# EECS 2011: Assignment 1

Due: February 10, 2021, 23:00 EST
May be done in groups of up to three students (from either of the sections M and Z)

## Motivation

The purpose of this assignment is to compare two different variations of linked lists in terms of time complexity for some of the operations.

## Introduction

Like the previous assignment, this assignment will involve (partially) implementing a `List`[1] interface. The `List` interface provides four methods for positional (indexed) **access** to list elements, two methods to **search** for a specified object, and two methods to **insert** and **remove** multiple elements at an arbitrary point in the list.
While some of these operations often execute in constant amount of time, the others may execute in time proportional to the index value, depending on an implementation (the `LinkedList` class, for example, is not very efficient when accessing elements in the middle of the list using an index, and `ArrayList` does not allow for efficient insertion or deletion, unless the element is at the end of the list).
*Skip list*[2] is a data structure that allows *O(log n)* search or indexed access complexity as well as *O(log n)* insertion complexity within an ordered sequence of *n* elements. While there are other data structures allowing for the same complexity, skip lists do have some advantages over them (e.g., insertions tend to affect nodes only in the immediate vicinity, unlike in balanced trees).

## Implementation

In this assignment, you will have to write a skip list-based partial implementation of the `List` interface. Note that there are two main modes in which skip list data structure implementations are used: a map mode, where a <key, data> pair is inserted, and a linear ranked list mode – similar to how one uses an array (Figure 1). The latter is the one applicable to this assignment.

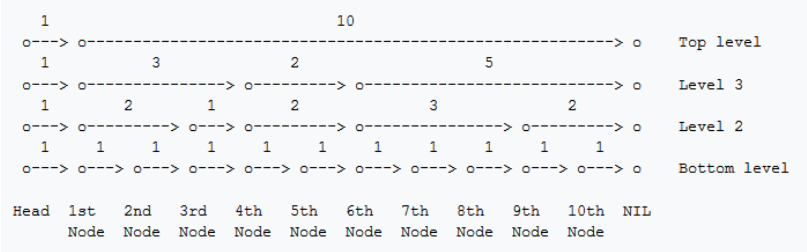


*Figure 1: An Indexable Skip List (Wikipedia)*

[1] https://docs.oracle.com/javase/8/docs/api/java/util/List.html
[2] https://en.wikipedia.org/wiki/Skip_list

You might find the following resources useful:
Indexable skip list
https://en.wikipedia.org/wiki/Skip_list#Indexable_skiplist

A Skip List Cookbook (a technical report by an author of skip lists, contains pseudocode for many operations)
http://cg.scs.carleton.ca/~morin/teaching/5408/refs/p90b.pdf

LinkedList implementation (to illustrate how a linked list is implemented in Java)
http://hg.openjdk.java.net/jdk7/jdk7/jdk/file/00cd9dc3c2b5/src/share/classes/java/util/LinkedList.java

## *Part 1*

Implement the following public methods in your implementation of the `List` interface, called `SkipList`:

```
1. boolean     add(E e)
2. void        add(int index, E element)
3. E           remove(int index)
4. E           get (int index)
5. int         size()
6. void        clear()
7. String      toString() (see Java API: AbstractCollection)
```

One public constructor should exist in your implementation: one that takes no parameters and creates an empty list when the class is instantiated.
The class should use generics. The behaviour of the methods in your implementation should be *equivalent* to that of Java Standard Library's classes (e.g., `ArrayList`; please refer to the class API online). For the remaining methods of the interface, you may just throw an exception

```
public type someUnneededMethod() {
      throw new UnsupportedOperationException();
}
```

Of course, you are free to implement any private or protected methods and classes as you see fit. However, you may not have any public methods other than the ones mentioned (or the ones present in the interface or class' superclass).

## *Part 2*

Nothing needs to be submitted for this part.
Create some tester class and use it with your list implementation to investigate its running time complexity as the number of items in the list increases. Like before, you could use values from 10 to 1,000,000 while measuring the running time. Confirm that the running time follows the expected logarithmic behaviour for both `add` and `get` methods.

NOTES:
1. Do not use *package*-s in your project (put your classes in the `default` package). Using packages will cost you a 10 % deduction from the assignment mark.

2. Some aspects of your code will be marked automatically (e.g., how it handles boundary cases and error conditions), using a custom tester class. It is also imperative you test your classes. If any of the java files that you submit do not compile, the whole submission will be given a grade of zero, regardless of how trivial the compiler error is. The code you submit should compile using

```
javac *.java
```

command in either Windows, Linux, or MacOS.

3. Your code should include Javadoc comments.

4. Using any `java.util` implementations of `Collection`[3] or `Map`[4] is not allowed (importing the `java.util.List` interface is of course fine).

## Submission

Find all the `java` files in your project and submit them electronically via eClass (no zipping is required). Only one file is expected, but you may write other classes, if you deem them necessary.
If working in a group, make only one submission per group and include a `group.txt` file containing the names and the student numbers of the group members. The deadline is firm. Contact the instructor *in advance* if you cannot meet the deadline explaining your circumstances.

## Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[5]. We look at whether the code passes the unit tests, satisfies the requirements of this document, and whether it conforms to the code style rules.

## Academic Honesty

Academic honesty will be strictly enforced in this course. Specifically, direct collaboration (e.g., sharing code or answers) is not permitted, and plagiarism detection software will be employed. You are, however, allowed to discuss the questions, ideas, or approaches you take.
Cite all sources you use (online sources – including web sites, old solutions, books, etc.) in your code comments; using textbook examples is allowed, but these must be cited as well.
E.g.,

```
1) /**
     * Constructs an empty list with the specified initial capacity.
```

[3] https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html
[4] https://docs.oracle.com/javase/8/docs/api/java/util/Map.html
[5] https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/

```
     *
     * @param   initialCapacity   the initial capacity of the list
     * @exception IllegalArgumentException if the specified initial
capacity
     *            is negative
     * uses code from www.xxx.yyy.edu/123.html for initialization
     *
     */

2) //formula based on Jane Smart's thesis, page XX, available at
https://...
       a = b + c;
```

Although using outside sources is allowed – with proper citing, if the amount of non-original work is excessive, your grade may be reduced. You might find the following article useful to have an idea what kind of collaboration is appropriate: https://en.wikipedia.org/wiki/Clean_room_design

**You may not post the contents of this assignment, nor solicit solutions on any external resources.**