Here are some Abstract Data Types that you have seen in class so far. We want to get some practice with using them. First, let's review the following ADTs.

List

A **list** is an ordered sequence of items. It is like an array, but can have variable length/size.

```
interface List<T> {
  void    add(T item);
  void insert(T item, int position); // insert item at this position
  T       get(int position);
  int    size();
}
```

Note: `List` is an interface; you cannot directly make a List. However, ArrayLists are Lists, and you can make them with: `List myList = new ArrayList<T>()`, where `T` is the item type.

Set

A **set** is an unordered collection of non-duplicate items.

```
interface Set<T> {
  void         add(T item);  // If item is already in the set, nothing changes.
  boolean contains(T item);
  List<T>    items();        // return a List of all items in some arbitrary order
  int         size();
  void      remove(T item);
}
```

Note: `Set` is an interface; you cannot directly make a Set. However, HashSets are Sets, and you can make them with: `Set mySet = new HashSet<T>()`, where `T` is the item type.

Map

A **map** associates or "maps" keys to values. Python has this concept too! In Python they're called dictionaries. You can also think of a map as a set of <key, value> pairs, except that keys cannot be duplicated, and that looking up an value by key is fast (constant time).


```
interface Map<K, V> {
  void    put(K key, V value); // put key into the map and associates it with
                               // value, replacing old value if it exists
  boolean containsKey(K key);
  List<K> keys();              // returns a List of all keys in some arbitrary order
}
```

Note: `Map` is an interface; you cannot directly make a Map. However, HashMaps are Maps, and you can make them with: `Map myMap = new HashMap<K, V>()`, where `K` is the key type and `V` is the value type.

Now let's write some methods!

1. Does arr have duplicates?

```
public static boolean findDups(int[] arr) {









}
```

2. Do any two elements in the array sum up to n?

Hint: You don't need to check all combinations.

```
public static boolean sumUp(int n, int[] arr) {









}
```

3. Missing number

arr contains all the numbers from `0` to `n` for some `n` except some number `k`.
Find `k`. *Don't worry about what happens if the precondition is not met.*

```
public static int missingNumber(int[] arr) {









}
```

4. Is s1 a permutation of s2?

To review: The permutations of `cat` are: `cat`, `cta`, `act`, `atc`, `tca`, `tac`.
Hint: Use a Map.

```
public static boolean isPermutation(String s1, String s2) {









}
```

5. Finding duplicates within a range

Given an `int[]` a and a boundary range `k`, find if there are any duplicates that are within `k` indices of each other. Examples:

- `findDuplicatesWithinK([1,2,3,1,4,3], 3) -> [1,3]`
- `findDuplicatesWithinK([1,2,3,1,4,3], 2) -> []`

Hint: If you end up with a `Set` or `List` of duplicates, here's how you can convert it to an array:

```
your_set_or_list.toArray(new int[your_set_or_list.size()])
```

```
public static int[] findDuplicatesWithinK(int[] a, int k) {









}
```