# Mastery II — Data Structures (T. II/20–21)

**Directions:**

- You have 9 hours and 50 minutes to complete the following *four* problems. Each problem has a preset performance goal. Meeting the goal with a correct answer will earn you full credit. Partial credits will be given to code that returns correct answers but doesn't run as fast.
- No collaboration of any kind whatsoever is permitted during the exam.
- You are expected to use an IDE.
- **WHAT IS PERMITTED:** The exam is open- book, notes, Internet, Google, Stack Overflow, etc.
- **WHAT IS *NOT* PERMITTED:** Communication/collaboration of any kind.
- We're providing a starter package. The fact that you're reading this PDF means you have successfully downloaded the starter pack. For ease, we're putting all the starter files in the same folder: `src/main/java`.
  Also, to save you some typing, we're supplying some unit tests in the starter package. Passing these tests doesn't mean you will pass our further testing. Failing them, however, means you will not pass the real one.

**Handing In:** To hand in, zip ***only*** the following files as `mastery2.zip` and upload your zip file to Canvas:

`AllAliases.java` `LanCable.java` `MinOps.java` `RecursivePal.java`

This means your code should only be in these files and nowhere else.

## Problem 1: Recursive Palindrome (10 points)

An array is a *palindrome* if it reads the same forward and backward. For example, `{4, 0, 4}` is a palindrome but `{2, 0, 4}` is not. From this definition, we know that if `a` is a palindrome, then the first half (precisely the first `len(a)//2` elements) is the reverse of the last half (precisely the last `len(a)//2` elements). Importantly, comparisons are made using the corresponding object's `.equals`.

...let's up the game a bit. An array `a` is said to be a *recursive palindrome* if (i) the array is a palindrome itself and (ii) its first half (i.e., the first `len(a)//2` element) is a recursive palindrome or empty. For example:

- `{1, 1, 5, 1, 1}` is a recursive palindrome. Note that the whole array is a palindrome. Further, the first half (i.e., `1, 1`) is a recursive palindrome.
- `{7, 8, 7, 7, 8, 7}` is a recursive palindrome.
- `{2, 0, 4, 0, 2}` is *not* a recursive palindrome. While the whole array is a palindrome, the first half (i.e., `2, 0`) is not a palindrome—and hence *not* a recursive palindrome.
- `{7, 4, 5, 5, 4, 7, 9, 7, 4, 5, 5, 4, 7}` is *not* a recursive palindrome.

**Your Task:** Inside the class `RecursivePal<T>`, write a public method

```
public class RecursivePal<T> {
    public boolean isRecursivePalindrome(T[] a) { ... }
}
```

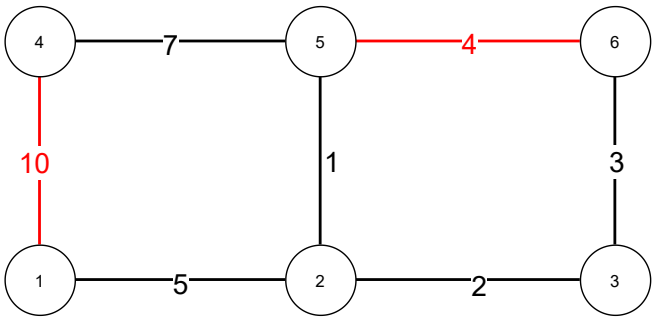
that takes as input an array `a` and returns a boolean indicating whether `a` is a recursive palindrome.

### Expectations: Promises, Constraints, and Grading

- $1 \leq$ `a.length` $\leq 50,000,000$.
- For full-credit, your code must run in $O(n)$ time, where $n =$ `a.length`. Your code must finish within 1 second on each of the test cases.
- You can write as many helper functions as you'd like.

## Problem 2: LAN Cable (10 points)

There are a number of network routers at MUIC. They are connected to each other by LAN cables either directly or indirectly. Lately, it has been observed that there are too many unnecessary cables in the network. Some cables can be safely disconnected and the network stays connected as more than one routes are available.




Consider the example above: there are 6 routers and 7 cables. Notice how we can remove some cables from the network without disconnecting it. For example, if we just remove the cable of length 7, the network is still connected. If we remove 7 and 4, the network's still connected and the total length removed is 11. Can we do better? Yes, if we remove 10 and 4, then the total is 14 and the network remains connected. However, if we remove 10 and 7, the total length is higher, but doing so will disconnect router 4 from rest of the network. Similarly, we will have the same problem with 10 and 5. Therefore, for this network, the maximum total length of cables we can remove is 11.

In this problem, we want to find the maximum total length of cables that can be disconnected from a given network with the condition that all routers must still be connected. Specifically, we are given a network with $n$ routers $R = \{1, 2, \ldots, n\}$ and $m$ cables $\mathrm{C} = \{(u_1, v_1, l_1), (u_2, v_2, l_2), \ldots, (u_m, v_m, l_m)\}$ where $u_i \in R$ and $v_i \in R$ denote the two routers connected by cable $i$, and $l_i$ denote the length of cable $i$. You may assume that $l_i$'s are unique, i.e., no duplicate lengths. You are to maximize the total length of cables removed while maintaining its connectivity. (*Hint:* How is this related to the MST?)

**Your Task:** Inside the class `LanCable`, write a public method

```
public long findMaxTotalLength(int n, int[] u, int[] v, int[] l)
```

where `n` is the number of routers, `u[i]` and `v[i]` represents the two routers connected by cable $i$, and `l[i]` represents the length of cable $i$. Below are some technical details:

- The number of cables `m` is verb@u.length@ (which is guaranteed to have the same length as `v`'s and `l`'s).
- In all of our test inputs, we promise that $1 \le u[i], v[i] \le$ `n`.
- Cables are bidirectional. For any pair of routers, there can be at most one cable between them—i.e., if `u[i] = p`, `v[i] = q`, there won't be another cable with the same endpoints `p` and `q` again.

**Example:** Consider the following input:

```
n = 6
u = [1,4,5,6,5,1,2]
v = [4,5,6,3,2,2,3]
l = [10,7,4,3,1,5,2]
```

This is the same network as shown in the diagram above. We expect the answer to be 11.

### Expectations: Promises, Constraints, and Grading

- We promise `n` $\le 50,000$ and `m` $\le 50,000$ The length of cable $i$ is $1 \le l_i \le 1,000$
- The initial network is guaranteed to be connected.
- For full-credit, your code must run in $O(m \log m)$ time. Your code must finish within 2 seconds on each of the test cases.

## Problem 3: Minimum Operations (10 points)

Define the following two operations for a list of integers `l`:

- `shift(l)` returns a new list after moving the front number to the back—i.e., it returns `l[1:] + l[0]` in Python's list notation.
- `reverse(l)` returns the reverse of the `l`—i.e., it returns `l[::-1]` in Python's list notation.

Nand and Nor are psychic pairs. When Nand comes up with a list `S`, Nor comes up with a list `T` such that `T` can be obtained from `S` by applying zero or more of `shift` and `reverse` operations in some order. As an example, consider `S = {1, 2, 3, 4}` and `T = {2, 1, 4, 3}`, we can see that

```
T = shift(shift(reverse(S)))
```

But this is not the only way to transform `S` to `T`. To illustrate, here are some other sequence of operations:

```
T = reverse(shift(shift(S)))
T = shift(reverse(shift(S)))
T = reverse(shift(shift(reverse(reverse(S)))))
```

Our goal in this problem is to find the smallest number of operations to achieve this transformation.

**Your Task:** Inside the class `MinOps`, write a public method

```
public int minimumOps(List<Integer> S, List<Integer> T)
```

that takes as input two lists `S` and `T` and returns **the smallest number of operations** to transform `S` into `T`.

**Examples:** Here are some examples on lists of length 5:

```
minimumOps(List.of(1,2,3,4,5), List.of(2,1,5,4,3)) == 3
minimumOps(List.of(5,4,3,2,1), List.of(1,5,4,3,2)) == 3
minimumOps(List.of(1,2,3,4,5), List.of(5,4,3,2,1)) == 1
```

### Expectations: Promises, Constraints, and Grading

- We promise that `S` and `T` will always have the same length $n$. You can expect $1 \leq n \leq 500$.
- We also promise that it is always possible to transform `S` into `T` in this way. Moreover, the input lists will have distinct numbers (no repeats).
- For full-credit, your code must finish within 1 second on each of the test cases.
- If you want to put array lists into a hash set (i.e., `HashSet<ArrayList<Integer>>`, you should know that this combination works well and works as you expect them to. This will be useful for, for example, checking if you have seen a particular array list before.

## Problem 4: All Aliases (10 points)

One real-life person usually has multiple online identities. In this problem, you will help an online platform identify all "aliases" of each individual. More specifically, on this platform, an online identity is given by an `Account` class, which has the following attributes:

```
private int id;
private String displayName;
private String email;
private String phoneNo;
```

The variables are private, but they can be accessed via their corresponding getter methods (e.g, `.getPhoneNo`). Moreover, we promise that **the account `id`s are all unique**—i.e., there will *not* be two different instances of `Account` in the input with the same `id`.

**Detective Work:** The company's research department has found the following:

- *Direct Knowledge.* Two accounts belong the same person if they have exactly the same `email` or `phoneNo`—or both.
- *Transitive Knowledge.* If accounts $A$ and $B$ belong to the same person, and accounts $B$ and $C$ belong to the same person, we know for fact that $A$ and $C$ belong to the same person. Hence, if we model direct knowledge as a graph, then two accounts belong to the same person when there is a path between them.

In the context of this company, the *main account* of a person is that person's account with the smallest `id`. You will write a program to find for each account the id of the main account that belongs to this person.

**Your Task:** Inside the class `AllAliases`, write a public method

```
public HashMap<Integer, Integer> findAllAliases(Account[] accounts)
```

that takes as input an array `Account` objects and returns a `HashMap` with the following properties:

- For each account `a` in the input array `accounts`, the id of this account (i.e., `a.getId()`) appears as a key in the returned `HashMap`, and
- the value corresponding to this key is the `id` of the main account that belongs to this person.
- No other keys are present in this map.

**Example:** An example will help illustrate the problem. Consider the following array of accounts:

```
Account[] accounts = {
     new Account(5, "Gift", "giftfy@gmail.com", "1234567777"),
     new Account(11, "Ike", "hike@gmail.com", "024298123"),
     new Account(72, "C++God", "omgsan@outlook.com", "0347127211"),
     new Account(3, "2Gift-too", "u998877@student.mahidol.edu", "1234567777"),
     new Account(1, "PlanetPython", "giftfy@gmail.com", "0816654433"),
     new Account(17, "4Ike", "hike@gmail.com", "1112"),
     new Account(14, "Kite", "omgsan@outlook.com", "1112")
};
```

It is clear from inspecting this array that the following pairs of accounts (identified by their id's) belong to the same person (using direct knowledge):

```
5 -- 1 (via email)
5 -- 3 (via phoneNo)
72 -- 14 (via email)
11 -- 17 (via email)
14 -- 17 (via phoneNo)
```

Hence, we know that 1, 3, 5 belong to the same person and per above, the main account is 1. We also know that 17, 72, 11, 14 belong to the same person, and per above, the main account here is 11. This leads to the following `HashMap` output (note: the relative ordering doesn't matter, just like in Python's dict):

```
{1=1, 17=11, 3=1, 5=1, 72=11, 11=11, 14=11}
```

### Expectations: Promises, Constraints, and Grading

- The number of accounts in the input array $n$ never exceeds $500,000$.
- None of the string attributes are longer than 100 characters in length.
- For full-credit, your code must run in $O(n \log n)$ time. Your code must finish within 5 seconds on each of the test cases.
- Do **NOT** modify `Account.java`. In our grading, this file will be overwritten with our original code.