# Project Progress Report

## Introduction

My project will be implementing skip-list, which is a data structure similar to that of a linked list. While a linked list keeps track of its next, and occasionally its previous node, a skip-list has different layers, where each node points to the next node, the node above, and the node below. In other words, a skip-list is a collection of linked lists stacked on top of each other. Each list will have several pointers, and as we climb its layers, our number of pointers will decrease. The bottom layer will be an implementation of an ordinary linked list, where our node points to the immediate next value, while the remaining layers have its node pointing to a value further in the list. Using this data structure over a linked list gives us the ability to search, add, and delete items in $\mathcal{O}(\log n)$ time for average cases.

A collection, conceptionally, is a group of similar things. One good example would be a game collection – which is the grouping of games a person might possess. We can apply this abstract definition to any other types of groups, such as the grouping of classes in Java. Architecturally, a collection in Java is the grouping of specific data types and their functionalities (in other words Objects). Oracle, through Java, provides its users with a Collections library which has three interfaces, all of which have different types of data structures. One of these interfaces is the List interface, which we'll used to implement our skip-list. In order for our data structure to gain membership, our skip-list will implement the methods contract by the List interface. In doing so, we'll be extending the Java Collection library.

Understanding how collections work in Java allows us to make a flexible skip-list that works with different data types. The concepts of generics extend this flexibility, by allowing our data structure to accept different subtypes of objects as parameters. Instead of being limited to work with primitive types such as int and char, making our list a generic data type will allow the user to store and retrieve Java Objects in our skip-list (for example Integers). Generics also

allows us to write dynamic methods that produce consistent results regardless of the input type. In other words, when we use generics, we do not need to write similar methods with different signatures to take into consideration all possible data types the user may use.

Considering all the above, my project will be building a skip-list that extends the Java Collection library and implements the List interface. The underlying implementation will be similar to a linked list; however, we'll extends our list to have pointers above and below each node. Doing so will improve the time complexity for searching, adding, and deleting node from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$ for average cases. Instead of limiting our skip-list to accepting one data type, we'll implement it as a generic list for it to receive any subtype of the Object class, producing and returning the same type.

## Task Breakdown: Goals and Challenges

The easiest way to start this project is by implementing the item-wrapper class. My plan is to first start by producing the relevant methods required to compare our values, making sure that the items of our list are inserted in a sorted order. Each instance of our wrapper class will have a pointer that will link to a node above, below, and next to it. Our first implementation will be more of a doubly-linked list. Before tackling the SortedSet portion of the project, I will deal with the methods within the iterator class, which will allow us to traverse the list.

I expect the SortedSet portion of the implementation to be the most difficult to write. Methods such as add and remove will require me to take into consideration the height of the list and how to manipulate it. I haven't made any progress writing the methods for this part of the project, but hopefully by mid-July I should be able to get started on it.