COMP 2402 Winter 2022
Assignment 3

Due: Thursday March 10 2:00pm
Topic focus: Lec 9 - 13
lates accepted within 24 hours
submit early and often

Academic Integrity

You may:

- Discuss general approaches with course staff,
- Discuss general approaches with your classmates,
- Use code and/or ideas from the textbook,
- Use a search engine / the internet to look up basic Java syntax,
- Send your code / screen share your code with course staff (although it is unlikely the course staff has the bandwidth to look at individuals' code, unfortunately.)

You may **not**:

- Send or otherwise share code or code snippets with classmates,
- Use code not written by you, unless it is code from the textbook (and you should cite it in comments),
- Use a search engine / the internet to look up approaches to the assignment,
- Use code from previous iterations of the course, unless it was solely written by you,
- Use the internet to find source code or videos that give solutions to the assignment.

If you ever have any questions about what is or is not allowable regarding academic integrity, **please do not hesitate to reach out to course staff**. We will be happy to answer. Sometimes it is difficult to determine the exact line, but if you cross it the punishment is severe and out of our hands. **Any student caught violating academic integrity, whether intentionally or not, will be reported to the Associate Dean** and be penalized as follows:

- First offence: F in the course.
- Second offence: One-year suspension from program.
- Third offence: Expulsion from the University.

These are standard penalties. More-severe penalties will be applied in cases of egregious offences. For more information, please see Carleton University's Academic Integrity Policy.

Coding Environment Setup

If you want suggestions on how to set up your coding environment, please watch this video or read the "Setting Up Your Programming Environment" post on piazza.

If you want Alexa's 20-minute "Assignment 1 setup" video, please go here. The setup for assignment 1 is almost identical to that of assignment 2 so there will not be a new video.

How to Get Help

Problem Solving & Assignment Tips Document

- We made this document just for you, full of tips to help you complete your programming assignments faster and better! We will add to it as the course progresses, and you are welcome to add your suggestions as well.

Piazza

- Good for getting a quick response, since it goes to a wider audience than email.
- Good for viewing questions that other students have. Chances are, if you have the question, so does someone else. If tagged correctly (e.g. "a2" for assignment 2), you can easily see all questions relating to a particular subject.
- Good for general questions that don't divulge your approach (e.g. "Am I allowed to use standard JCF data structures in my solution?" "Does anyone else get this strange "file not found" error when submitting?" etc.)
- Good for asking specific-to-you questions of the course staff, which you can ask "privately" on piazza. This gets you the benefits of a wider audience yet keeps your personal details private to course staff. (You can and should also contact Alexa this way.)

Student Hours

- Good for quick questions that may have some back and forth.
- Good for clarifications.
- Good for "tips and tricks".
- Not good for debugging sessions. The person holding the student hour may have many people waiting, and as such cannot spend 20 minutes helping you debug, unfortunately. They may give you a push in the right direction, then ask you to go back in line while you work with that push, while they help someone else.

Discord

- Good for light social interactions and commiseration.

- Please remember that this discord is an official class discord, and as such you should keep your behaviour “professional”. Please be respectful. Jokes are great, just not at the expense of an individual or a specific group. Disrespectful behaviour (intentional or not) will be met with our zero-tolerance policy (removal from server.)
- This is not a good place for asking questions, as the course staff will be spending most of their time monitoring piazza. Piazza has a better system for tracking open questions and answers, and so it is more time efficient for the course staff to spend their time there instead of discord.

Grading

This assignment will be tested and graded by a computer program (and **you can submit as many times as you like, your highest grade is recorded**). For this to work, there are some important rules you must follow:

- Keep the directory structure of the provided zip file. **If you find a file in the subdirectory comp2402w22a3 leave it there.**
- Keep the package structure of the provided zip file. **If you find a package comp2402w22a3; directive at the top of a file, leave it there.**
- Do not rename or change the visibility of any methods already present. If a method or class is public, leave it that way.
- Do not change the `main(String[])` method of any `PartX` class. These are setup to read command line arguments and/or open input and output files. Don't change this behaviour. Instead, learn how to use command-line arguments to do your own testing.
- **Submit early and reasonably often.** The submission server compiles and runs your code, and gives you a mark. You can submit once every 5 minutes and we keep track of your highest overall score. There is no excuse for submitting code that does not compile or does not pass tests. The 5 minute wait limit is to prevent you from using the server to debug, potentially overloading it. (You should debug locally! More on that later.)
- Write efficient code. The submission server places a limit on how much time it will spend executing your code, even on inputs with a million lines. For some questions it also places a limit on how much memory your code can use. If you choose and use your data structures correctly, your code should easily execute within the time limit. Choose the wrong data structure, or use it the wrong way, and your code will be too slow or use too much space for the submission server to work (resulting in a failure of that test).

Submitting and Server Tests

The submission server is here. If this is your first time submitting, you will need to get a secret key which you will then use to upload your assignment to the submission server. If you have issues, please post to piazza to the Instructors (or the class) and we’ll see if we can help.

When you submit your code, the server runs tests on your code. **These are bigger and better tests than the small number of tests provided in the "Local Tests" section further down this document**. They are obfuscated from you, because **you should try to find exhaustive tests of your own code**. This can be frustrating because you are still learning to think critically about your own code, to figure out where its weaknesses are. But have patience with yourself and make sure you read the question carefully to understand its edge cases.

Warning: Do not wait until the last minute to submit your assignment. There is a hard 5 second limit on the time each test has to complete, and 5 minutes between submissions. If the server is heavily loaded, borderline tests may start to fail. **You can submit multiple times and your best score is recorded** so there is no downside to submitting early.

The Assignment

Purpose & Goals

Generally, the assignments in this course are meant to give you the opportunity to practice with the topics of this course in a way that is challenging yet also manageable. At times you may struggle and at others it may seem more straight-forward; just remember to keep trying and practicing, and over time you will improve.

Specifically, assignment 3 aims to improve your understanding of the SortedSet and PriorityQueue interfaces by:

- Using the ArrayList, LinkedList, and SortedSet or PriorityQueue interfaces to solve a variety of problems (Part A),
- Implementing some new methods in the `IntBST`, `BinarySearchTree`, and `BinaryHeap` classes (Part B), and
- Reflecting on your choice of data structures and algorithms (Part C).

Note that you are welcome to use any data structure from previous assignments to complete current assignments.

Setup

(Alexa made a 20-minute Assignment 1 Setup video that may help you do the following steps; it is very similar to the setup for Assignment 1.)

Start by downloading and decompressing the Assignment 3 Zip File (comp2402w22a3.zip on piazza's Resources tab), which contains a skeleton of the code you need to write. You should have the files: Part0.java, Part1.java, Part2.java, Part3.java, Part4.java, IntBST.java, BinarySearchTree.java, BinaryHeap.java, BinaryTree.java, DefaultComparator.java and SSet.java.

The skeleton code in the zip file compiles fine. Here's what it looks like when you unzip, compile, and run `Part0` and `Part1` from the command line:

```
a3 % pwd
/Users/asharp/2402/a3
a3 % ls
comp2402w22a3.zip
a3 % unzip comp2402w22a3.zip
Archive:  comp2402w22a3.zip
   creating: comp2402w22a3/
  inflating: comp2402w22a3/IntBST.java
  inflating: comp2402w22a3/BinaryTree.java
  inflating: comp2402w22a3/Part1.java
  inflating: comp2402w22a3/Part0.java
  inflating: comp2402w22a3/BinarySearchTree.java
  inflating: comp2402w22a3/Part3.java
  inflating: comp2402w22a3/SSet.java
  inflating: comp2402w22a3/Part2.java
  inflating: comp2402w22a3/BinaryHeap.java
  inflating: comp2402w22a3/Part4.java
  inflating: comp2402w22a3/DefaultComparator.java
a3 % ls
comp2402w22a3        comp2402w22a3.zip
a3 % javac comp2402w22a3/*.java
a3 % ls
comp2402w22a3        comp2402w22a3.zip
a3 % java comp2402w22a3.Part0
3
50
4
13 [I hit Ctrl-D to end the input here]
3D [The D here is a remnant of the Ctrl-D]
50
4
13
Execution time: 5.986822196
a3 % java comp2402w22a3.Part1
Execution time: 2.2645E-5
```

You can also run the `IntBST`, `BinarySearchTree`, and `BinaryHeap` programs:

```
a3 % java comp2402w22a3.IntBST
<... output omitted >
```

```
a3 % java comp2402w22a3.BinarySearchTree
<... output omitted >
a3 % java comp2402w22a3.BinaryHeap
<... output omitted >
```

If you are having trouble running these programs, **figure this out first before attempting to do the assignment**. Check the assignment 3 FAQ on piazza and/or the aforementioned setup video, or ask a question there if you are stuck and course staff or another student will likely help you fairly quickly.

Part A: The Interface

The file `Part0.java` in the zip file actually does something. You can use its `doIt()` method as a starting point for your solutions. Compile it. Run it. Test out the various ways of inputting and outputting data. **You should not need to modify Part0.java, it is a template. Do not modify the main(String[]) methods for any PartX.java file, as they are expected to remain the same for the server tests.**

You can download some sample input and output files for each question as a zip file (a3-io.zip on piazza's Resources tab). If you find that a particular question is unclear, you can probably clarify it by looking at the sample files for that question. Part of problem solving is figuring out the exact problem you are trying to solve.

You may assume that all lines are integers under 32 bits. If you need some help with modular arithmetic, try this khan academy page. As with assignment 2, let's just use java's floorMod(x,d) to compute x mod d. This always returns a non-negative integer, which will simplify our lives.

1. [8 marks] Given a file where each line is a single (32-bit) integer, read in all lines of the file one at a time; take the sum (modulo 2402) of the chain of `x` elements obtained by starting at the index-0 element, then using that element as the index of the next element in the chain, proceeding until you have summed `x` elements. When you use an element as an index, you should modulo it by the length of the list. As usual, `x≥0` is a parameter to the `doIt` function. For example, if `x=3` and the input is

   5 ← index 0, first element of the chain. Leads us to index 5%6=5.
   4
   8
   10
   2
   6 ← index 5, second element of the chain. Leads us to index 6%6=0.

   Then the chain of length 3 consists of the elements at indices 0, 5, then 0 again, for a total sum of (5+6+5)%2402=16. If the last line had been 7 then the output would be (5+7+4)=16.

Hint: To get the full points for this question, you will need to optimize for certain inputs. So once you have a working / correct solution, think to yourself: for what inputs is the time and/or space I'm using overkill? In those situations, can I cut out early or use less space?

Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier.

2. [10 marks] (Assignment 3.1 Redux) Given a file where each line is a single (32-bit) integer, read in all lines of the file one at a time; construct the chain of `x` elements as you did in Part 1, but instead of summing each element `c` of the chain, sum the minimum element `d ≥ c` prior to `c` in the chain (`0` if no such `d` exists.). As with Part 1, your sum should be modulo 2402, and `x≥0` is a parameter to the `doIt` function. For example, if `x=3` and the input is

5	← index 0, first element of the chain. Leads us to index 5%6=5.
4	
8	
10	
2	
6	← index 5, second element of the chain. Leads us to index 6%6=0.

Then the chain of length 3 is 5, 6, 5. For the first 5, there is no element d ≥ 5 in the (empty) chain so far so it contributes nothing. For the 6, there is no element d ≥ 6 in the chain so far (5), so it contributes nothing. For the final 5, the 5 ≥ 5 and so our final sum is 5. If the last line had been a 1 then the output would be (5+5)=10 because 5 ≥1 and 5 ≥ 4 and the chain is 5, 1, 4.

Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier.

3. [6 marks] (Assignment 3.1 Redux) Given a file where each line is a single (32-bit) integer, read in all lines of the file one at a time; construct the chain of `x` elements as you did in Part 1, except that as you process element `c` at index `i`, you'll modify your list by inserting at index `(i+1)%l.size()` the element currently at list position `l.size()-1-i` (i.e. `i` positions from the end of the list.) As with Part 1, your sum should be modulo 2402, and `x≥0` is a parameter to the `doIt` function. For example, if `x=3` and the input is

8	← index i=0, first element of the chain. Insert 9 at index 1.
4	
7	
10	
2	
9	

As we process the first element of the chain (the 8 at index 0), we add 8 to our sum and also insert the element at index 9 to our list at index i+1=1. We now have the list

8
9 ← index i=8%7, second element of the chain. Insert 2 at index 2.
4
7
10
2
9

As we process the second element of the chain (the 9 at index 8%7), we add 9 to our sum and also insert the element at index (6-8)%7=5 (a 2) at index 1+1=2. This gives the modified list

8
9 ← index i=9%8, third element of the chain. Add 9 to our sum, done.
2
4
7
10
2
9

Thus our chain was the 3 elements 8 + 9 + 9 = 26.

Note: You first get the current value of the chain, then add to the list, then go to the next chain. The example above hopefully illustrates the difference if you were to, say, go to the next chain and then add to the list.

Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier. See the note just before Part 1's description.

4. [10 marks] (Assignment 3.2 Redux) Given a file where each line is a single (32-bit) integer, read in all lines of the file one at a time; construct the chain of `x` elements as you did in Part 2, but instead of summing each element of the chain, sum the minimum multiple of 5 in the chain so far (`0` if no such multiple exists.) There is one twist, which is that anytime your current chain element is divisible by 24, remove the minimum multiple of 5 that remains in the chain at this point in time (if there is a multiple of 5 remaining) (do this after you get the min multiple for the sum). As with Part 1, your sum should be modulo 2402, and `x≥0` is a parameter to the `doIt` function. For example, if `x=5` and the input is

   10 ← index i=0, 1st element of chain. nothing to add to sum, but 10 mult of 5.
   also 5th element of chain. add 10 to sum.
   20
   5 ← index i=2, 3rd element of chain. add 10 to sum. 5 mult of 5.
   3

2 ← index i=4, 2nd element of chain. Add 10 to sum.
24 ← index i=5, 4th element of chain. Add 5 to sum. 24 div by 24, remove min

then our output is 10+10+5+10=35.

Note: If your current chain elements happens to be both a multiple of 5 and a multiple of 24 then you should first do the retrieval of the minimum for your sum, then the removal of the minimum in the chain thus far, and then add the current multiple of 5.

Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier. See the note just before Part 1's description.

Part B: The Implementation

The (provided) `BinarySearchTree`, `IntBST`, and `BinaryHeap` classes are (incomplete) implementations of the `SortedSet` and `BinaryHeap` interfaces. Your task is to complete the implementation of the `IntBST`, `BinarySearchTree`, and `BinaryHeap` classes, as specified below. Default implementations are provided for the 4 methods so that you can compile and run the main methods of the relevant classes right out of the gate; these defaults will not pass the server tests, however.

There are some visualization tools out there that might help you write and/or debug your BST-related code: https://www.cs.usfca.edu/~galles/visualization/BST.html and https://people.ksp.sk/~kuko/gnarley-trees/BST.html.

5. [10 marks] (Assignment 1.6 & 2.5 Redux) In the `IntBST` class, implement
   `public void sum(IntBST other)`
   Replace the `i`th least element of `this IntBST` by (`i`th least element of this)+(`i`th least element of other). That is, the elements from `this` are now each value-shifted by the corresponding entry in `other`. If `other` is shorter than `this`, just repeat `other` as many times as is necessary until you have shifted every element of `this` once.

   For example, if the set is originally `s={4,6,8,10}` and `other={1,2,5}` then `s.sum(other)` changes `s` to `{5,8,11,13}`. (Note that now we're dealing with Sets and not Lists, so we cannot have duplicates and the order in which we write a set is not important.)

   Note: We've seen this problem on two assignments, so getting something correct is clearly not what my focus is here. You'll need to think about where/when your correct solution takes a long time, and try to do better in those situations.

6. [8 marks] (Assignment 2.7 & 2.9 Redux) In the `BinarySearchTree` class, implement

   ```
   public void insertSingleBlock(BinarySearchTree<T> other)
   ```

   which inserts the `BinarySearchTree other` at the appropriate location in `this BinarySearchTree`. There is one key property of `other`, which is that `other` contains elements within some gap from `this`, so it is guaranteed to insert as a single block in the sorted list.

   For example, if the set s is originally `s={10,20,30,40,50}` and `other={21,24}` then `s.insertSingleBlock(other)` changes `s` to `{10,20,21,24,30,40,50}`. An input `other={21,31}` would be invalid, for example, since `{21,31}`could not be inserted into s as a single block in sorted order (and maintain the sorted order.) You do not have to check for this property; you may assume that other has this property already.

7. [8 marks] In the `IntBST` class, implement the method

   ```
   public int numDescendantsMod(int y)
   ```

   without recursion. This method should return the number of nodes that have a value that is 0 mod y.

8. [12 marks] In the `BinaryHeap` class, implement

   ```
   public int smallest(int k)
   ```

   which returns the kth smallest element stored in this the `BinaryHeap`.

   For example, if the heap contains the elements `h=[4,10,20,12,12,30,24]` then `smallest(1)`returns 4, `smallest(2)` returns 10, `smallest(3)`returns 12, as does `smallest(4)` and so on. You may assume that the input k is an integer between 1 and `size()`.

Part C: The Debrief

9. [5 marks] Do the assignment 3 debrief (multiple-choice questions) on brightspace after the programming deadline has passed (and the solutions are available). The debrief questions are due 8 days after the assignment deadline (in this case, by Friday March 18th, 2:00pm.) No lates accepted.
   The debrief questions are meant to encourage retrospection on what you were meant to learn from the assignment. If you were stuck on a problem, this is an opportunity to look at the solutions and at the assignment 3 debrief, to figure out what went wrong for you, and to still learn what you were meant to learn.
   It is also an opportunity to provide feedback to Prof Alexa on certain aspects of the course so far.

Local Tests

For Parts 1, 2, 3, and 4, you can download some sample input and output files for each question as a zip file (a3-io.zip, on the Resources tab on piazza). Once you get a sense for how to test your code with these input files, write your own tests that test your program more thoroughly (**the provided tests are not exhaustive!**)

For Parts 5, 6, 7, and 8 (`BinarySearchTree`, `IntBST` and `BinaryHeap`), the main method of each file provides some tests you can and should add / modify as you go along.

Server Tests

See the "Submitting and Server Tests" section further up this document.

Tips, Tricks, and FAQs

For the most up-to-date Assignment-specific FAQ, please see the Assignment 3 FAQ post on piazza (and/or the a3 filter.)

Regarding tips and tricks, please see the Problem Solving & Assignment Tips document (which may be enhanced as the semester progresses, so check back frequently!)