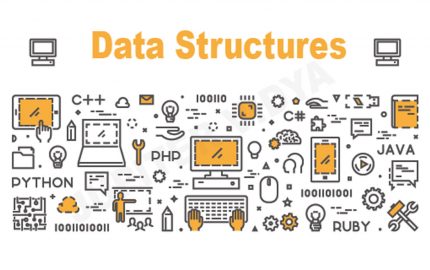
**Finding Report- Oct 16th 2018**

**Incorrect implementations of data structures.**

A *data structure* is a mechanism for representing data in your programs. Many of you have heard terms like linked list, tree, and array. Each of these are logical representations of data that correspond to some architectural structure of what you're trying to represent.

One of the most common mistakes I’ve found programmers do both experienced coders and newbies alike is paying too little attention to data structure choice. Since almost all your code builds on your choice of data representation method, choosing the wrong data structure can have costly implications down the line.

Figure 1.1, jupitervidya.com, Jupiter vidya

Here's one example that illustrates this sort of design error: choosing a simple stack or queue, instead of a circular queue. Think of a stack as a stack of dishes. You put the bottom dish down, then another dish on top, then another, and so forth.

If you want to remove a dish, you take it from the top of the stack. This is called *last-in, first-out.* The problem is, if you need to remove something earlier in the stack, it's a hassle. Let's say you have ten dishes in the stack. To get to the first one, you have to remove all the others first.

Now, let's think of a queue. When you stand in line at the bank, you're in a queue. The first person in is also the first person out. As soon as the first person is served, the next person is up, and that person is served. The other thing that happens is that each person takes a step forward, moving up in the queue.

What happens when too many people show up? They're either turned away or the line goes out the door. And when the first person is called, all these people have to move.

When you have a lot of data, a queue of this sort can be enormously inefficient. Each time data is pulled from the beginning of the queue, all the data needs to move. We're in a big data world, where we have a constant flow of data through our systems.

In this context, it might be better to implement a circular queue. In this case, the data never moves. Instead, a pointer is set up to point to the beginning and end of the queue and, internally, the queue wraps around itself, so that the data is organized in a ring instead of a line. When a data element is used and eliminated from the ring, there's no need to move all the data in the ring. All that happens is the first element pointer points to a new element in the ring.

This is but one of many examples of how the choice of correct data structure can have enormous implications on the efficiency and effectiveness of your code.

**Solutions**

1. Employ Testing frameworks with respected to the development platform you are using.
2. Research thoroughly about a multiple data structures, do sample tests and use the data to accurately pinpoint possible candidates.
3. Do not ignore compiler warnings/errors. Look into them specially if it’s with regard to the data structure.
4. Expose your implementations for regular code review.
5. Write pseudo code before implementing a data structure to help plan the implementation properly.
6. Assess the efficiency of the algorithm with respect to the data structure using Big O.
7. Write unit and integration tests before deployment or finalizing the code base for production.
8. Attending coding meetings to learn more about different ways a data structure could be implemented.

**Created by the 16th of October 2018**

**Updated on 17th of October 2018**

**Finalized on \_\_ of October 2018**