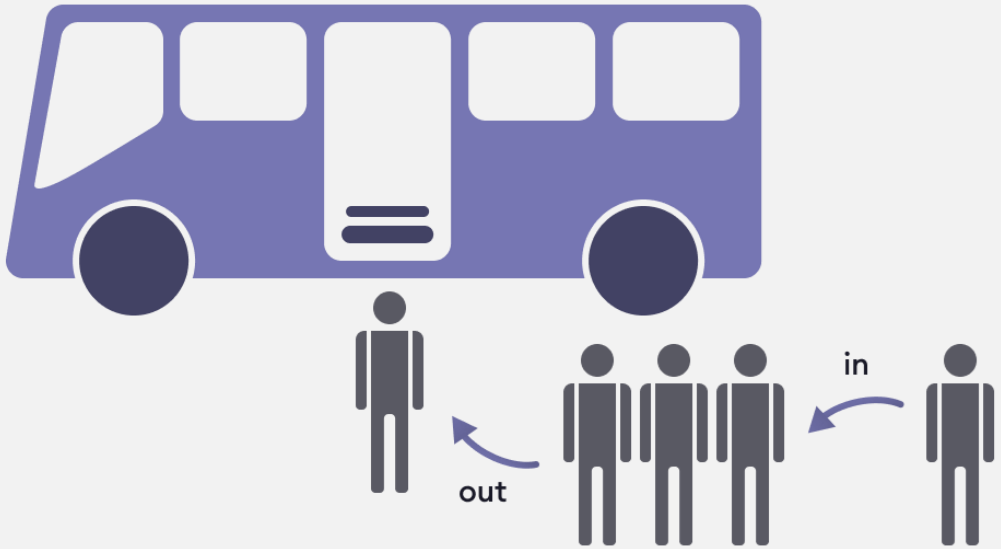


Abstract data types – Queue

Example: queue at the cash register

Queue – store elements that are accessed in first in, first out order (FIFO)
Not indexed. You can only access the oldest remaining element.

If we have **List** interface:
- add(size(),x)
- remove(0)



Methods:
- size()
- add(x) – add the element x to the end of the queue
- remove() – remove the first element of the queue

this element that has been in the queue the longest

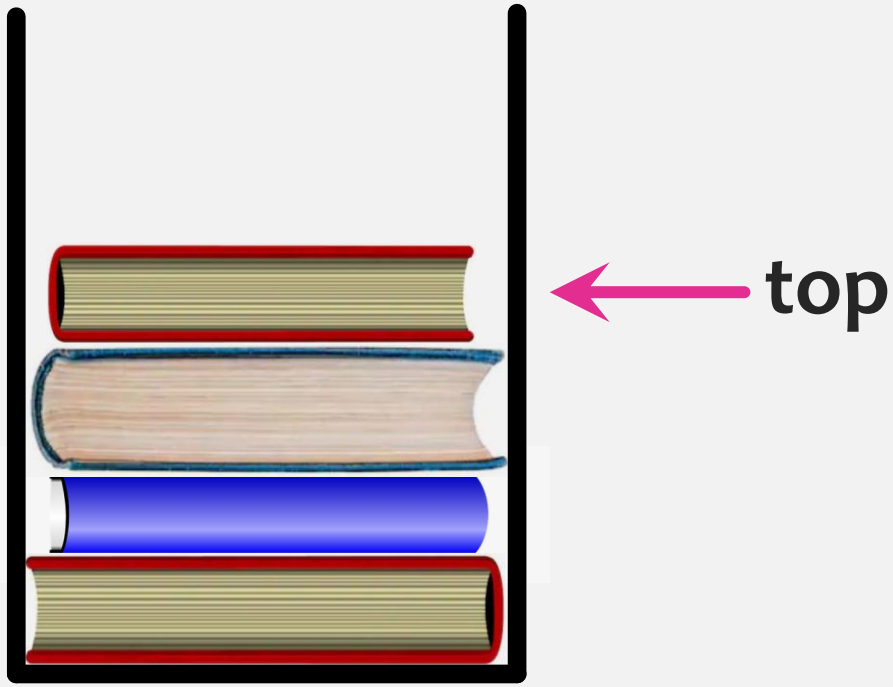
Abstract data types

Example: browser back button; stack of books or plates

Stack – store elements that are accessed in First-In-Last-Out order (FILO)

≡ Last-In-First-Out (LIFO);

No indices.



Methods:

- size()
- add(x) – add the element x to the top of the stack
- remove() – remove the element that was most recently added

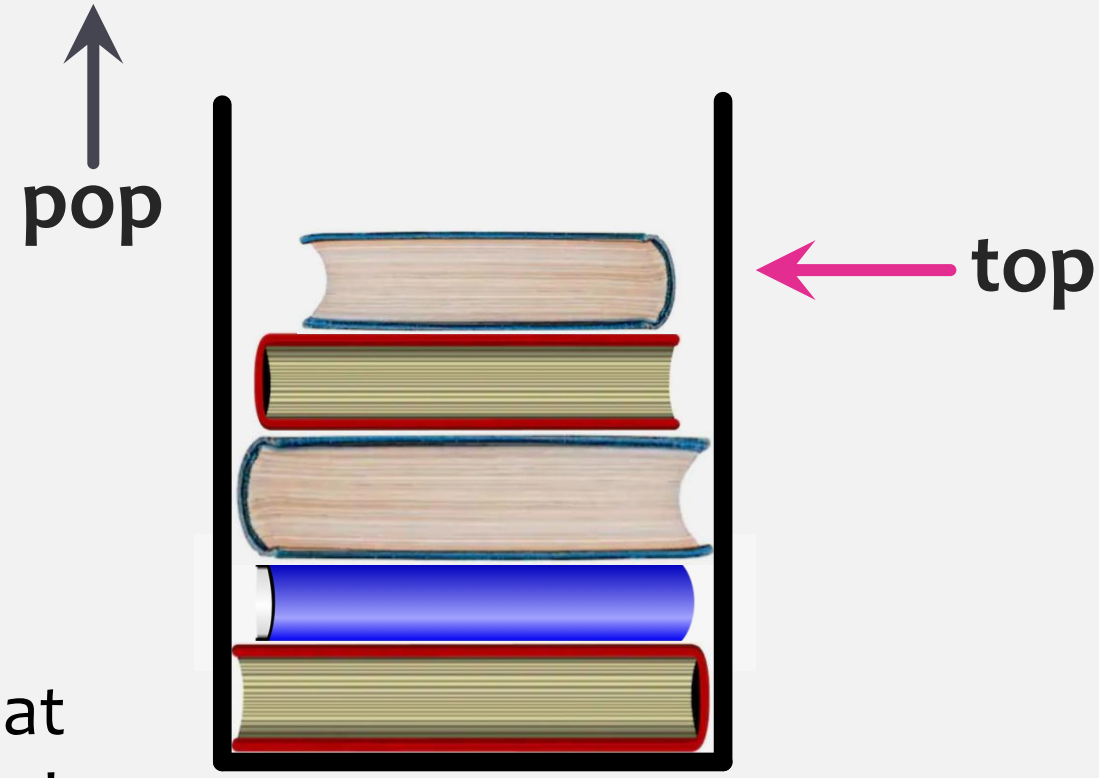
Abstract data types

Example: browser back button; stack of books or plates

Stack – store elements that are accessed in First-In-Last-Out order (FILO)

≡ Last-In-First-Out (LIFO);

No indices.



Methods:

- size()
- add(x) – add the element x to the top of the stack
- remove() – remove the element that was most recently added

Abstract data types

Priority Queue – stores elements that are accessed by min priority first.

Example: hospitals; VIP at clubs

Methods:

- size()
- add(x) – adds item with priority x
- remove() – removes the element with the lowest value/priority

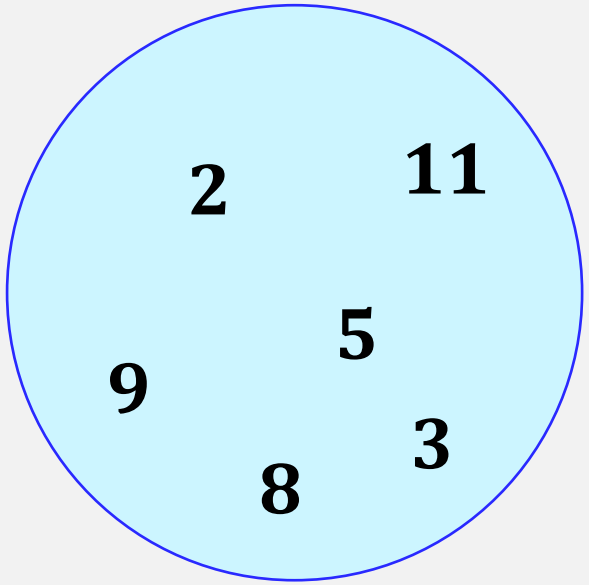
Abstract data types

Example: students in the class; contacts in my phone

Set – unordered collection of distinct elements

USet – unordered set

SSet – sorted set



Methods:

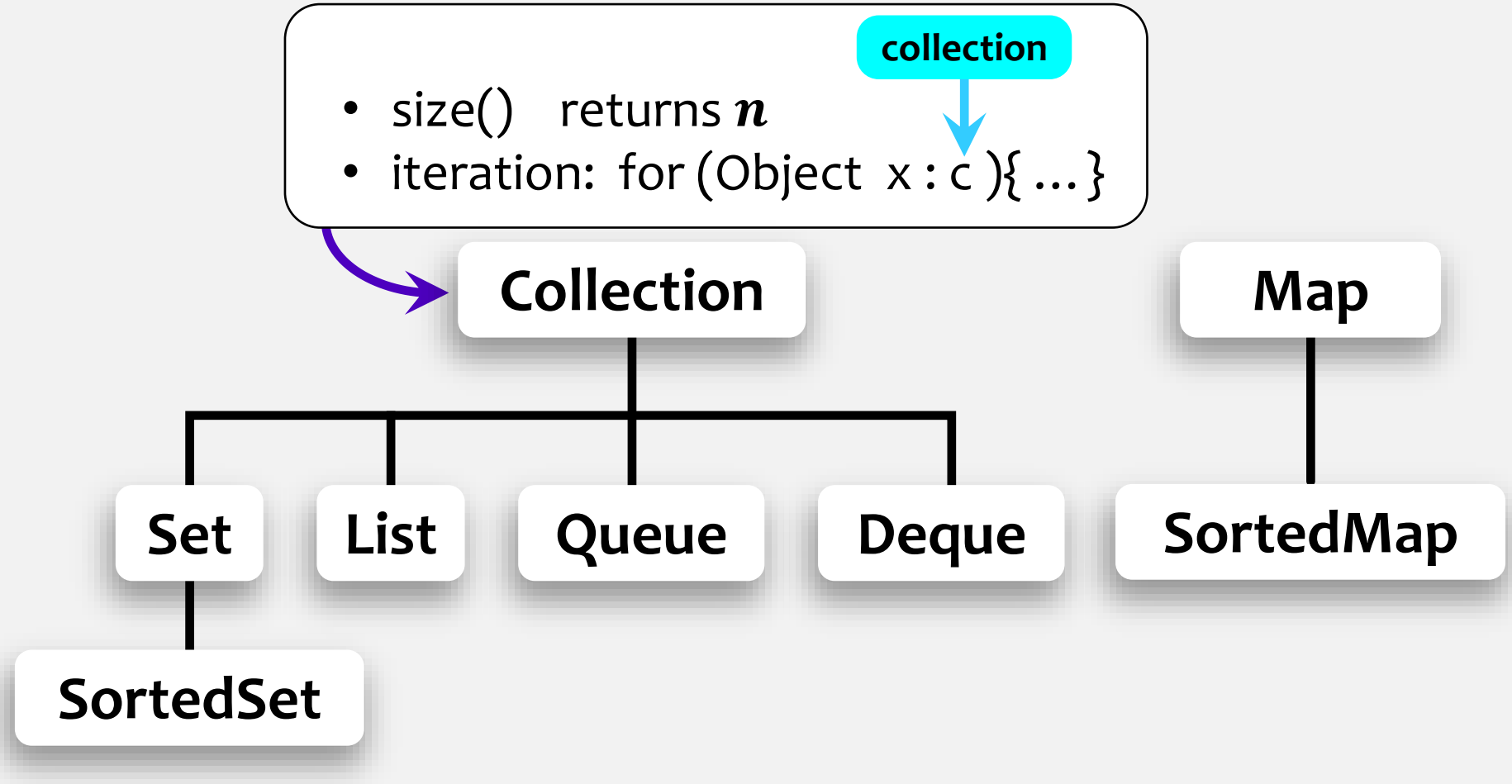
- size()
- add(x) – add the element x (if not already present); return **true** (**false**)
- remove(x) – remove and return x. Return **null** if no such element x exists
- find(x) – generally returns whether x is in the set, but not quite

  USet – returns any y that is equivalent to x.

  If x is not in the set, returns **null**.

  SSet – if x is not in set, returns successor → **min** $y \geq x$

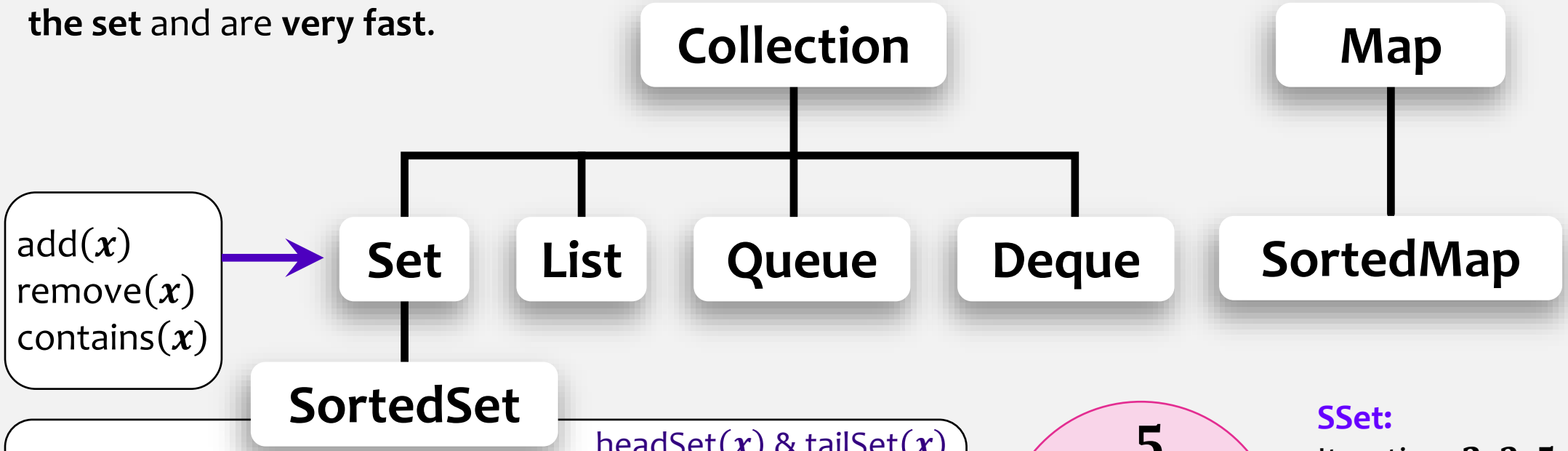
The Java Collections Framework: Interfaces



https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html

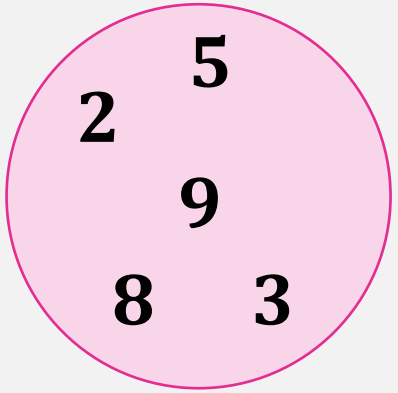
The Java Collections Framework: Interfaces

In a good set implementation, these operations on sets are **independent of the size of the set** and are **very fast**.



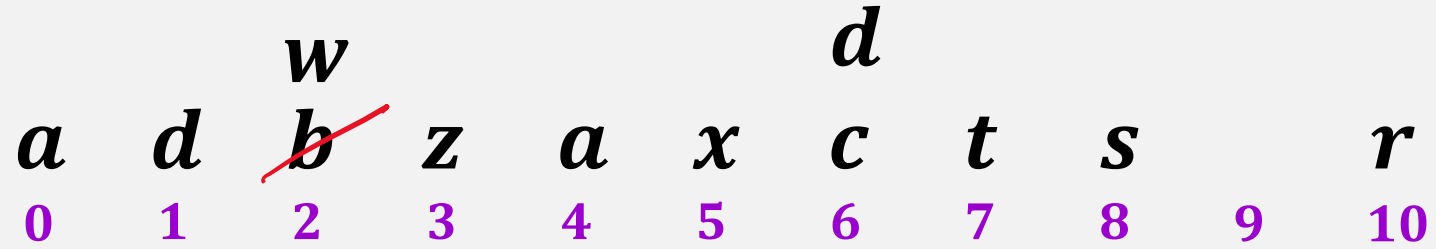
first() – smallest item in set
last() – largest item
headSet(x): returns everything smaller than x
tailSet(x): returns everything that is greater or equal to x
subSet(x, y): everything that is $\geq x$ and $< y$

headSet(x) & tailSet(x) return a view of the set; not a new copy



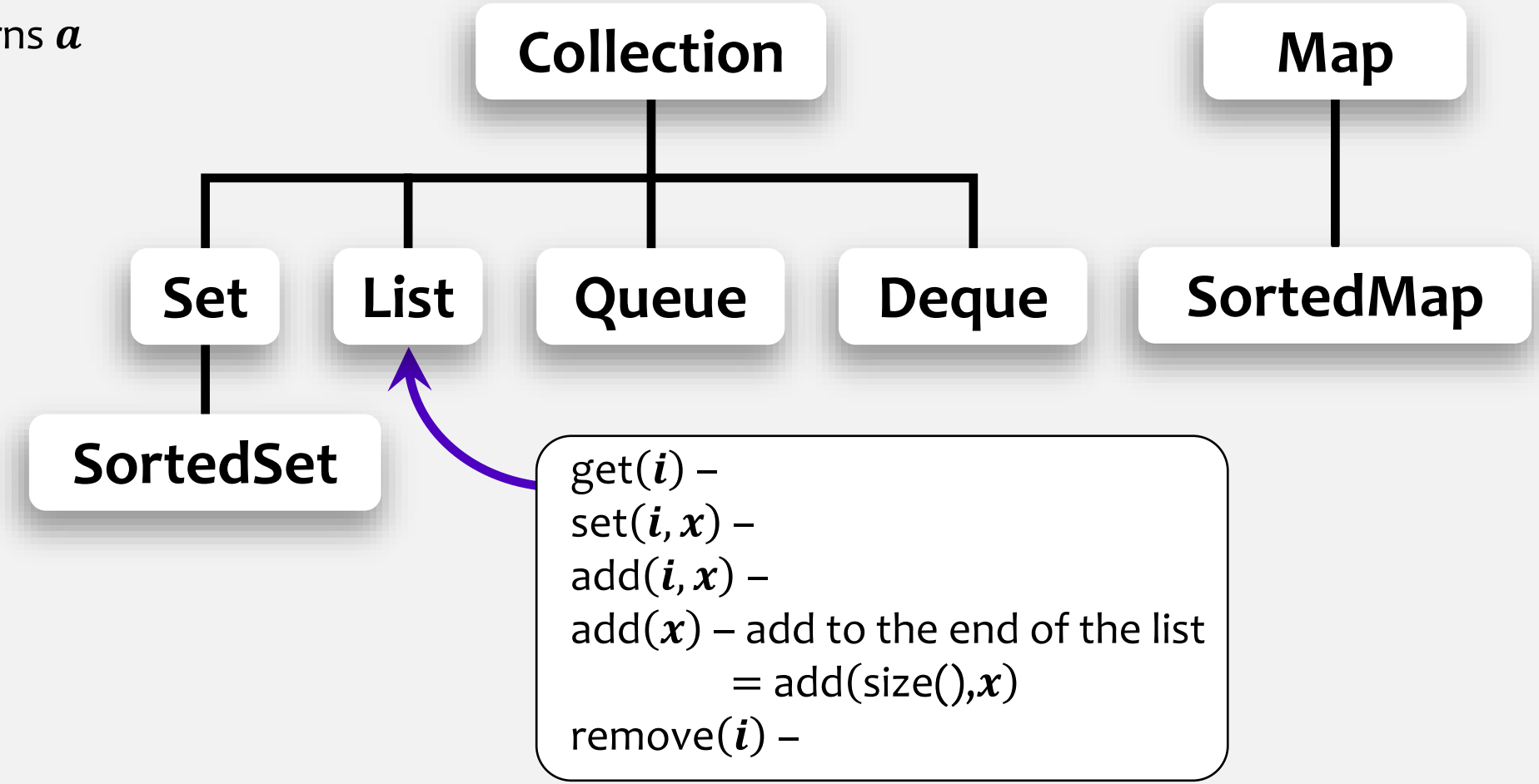
SSet:
Iteration: **2, 3, 5, 8, 9**
first() returns **2**
last() returns **9**
headSet(**6**) returns **2, 3, 5**
headSet(**2**) returns empty set
tailSet(**5**) returns **5, 8, 9**

The Java Collections Framework: Interfaces



List:

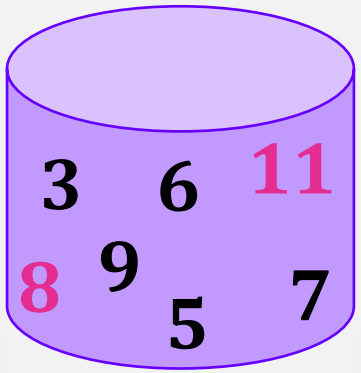
get(4) returns a
set(2, w)
add(6, d)
add(r)
remove(5)

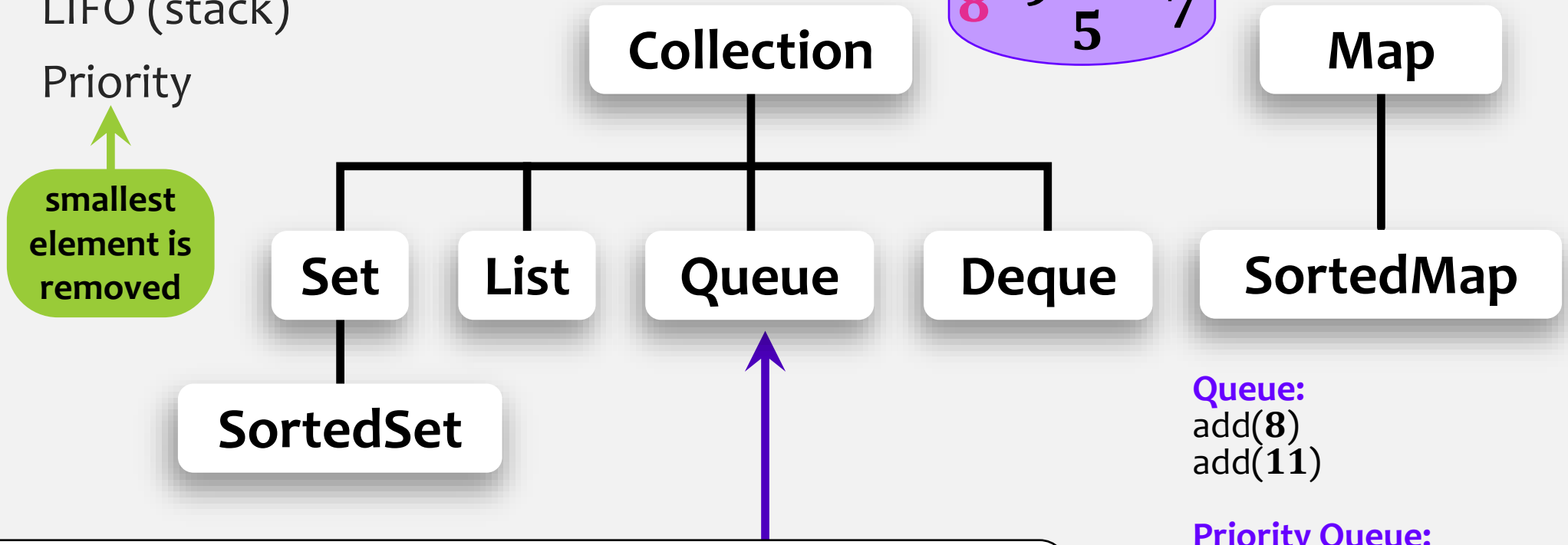


The Java Collections Framework: Interfaces

Three main kinds of queues:

1. FIFO
2. LIFO (stack)
3. Priority





add(x) –
remove() – remove one element from the queue and return it.
(the exact element depends on the "**queueing discipline**")

Queue:
add(**8**)
add(**11**)

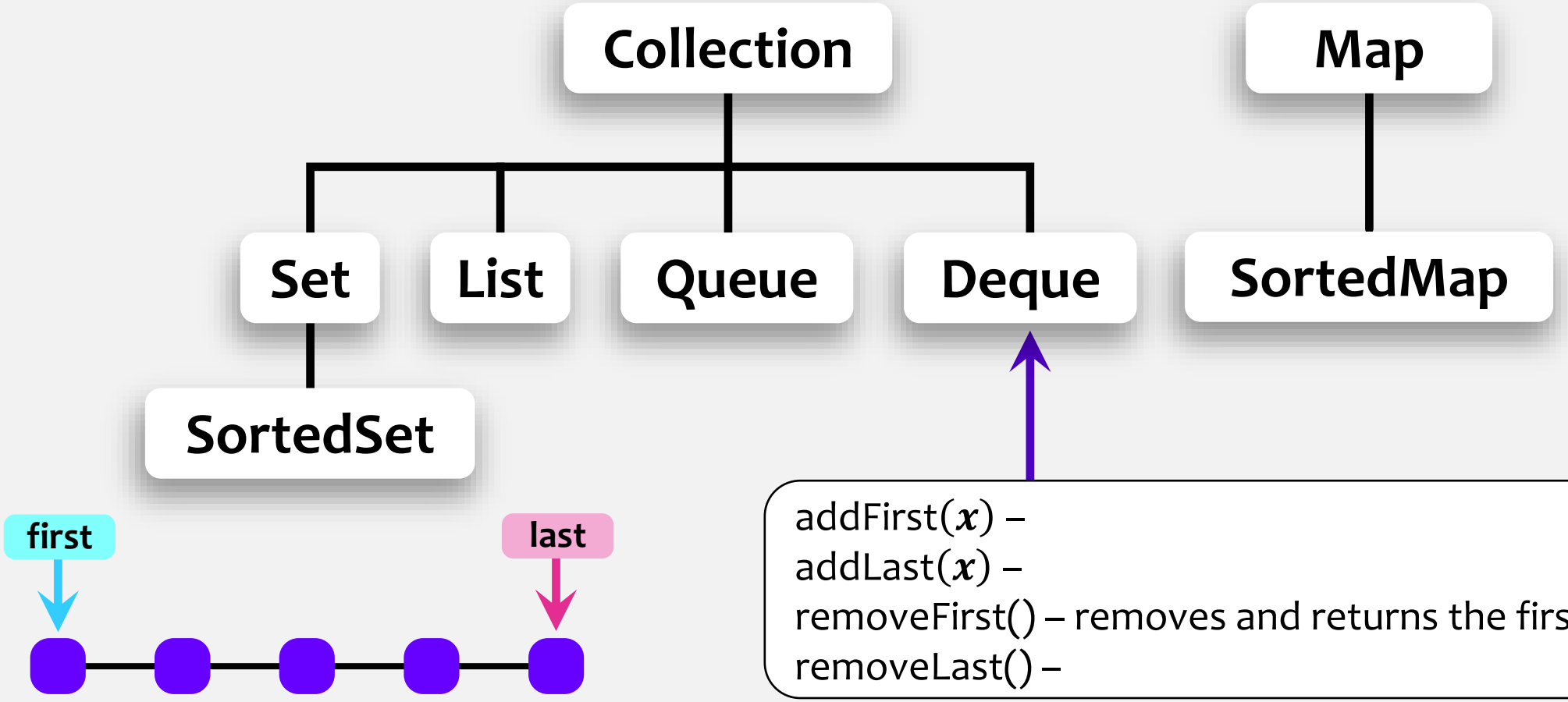
Priority Queue:
remove() – removes/returns **3**

LIFO Queue:
remove() – removes/returns **11**

The Java Collections Framework: Interfaces

Deque – double-ended queue (no indices)

It is a sequence, where you can add and remove to/from either end.



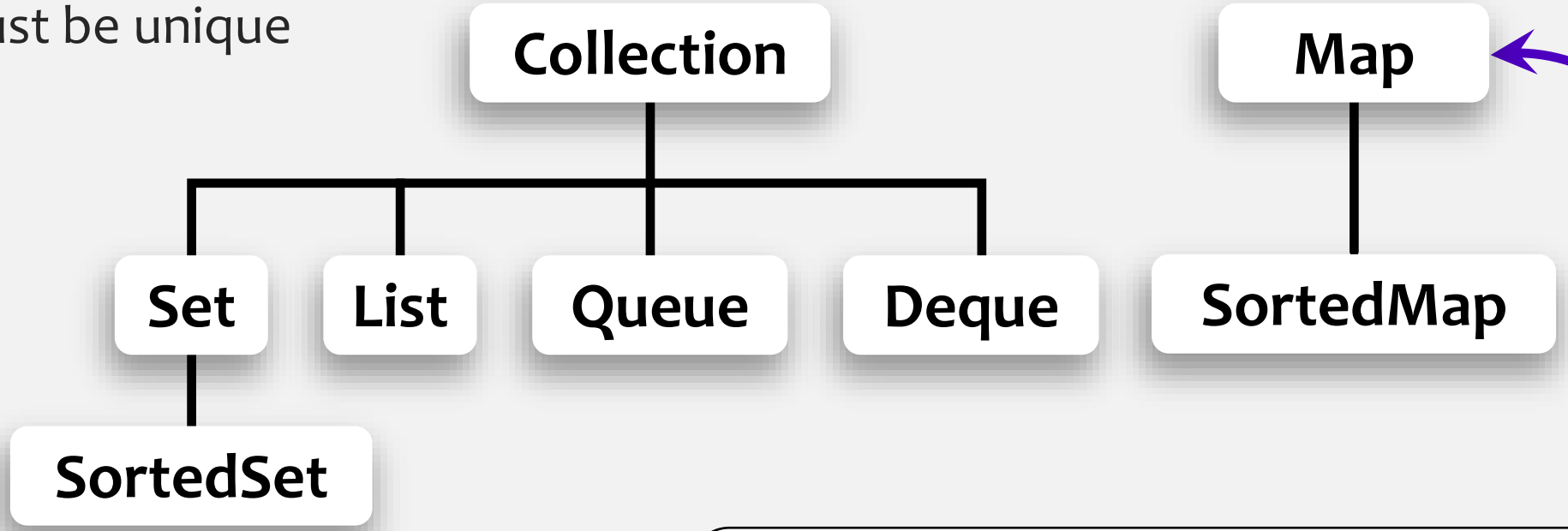
The Java Collections Framework: Interfaces

Map (dictionary)

- maps a set of keys to values
- Keys must be unique

Map:
put("***Mia***", **95**)

Key	Value
"Mia"	~~75~~ **95**
"Ray"	**80**



Sorted Map – a Map in which a key set is a **Sorted Set**

get($\boldsymbol{k}$) – returns the value associated with $\boldsymbol{k}$
put($\boldsymbol{k}, \boldsymbol{v}$) – associates the value $\boldsymbol{v}$ with the key $\boldsymbol{k}$
removeKey($\boldsymbol{k}$) – removes the key $\boldsymbol{k}$ from the key set
containsKey($\boldsymbol{k}$) –

Implementations

Example in Java:

```
List<String> l = new ArrayList<>();
```

Set → HashSet

SortedSet → TreeSet

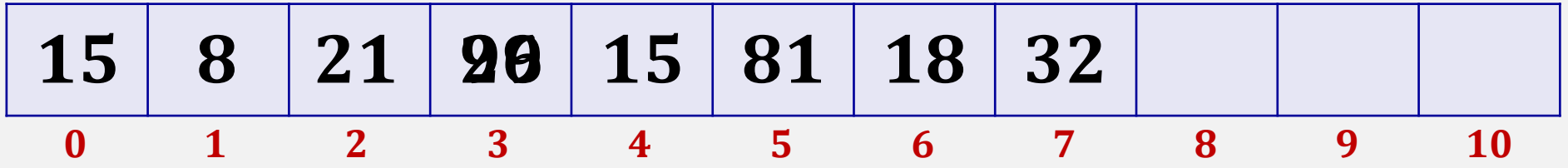
List interface – ArrayList implementation

List can be implemented using single array.

a = new Integer[11]

n = ~~8~~ 9

add(3,99)



i

get(i): return $a[i]$

set(i,x): $y \leftarrow a[i]$
$a[i] \leftarrow x$
return y

$0,..,n-1$

$0,..,n$

add(i,x): takes time $O(n-i+1)$

time for shifting

This operation is fast only when $i \approx n = \text{size}()$

This is slow. We need to move $n-i$ elements

What if array does not have that one extra position at the back?

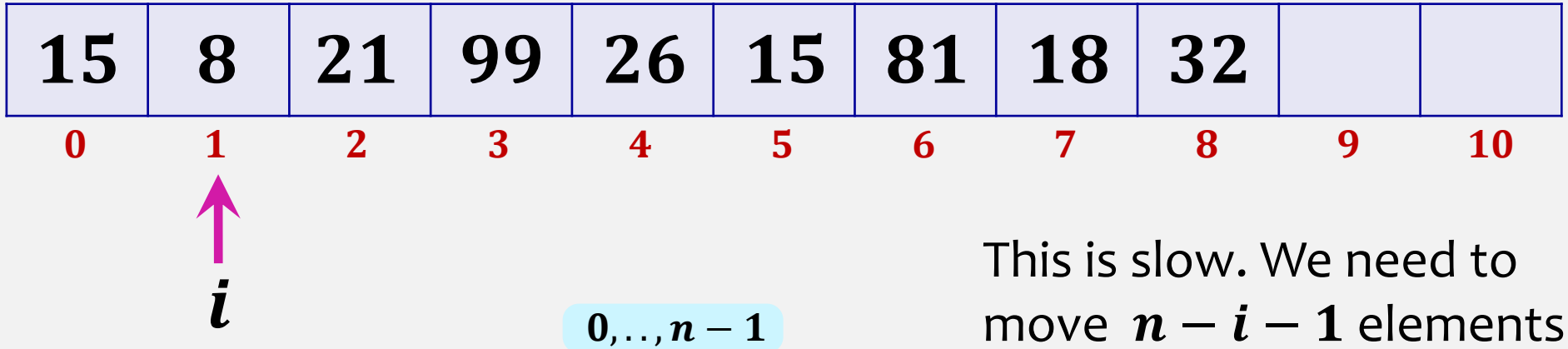
List interface – ArrayList implementation

List can be implemented using single array.

a = new Integer[11]

n = ~~8 9~~ 8

array a:



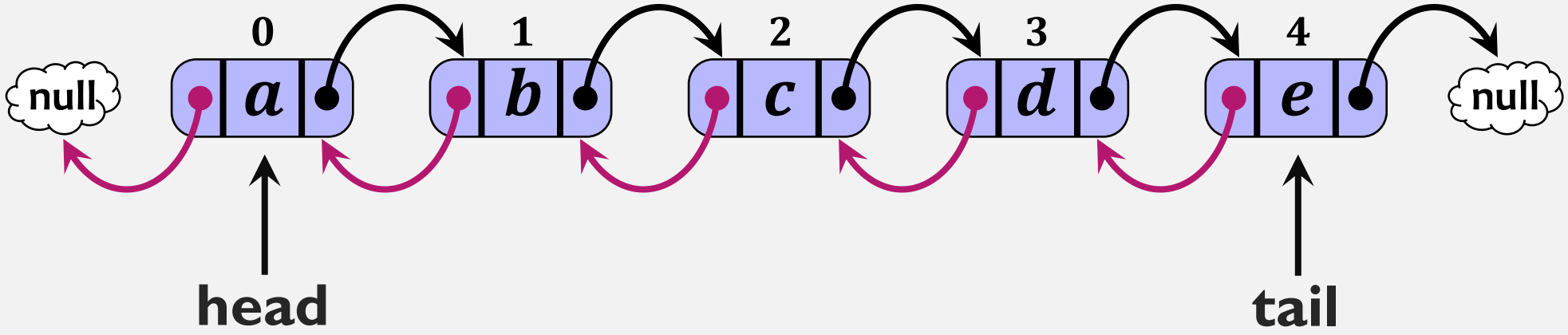
This is slow. We need to move $n - i - 1$ elements

$0, .., n - 1$

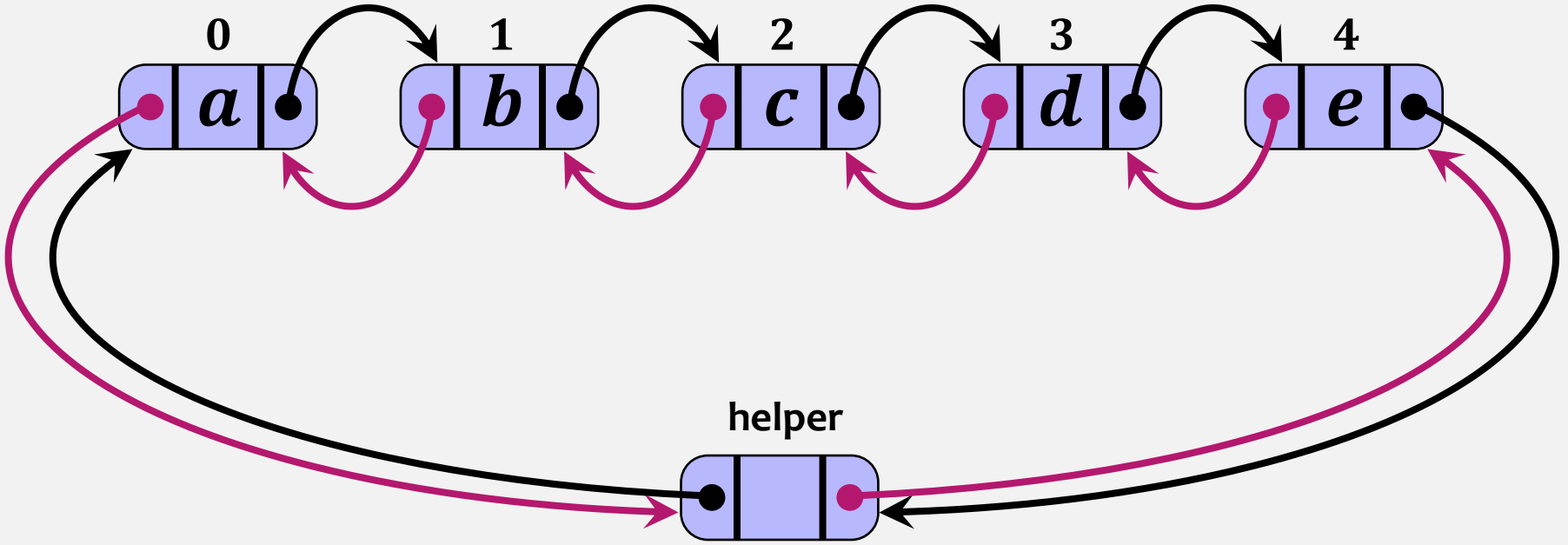
remove(i): takes time $O(n - i + 1)$

This operation is fast only when $i \approx n = \text{size}()$

List interface – LinkedList implementation

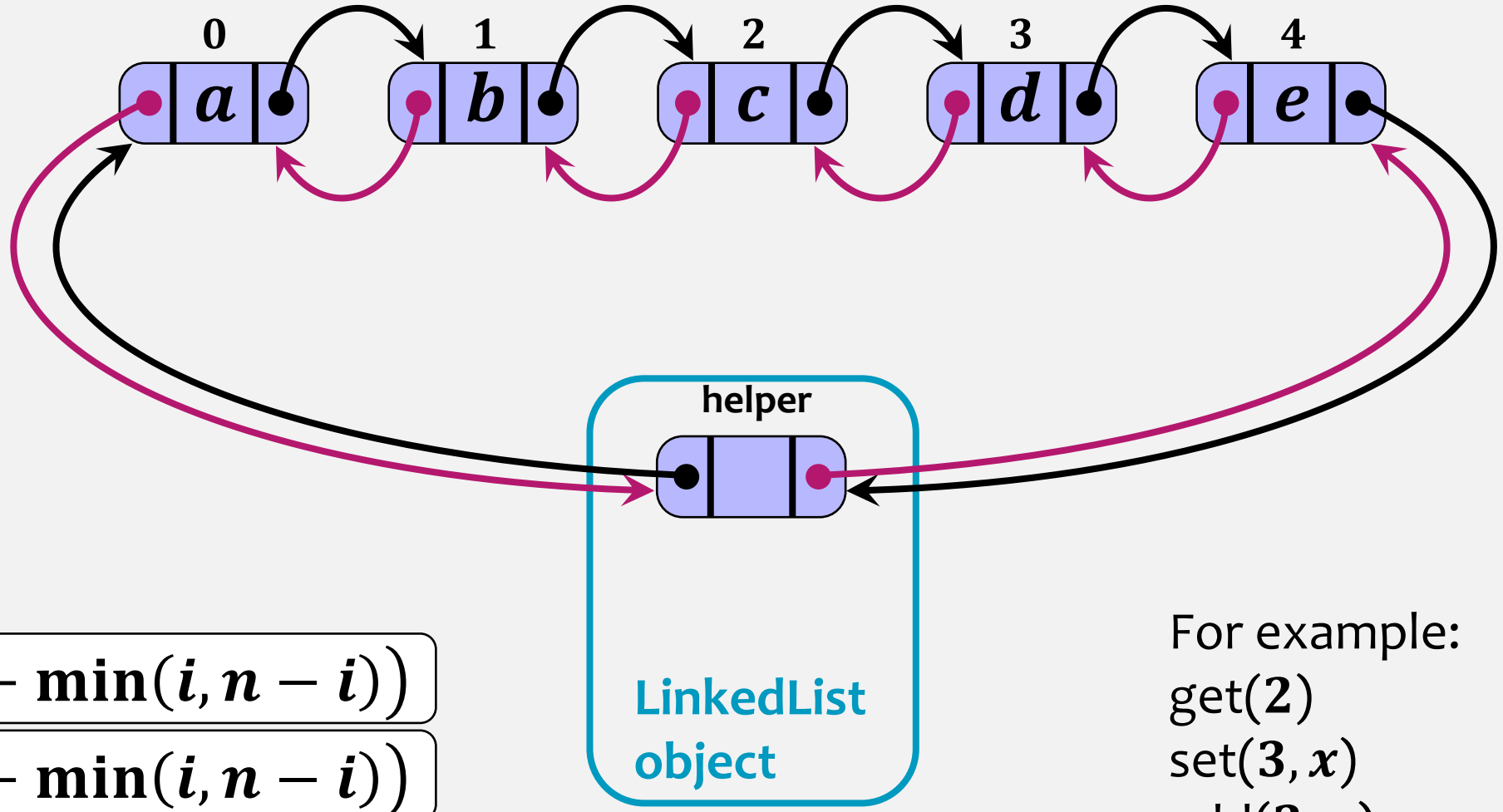


List interface – LinkedList implementation



List interface – LinkedList implementation

indices are implicit



get(i) $O(1 + \min(i, n - i))$

set(i, x) $O(1 + \min(i, n - i))$

add(i, x)

remove(i)

For example:

get(2)

set($3, x$)

add($3, a$)

remove(1)

LinkedList vs ArrayList

add/remove operations:

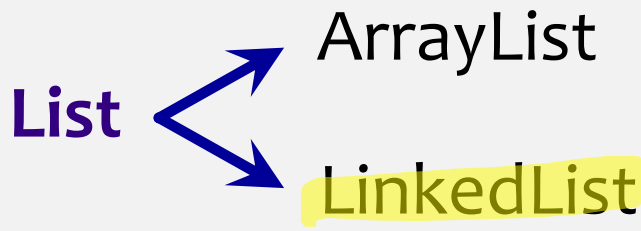
	LinkedList	ArrayList
	$O(1 + \min(i, n - i))$	$O(n - i + 1)$
If i is small:	**fast**	**slow**

Add/remove operations near each end of the list are **fast** for **LinkedList**

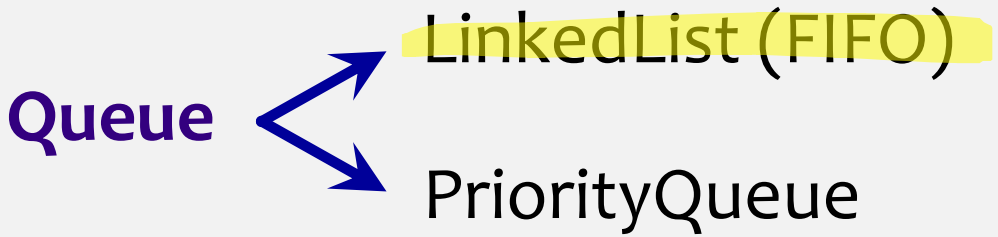
ArrayList is great when you need random access without adding or removing elements.

LinkedList does not give you fast random access, but it is very fast for adding/removing at either end.

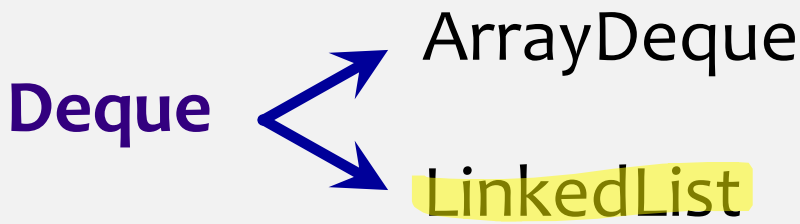
Implementations



Set → HashSet



SortedSet → TreeSet



How to get started

- Review asymptotic analysis (Big-$\boldsymbol{O}$, $\boldsymbol{\Omega}$, $\boldsymbol{\Theta}$)
- Get familiar with the course's website.
- Read outline of the course and the Calendar
- Introduce yourself in the discussion forum
- Start working on **assignment 1** as soon as it gets posted
- Office Hours start Friday, September 15th
- And practice, practice, practice!