This week (Chapter 2):

- Abstract data types (**Today**)
- Data structures "discovery lab" (Thursday)
- Data Structures (Friday and next week Monday)
- Programming practices (next week Monday)
- (Begin Case Studies next week Wednesday)

Today:

- Algorithmic elements (finishing Algorithms and Efficiency)
- Definition *abstract data type*, especially in contrast with *data structure*
- The "canonical" ADTs
- Introduction to conventions of our code base

	Best case	Worst case	Expected case
Bounded linear search			
Binary search			
Quick sort			

Algorithmic element 1

Can you jump directly to the thing you're looking for?

Algorithmic element 2

Are you descending a binary tree of the data?

Algorithmic element 3

Do you need to touch every element in the data?

Algorithmic element 4

For every element, do you need to descend a tree, or for every element in the tree, do you touch every element?

Algorithmic element 5

For every element in the data, do you need to a suboperation on the rest of the data?

Algorithmic element 6

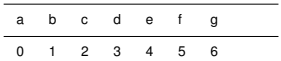
Do you need to consider all combinations of input elements?

An abstract data type (ADT) is a data type whose representation is hidden from the client. Implementing an ADT as a Java class is not very different from implementing a function library as a set of static methods. The primary difference is that we associate data with the function implementations and we hide the representation of the data from the client. When using an ADT, we focus on the operations specified in the API and pay no attention to the data representation; when implementing an ADT, we focus on the data, then implement operations on that data.

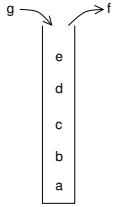
[Sedgewick and Wayne, *Algorithms*, Pg 64; also cf pg 84]

The "canonical ADTs":

- **List.** Linear collection with sequential and random access.
- **Stack.** Linear collection with LIFO access.
- **Queue.** Linear collection with FIFO access.
- **Set.** Unordered collection with binary membership.
- **Bag.** Unordered collection with enumerated membership.
- **Map.** Unordered collection of associations between keys and values.

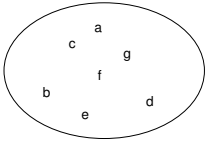


List

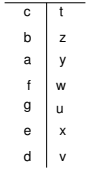


Stack

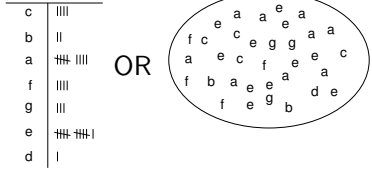
Queue



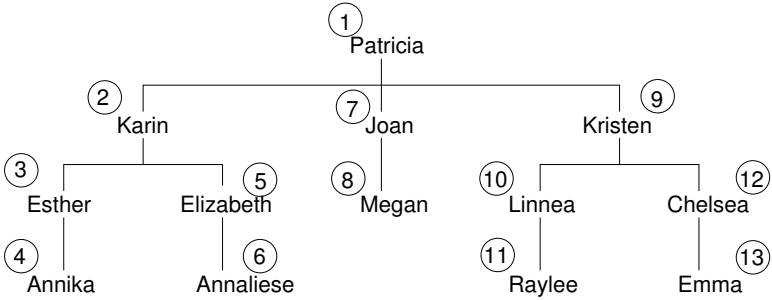
Set

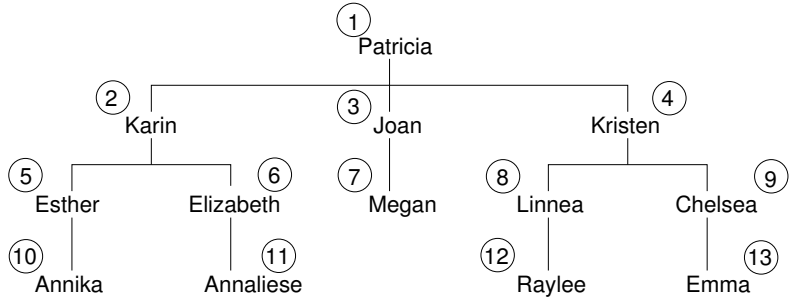


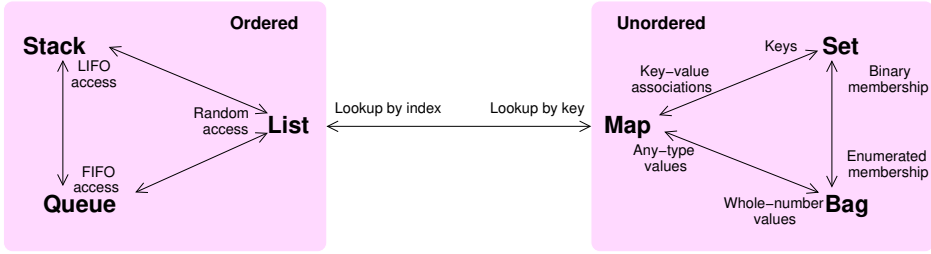
Map

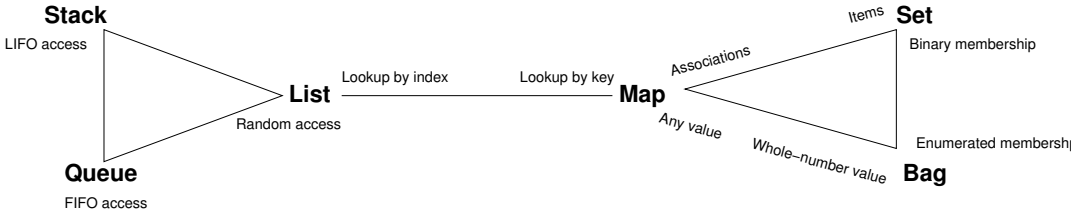


Bag


1
Patricia
2
Karin
7
Joan
9
Kristen
3
Esther
5
Elizabeth
8
Megan
10
Linnea
12
Chelsea
4
Annika
6
Annaliese
11
Raylee
13
Emma


1
Patricia
2
Karin
3
Joan
4
Kristen
5
Esther
6
Elizabeth
7
Megan
8
Linnea
9
Chelsea
10
Annika
11
Annaliese
12
Raylee
13
Emma


Ordered
Stack
LIFO
access
Random
access
List
FIFO
access
Queue
Lookup by index
Lookup by key
Unordered
Keys
Set
Key−value
associations
Binary
membership
Map
Any−type
values
Enumerated
membership
Whole−number
values
Bag


Stack
LIFO access
Queue
FIFO access
List
Random access
Lookup by index
Lookup by key
Map
Associations
Any value
Whole−number value
Items
Set
Binary membership
Enumerated membershp
Bag

The four basic ways to implement an ADT:

- Use an array
- Use a linked structure
- Use an "advanced" data structure, varying and/or hybridizing linked structures and arrays
- Adapt an existing implementation of another ADT.

Coming up:

Due **Wed, Sept 10***:*
Finish reading Section 2.1
Do Ex 1.11
Take ADT quiz

Due **Mon, Sept 15***:*
Read Section 2.(2, 4, & 5)
Take data structures quiz

Also:
Do **"Implementing ADTs"** *project (Due Fri, Sept 19)*