1 Assorted ADTs

A **list** is an ordered collection, or *sequence*.

```java
List
add(element); // adds element to the end of the list
add(index, element); // adds element at the given index
get(index); // returns element at the given index
size(); // the number of elements in the list
```

A **set** is a (usually unordered) collection of unique elements.

```java
Set
add(element); // adds element to the collection
contains(object); // checks if set contains object
size(); // number of elements in the set
remove(object); // removes specified object from set
```

A **map** is a collection of key-value mappings, like a dictionary in Python. Like a set, the keys in a map are unique.

```java
Map
put(key, value); // adds key-value pair to the map
get(key); // returns value for the corresponding key
containsKey(key); // checks if map contains the specified key
keySet(); // returns set of all keys in map
```

Stacks and queues are two similar types of linear collections with special behavior. A **stack** is a last-in, first-out ADT: elements are always added or removed from one end of the data structure. A **queue** is a first-in, first-out ADT. Both data types support three basic operations: push(e) which adds an element, peek() which returns the next element, and poll() which returns and removes the next element.

Java defines an interface that combines both stacks and queues in the **Deque**. A **deque** (double ended queue, pronounced “deck”) is a linear collection that supports element insertion and removal at both ends.

```java
Deque
addFirst(e); // adds e to front of deque
removeFirst(); // removes and returns front element of deque
getFirst(); // returns front element of deque
addLast(e); // adds e to end of deque
removeLast(); // removes and returns last element of deque
getLast(); // returns last element of deque
```
Generally-speaking, a **priority queue** is like a regular queue except each element has a priority associated with it which determines in what order elements are removed from the queue.

```java
PriorityQueue
    add(e); // adds element e to the priority queue
    peek(); // looks at the highest priority element, but does not remove it from the PQ
    poll(); // pops the highest priority element from the PQ
```

## 2 Solving Problems with ADTs

For each problem, which of the ADTs given in the previous section might you use to solve each problem? Which ones will make for a better or more efficient implementation?

(a) Given a news article, find the frequency of each word used in the article.

(b) Given an unsorted array of integers, return the array sorted from least to greatest.

(c) Implement the forward and back buttons for a web browser.
Java supports many built-in ADTs and data structures that implement these ADTs. But if we want something more complicated, we'll have to build it ourselves.

If you wish to use sorting as part of your design, assume that it will take $\Theta(N \log N)$ time where the length of the sequence is $N$.

(a) Suppose we want an ADT called BiDividerMap that allows lookup in both directions: given a value, return its corresponding key, and vice versa. It should also support numLessThan which returns the number of mappings whose key is less than a given key.

1. BiDividerMap
2. put(k, V); // put a key, value pair
3. getKey(K); // get the value corresponding to a key
4. getValue(V); // get the key corresponding to a value
5. numLessThan(K); // return number of keys in map less than K

Describe how you could implement this ADT by using existing Java ADTs as building blocks. Come up with an idea that is correct first before trying to make it more efficient.

(b) Next, Suppose we would like to invent a new ADT called MedianFinder which is a collection of integers and supports finding the median of the collection.

1. MedianFinder
2. add(x); // adds x to the collection of numbers
3. median(); // returns the median from a collection of numbers

Describe how you could implement this ADT by using existing Java ADTs as building blocks. What’s the most efficient implementation you can come up with?
Define a Queue class that implements the push and poll methods of a queue ADT using only a Stack class which implements the stack ADT.

*Hint:* Consider using two stacks.