Programming Assignment #1

Tests due: 9/14/25 @ 11:59PM
Implementation due: 9/21/25 @ 11:59PM

Assignment Link: https://classroom.github.com/a/7fbSEi8U

Please read through the entire writeup before beginning the programming assignment

Objectives

1. Implement a data structure based on linked lists
    a. Gain a better understanding of constant vs linear runtime
    b. See an example of a tactical optimization that improves constant factors
    c. Be able to compare access-by-reference vs access-by-index

Useful Links

1. The Java API
    a. Comparable
    b. Optional
    c. Iterator
    d. Iterable
    e. Exceptions
2. Testing with JUnit
    a. Assertions

Submission Process, Late Policy and Grading

Testing due date: 9/14/25 @ 11:59PM
Implementation due date: 9/21/25 @ 11:59PM
Total points: 30 (5 for testing + 20 for correctness + 5 for runtime)

The project grade is the grade assigned to the latest (most recent) submission made to Autolab (or 0 if no submissions are made). Autolab will pull your submission from your GitHub repository, so you must make sure that any changes you want to be included in your grade have been **committed and pushed.**

- If your submission is made before the deadline, you will be awarded 100% of the points your project earns.
- If your submission is made up to 24 hours after the deadline, you will be awarded 75% of the points your project earns.
- If your submission is more than 24 hours after the deadline, but within 48 hours of the deadline, you will be awarded 50% of the points your project earns.
- If your submission is made more than 48 hours after the deadline, it will not be accepted.

You will have the ability to use three grace days throughout the semester, and at most two per assignment (since submissions are not accepted after two days). Using a grace day will negate the 25% penalty per day, but will not allow you to submit more than two days late. Please plan accordingly. You will not be able to recover a grace day if you decide to work late and your score is not sufficiently higher. Grace days are automatically applied to the first instances of late submissions, and are non-refundable. For example, if an assignment is due on a Sunday and you make a submission on Monday, you will automatically use a grace day, regardless of whether you perform better or not. Be sure to test your code before submitting, especially with late submissions in order to avoid wasting grace days.

Keep track of the time if you are working up until the deadline. Submissions become late after the set deadline. Keep in mind that submissions will close 48 hours after the original deadline and you will not be able to submit your code after that time.

Note: No late submissions will be accepted for the testing portion of the assignment, and no grace days can be used on the testing portion of the assignment.

Setup

In order to complete this project, you must have completed PA0. If you are working on a machine other than the one you used in PA0, you must at least complete steps 2 and 4 in order to get IntelliJ and GitHub working properly.

Once you have ensured your development environment is setup as in PA0, you can accept the PA1 assignment in GitHub Classroom (here), and create a new IntelliJ project from VCS with your newly created repository.

Instructions

In this PA you will be filling in the body of 11 functions to complete the implementation of a sorted linked list data structure. Additionally, you will be required to write tests which can verify the correctness of valid implementations, and detect bugs in incorrect implementations.

Hint: Yes…there are 11 functions to write, but if you think carefully, you will realize that only ~4 of them require detailed implementation, and the others can be largely implemented by calling those 4 functions with only slight modification. **Plan out your implementation before coding.**

The linked list you will be implementing in this PA differs from the vanilla Linked Lists discussed in class so far in 3 distinct ways:

- **Ordering:** The data in the linked list must be stored in sorted order. This affects how items are inserted, as well as how items are searched for.
- **Hints:** Many of the functions have versions which take a node in the list as a hint. This hint provides those functions with a (hopefully) better location to start their search from.
- **Duplicates:** Rather than having a distinct linked list node for every insertion into the list, linked list nodes now store a count in addition to the value. If there are multiple insertions of the same value into the list, the count of the associated linked list node is incremented instead of creating a new node.

After you complete your tests, make sure to commit **and push** your work to GitHub, and submit to the PA1 testing submission in Autolab. After completing your implementation, make sure to commit **and push** your work to GitHub, and submit to the PA1 implementation submission in Autolab. **You should only make changes to `SortedListTests.java` and `SortedList.java`.**

Hint: It is advised that you commit and push frequently rather than waiting until you've completed everything.

Hint: Although you will get feedback from Autolab about correctness of your solutions, you should get in the habit of testing locally, and adding test cases as needed. This will be a more effective/efficient means of development, and will also give you a better understanding of the content of this programming assignment in the process.

0. Testing Phase (due 9/14/25)

For this part, modify the file `SortedListTests.java` to include new test cases.

Your test cases will first be run against a correct implementation of `SortedList`. If your tests fail the correct implementation you will receive 0 points for the testing phase.

Your test cases will then be run against several broken implementations of `SortedList`. You will get points for each broken implementation that at least one of your tests fail.

1. Define the search functions

Implement the following functions related to searching for a particular value in our linked list:

`Optional<LinkedListNode<T>> findRefBefore(T elem)`

If the list contains **`elem`**, then this function should return a reference to that list node.

If the list does not contain **`elem`**, then this function should return a reference to the node that would immediately come before a node with value **`elem`** if it were to be inserted into the list.

If **`elem`** is smaller than every element in the list (ie the **`elem`** would be inserted at the head of the list) then this function should return **`Optional.empty()`**.

For a list of length ***n***, this function should run in ***O*(*n*).**

`Optional<LinkedListNode<T>> findRefBefore(T elem, LinkedListNode<T> hint)`

The return value of this function should be the same as the return value of **`findRefBefore(T elem)`**.

The second argument to this function, **`hint`**, is a reference to a node in the list. The search for the node containing **`elem`** should begin at **`hint`** rather than the head of the list. The behavior of this function is undefined if **`hint`** is not in the list.

If **`hint`** is at position ***i*** and the node with value **`elem`** is at position ***j***, then this function should run in ***O*(|*i* - *j*|)**.

`Optional<LinkedListNode<T>> findRef(T elem)`

If the list contains **`elem`**, then this function should return a reference to that list node.

Otherwise this function should return **`Optional.empty()`**.

For a list of length ***n***, this function should run in ***O*(*n*).**

`Optional<LinkedListNode<T>> findRef(T elem, LinkedListNode<T> hint)`

The return value of this function should be the same as the return value of **`findRef(T elem)`**.

The second argument to this function, **`hint`**, is a reference to a node in the list. The search for the node containing **`elem`** should begin at **`hint`** rather than the head of the list. The behavior of this function is undefined if **`hint`** is not in the list.

If **`hint`** is at position ***i*** and the node with value **`elem`** is at position ***j***, then this function should run in ***O*(|*i* - *j*|)**.

Hint: *Only one of these functions really needs a detailed implementation if you then use that function as the basis for defining the other 3.*

2. Define the indexing functions

Implement the following functions related to getting an element at a specific position in the list:

`LinkedListNode<T> getRef(int idx)`

Return a reference to the node containing the element at the provided index. Note that the index provided is with respect to the elements that have been inserted, not the actual nodes of the list since there may be fewer nodes than elements.

Throw an **`IndexOutOfBoundsException`** if the provided index is invalid (i.e., **`idx < 0`** or **`idx >= length`**).

For a list of length ***n***, this function should run in ***O*(*n*).**

`T get(int idx)`

Return the value at the provided index. Note that the index provided is with respect to the elements that have been inserted, not the actual nodes of the list since there may be fewer nodes than elements.

Throw an **`IndexOutOfBoundsException`** if the provided index is invalid (i.e., **`idx < 0`** or **`idx >= length`**).

For a list of length ***n***, this function should run in ***O*(*n*).**

Hint: *Only one of these functions really needs a detailed implementation if you then use that function as the basis for defining the other one.*

3. Define the insertion functions

Implement the following functions related to inserting an element into the list:

`LinkedListNode<T> insert(T elem)`

Insert **`elem`** into the linked list and return the linked list node where it was inserted.

The element should be placed so that the list remains in sorted order. After the insertion, the returned node's **`next`** method should return a reference to the next greatest node, and the **`prev`** method should return a reference to the next least node.

If **`elem`** is already in the list, no new node should be created. Instead, increment the count of the existing node.

For a list of length ***n***, this function should run in ***O*(*n*).**

`LinkedListNode<T> insert(T elem, LinkedListNode<T> hint)`

Insert **`elem`** into the linked list and return the linked list node where it was inserted.

This function should behave exactly like **`insert(T elem)`** but with the search for the insertion point beginning at **`hint`** rather than the head of the list. The behavior of this function is undefined if **`hint`** is not in the list.

If **`hint`** is at position ***i*** and the node with value **`elem`** should be at position ***j***, then this function should run in ***O*(|*i* - *j*|)**.

Hint: *Much of the logic for insertion can rely on previously defined functions.*

4. Define the removal functions

Implement the following functions related to removing an element from the list:

`T remove(LinkedListNode<T> ref)`

Remove a single instance of the element held by the node referenced by **`ref`** from the list, and return its value. The behavior of this function is undefined if **`ref`** is not in the list.

If this is the last element of its value, then the node referenced by **`ref`** should also be removed from the linked list.

This function should run in ***O*(1).**

`T removeN(LinkedListNode<T> ref, int n)`

Remove **`n`** instances of the element held by the node referenced by **`ref`** from the list, and return its value. The behavior of this function is undefined if **`ref`** is not in the list.

If after removing **`n`** elements, there are no more elements of the referenced value, then the node referenced by **`ref`** should also be removed from the linked list.

Throw an **`IllegalArgumentException`** if **`n`** is larger than the number of elements held by **`ref`**.

This function should run in ***O*(1).**

`T removeAll(LinkedListNode<T> ref)`

Remove all instances of the element held by the node referenced by **`ref`** from the list, return its value, and remove the node from the linked list. The behavior of this function is undefined if **`ref`** is not in the list.

This function should run in ***O*(1).**

Hint: *Only one of these functions really needs a detailed implementation if you then use that function as the basis for defining the other 2.*

Additional Notes

Doubly Linked Lists

A doubly linked list is a collection data structure that stores each of its elements wrapped in a linked list node (**`LinkedListNode`** for this project).

A doubly linked list contains an optional reference to the first node (**`headNode`**) in the list and an optional reference to the last node (**`lastNode`**) in the list. Each node contains the value it stores, and references to the next and previous nodes (if they exist…see the section on **`Optional`** types in Java). In this assignment, each node also contains a count that defines how many times the associated value has been inserted into the list.

To insert an element into a doubly-linked list, all of the relevant references need to be updated. **Make sure you handle all of the corner cases!**

1. **`newNode.next`** and **`newNode.prev`** need to be set appropriately
2. **`newNode.prev.next`** needs to be updated if it exists
3. **`newNode.next.prev`** needs to be updated if it exists
4. **`headNode`** needs to be updated if newNode is inserted at the front
5. **`lastNode`** needs to be updated if newNode is inserted at the end
6. **`SortedList.length`** and **`LinkedListNode.count`** should be updated

Ordering in Java

Elements stored in **`SortedList`** must extend the **`Comparable`** interface in Java. This interface defines a method: **`int compareTo(T o)`** that behaves as follows:

- **`a.compareTo(b) < 0`** if a < b
- **`a.compareTo(b) == 0`** if a = b
- **`a.compareTo(b) > 0`** if a > b

Operations that insert elements into the list should use **`compareTo`** to ensure that the new element is placed at the right location in the list. For example, if you want to check if **`value1`** is less than **`value2`**, check if **`value1.compareTo(value2) < 0`** instead of **`value1 < value2`**.

Optional Objects in Java

Because in general there is no guarantee that a **`LinkedListNode`** will have a next or previous node, we cannot store references to **`LinkedListNodes`** directly. Similarly, for many of the **`SortedList`** functions, there is no guarantee that a given node will be found, so we cannot return a reference to a **`LinkedListNode`** directly.

In both cases, we use the **`Optional`** class in Java. The **`Optional<T>`** class has two static methods for creating objects, **`Optional<T> of(T value)`** and **`Optional<T> empty()`**.
If for example, we have a function that may or may not return an **`int`**, then in Java we would declare the return type of that function to be **`Optional<int>`**. To return an integer, ie 2, our function would return **`Optional.of(2)`**. To return nothing our function would return **`Optional.empty()`**.

To interact with **`Optional`** objects, the **`Optional<T>`** class has two useful methods: **`isPresent()`**, and **`get()`**. The **`isPresent()`** method returns true if the **`Optional`** object actually holds a value, and returns false otherwise. The **`get()`** method returns the value held by the **`Optional`** object, and should generally only be called after checking that **`isPresent()`** returns true.

More information on **`Optional`** in Java can be found here.

Other Tips

1. It is important to understand that the mental model we have of our collection, and its actual implementation are different. For example, if the linked list in this assignment has values 1, 1, 1, 2, 3 inserted into it, the sequence it represents is 1, 1, 1, 2, 3. There are 5 elements in that order. The actual implementation of our linked list however would only have 3 actual linked list nodes.
2. The linked list implementation already has the update and iterator methods defined for you. Feel free to reference them as needed. We also have provided a main function and some sample tests. Familiarize yourself with these to see examples of how the **`SortedList`** class may be used. Run them!
3. When writing tests, it is useful to start with tests that test one function at a time, before moving on to tests that test the interaction between multiple functions. If you test multiple functions in the same test without being sure they are individually correct, then if the test fails it is hard to know where to begin. For example, if you are trying to test the remove function, but you create the linked list that you will test it on by calling insert, then if the test fails you don't actually know if the remove call was incorrect, or the insert call was incorrect. Building your example lists manually (by explicitly creating the nodes and setting the next/prev refs by hand) you can have more focused testing.
4. For testing, try to think about what corner cases or oddities would trip you up when implementing the list yourself. Think about details which could be easily overlooked. Then write tests which check for these corner cases and details. In order to do this, you must have a good understanding of the problem you are trying to solve. **Make sure you understand exactly how the linked list in this assignment is supposed to behave before writing tests.** Draw out examples on paper, and ask clarifying questions as needed.

Academic Integrity

As a gentle reminder, please re-read the academic integrity policy of the course. I will continue to remind you throughout the semester and hope to avoid any incidents.

What Constitutes a Violation of Academic Integrity?

These bullets should be obvious things not to do (but commonly occur):

- Turning in your friend's code/write-up (obvious).
- Turning in work generated by ChatGPT or another AI tool (obvious)
- Turning in solutions you found on Google with all the variable names changed (should be obvious). This is a copyright violation, in addition to an AI violation.
- Turning in solutions you found on Google with all the variable names changed and 2 lines added (should be obvious). This is also a copyright violation.
- Paying someone to do your work. You may as well not submit the work since you will fail the exams and the course.
- Posting to forums asking someone to solve the problem.

Note: Aggregating every [stack overflow answer|result from google|other source] because you "understand it" will likely result in full credit on assignments (if you aren't caught) and then failure on every exam. Exams don't test if you know how to use Google, but rather test your understanding (i.e., can you understand the problems to arrive at a solution on your own). Also, other students are likely doing the same thing and then you will be wondering why 10 people that you don't know have your solution.

Other violations that may not be as obvious:

- Working with a tutor who solves the assignment with you. If you have a tutor, please contact me so that I may discuss with them what help is allowed.
- Sending your code to a friend to help them. If another student uses/submits your code, you are also liable and will be punished.
- Joining a chatroom for the course where someone posts their code once they finish, with the honor code that everyone needs to change it in order to use it.
- Reading your friend's code the night before it is due because you just need one more line to get everything working. It will most likely influence you directly or subconsciously to solve the problem identically, and your friend will also end up in trouble.

What Collaboration is Allowed?

Assignments in this course should be solved individually with only assistance from course staff and allowed resources. You may discuss and help one another with technical issues, such as how to get your compiler running, etc.

There is a gray area when it comes to discussing the problems with your peers and I do encourage you to work with one another to solve problems. That is the best way to learn and overcome obstacles. At the same time you need to be sure you do not overstep and not plagiarize. Talking out how you eventually reached the solution from a high level is okay:

> "I used a stack to store the data and then looked for the value to return."

but explaining every step in detail/pseudocode is not okay:

> "I copied the file tutorial into my code at the start of the function, then created a stack and pushed all of the data onto the stack, and finished by popping the elements until the value is found and use a return statement."

The first example is OK but the second is basically a summary of your code and is not acceptable, and remember that you shouldn't be showing any code at all for how to do any of it. Regardless of where you are working, you must always follow this rule: Never come away from discussions with your peers with any written work, either typed or photographed, and especially do not share or allow viewing of your written code.

What Resources are Allowed?

With all of this said, please feel free to use any [files|examples|tutorials] that we provide directly in your code (with proper attribution). Feel free to directly use anything from lectures or recitations. You will never be penalized for doing so, but should always provide attribution/citation for where you retrieved code from. Just remember, if you are citing an algorithm that is not provided by us, then you are probably overstepping.

More explicitly, you may use any of the following resources (with proper citation/attribution):

- Any example files posted on the course webpage (from lecture or recitation).
- Any code that the instructor provides.
- Any code that the TAs provide.
- Any code from the Java API (https://docs.oracle.com/javase/8/docs/api/)

Omitting citation/attribution will result in an AI violation (and lawsuits later in life at your job). This is true even if you are using resources provided.

Amnesty Policy

We understand that students are under a lot of pressure and people make mistakes. If you have concerns that you may have violated academic integrity on a particular assignment, and would like to withdraw the assignment, you may do so by sending us an email **BEFORE THE VIOLATION IS DISCOVERED BY US**. The email should take the following format:

Dear Dr. Mikida and Dr. Bosse,

I wish to inform you that on assignment X, the work I submitted was not entirely my own. I would like to withdraw my submission from consideration to preserve academic integrity.

J.Q. Student
Person #12345678
UBIT: jqstuden

When we receive this email, student J would receive a 0 on assignment X, but would not receive an F for the course, and would not be reported to the office of academic integrity.