Important

There are general homework guidelines you must always follow. If you fail to follow any of the following guidelines, you risk receiving a **0** for the entire assignment.

1. All submitted code must compile under **JDK 17**. This includes unused code, so don't submit extra files that don't compile. Any compile errors will result in a 0.
2. Do not include any package declarations in your classes.
3. Do not change any existing class headers, constructors, instance/global variables, or method signatures. For example, do not add `throws` to the method headers since they are not necessary.
4. Do not add additional public methods.
5. Do not use anything that would trivialize the assignment. (e.g. Don't import/use `java.util.ArrayList` for an `ArrayList` assignment. Ask if you are unsure.)
6. Always be very conscious of efficiency. Even if your method is to be $O(n)$, traversing the structure multiple times is considered inefficient unless that is absolutely required (and that case is extremely rare).
7. You are expected to implement all of the methods in this homework. Each unimplemented method will result in a deduction.
8. You must submit your source code, the `.java` files, not the compiled `.class` files.
9. Only the last submission will be graded. Make sure your last submission has **all** required files. Resubmitting will void all previous submissions.
10. After you submit your files, redownload them and run them to make sure they are what you intended to submit. You are responsible if you submit the wrong files.

Collaboration Policy

Every student is expected to read, understand and abide by the Georgia Tech Academic Honor Code.

When working on homework assignments, you **may not** directly copy code from any source (other than your own past submissions). You are welcome to collaborate with peers and consult external resources, but you **must** personally write all of the code you submit. **You must list, at the top of each file in your submission, every student with whom you collaborated and every resource you consulted while completing the assignment**.

You may not directly share any files containing assignment code with other students or post your code publicly online. If you wish to store your code online in a personal private repository, you can use Github Enterprise to do this for free.

The only code you may share is JUnit test code on a pinned post on the official course Piazza. Use JUnits from other students at your own risk; **we do not endorse them**. See each assignment's PDF for more details. If you share JUnits, they **must** be shared on the site specified in the Piazza post, and not anywhere else (including a personal GitHub account).

Violators of the collaboration policy for this course will be turned into the Office of Student Integrity.

Style and Formatting

It is important that your code is not only functional, but written clearly and with good programming style. Your code will be checked against a style checker. The style checker is provided to you, and is located on Canvas. It can be found under Files, along with instructions on how to use it. A point is deducted for every style error that occurs. If there is a discrepancy between what you wrote in accordance with good style and the style checker, then address your concerns with the Head TA.

Javadocs

Javadoc any helper methods you create in a style similar to the existing javadocs. If a method is overridden or implemented from a superclass or an interface, you may use `@Override` instead of writing javadocs. Any javadocs you write must be useful and describe the contract, parameters, and return value of the method. Random or useless javadocs added only to appease checkstyle will lose points.

Vulgar/Obscene Language

Any submission that contains profanity, vulgar, or obscene language will receive an automatic zero on the assignment. This policy applies not only to comments/javadocs, but also things like variable names.

Exceptions

When throwing exceptions, you must include a message by passing in a String as a parameter. **The message must be useful and tell the user what went wrong.** "Error", "BAD THING HAPPENED", and "fail" are not good messages. The name of the exception itself is not a good message.
For example:
Bad: `throw new IndexOutOfBoundsException("Index is out of bounds.");`
Good: `throw new IllegalArgumentException("Cannot insert null data into data structure.");`

In addition, you may not use try catch blocks to catch an exception unless you are catching an exception you have explicitly thrown yourself with the `throw new ExceptionName("Exception Message");` syntax (replacing `ExceptionName` and `Exception Message` with the actual exception name and message respectively).

Generics

If available, use the generic type of the class; do **not** use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`. Using the raw type of the class will result in a penalty.

Forbidden Statements

You may not use these in your code at any time in CS 1332.

- `package`
- `System.arraycopy()`
- `clone()`
- `assert()`
- `Arrays` class
- `Array` class

- `Thread` class
- `Collections` class
- `Collection.toArray()`
- Reflection APIs
- Inner or nested classes
- Lambda Expressions
- Method References (using the :: operator to obtain a reference to a method)

If you're not sure on whether you can use something, and it's not mentioned here or anywhere else in the homework files, just ask.

Using print statements to debug is fine, but you must remove them before submitting your code. We expect clean runs - printing to the console when we're grading will result in a penalty. If you submit print statements, we will take off points.

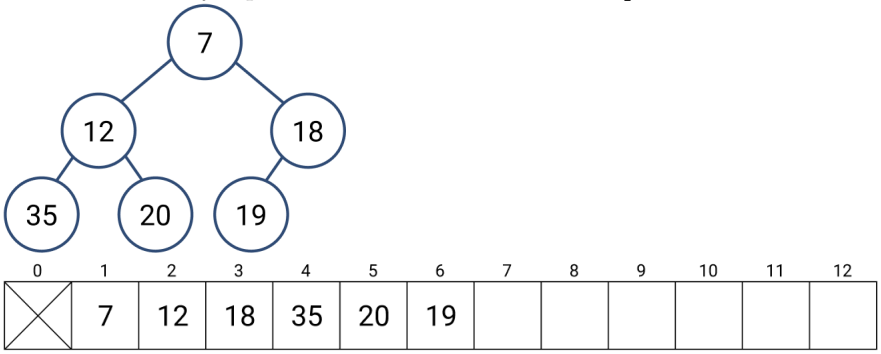
JUnits

We have provided a **very basic** set of tests for your code. These tests do not guarantee the correctness of your code (by any measure), nor do they guarantee you any grade. You may additionally post your own set of tests for others to use on the Georgia Tech GitHub as a gist. Do **NOT** post your tests on the public GitHub. There will be a link to the Georgia Tech GitHub as well as a list of JUnits other students have posted on the class Piazza.

If you need help on running JUnits, there is a guide, available on Canvas under Files, to help you run JUnits on the command line or in IntelliJ.

Heap

You are to code a `Heap`, specifically a `MinHeap`, that is backed by an array of contiguous elements. Here is a tree and array representation of the same `MinHeap`:



A `MinHeap` is a type of binary tree with two main properties.

- **Shape Property**: The tree must be complete. All levels of the tree must be full except the bottom-most level. If the bottom-most level is not full, it must be filled from left to right without any 'gaps' between nodes.
- **Order Property**: Each node's data is smaller than the data in its two children. There is no explicit relationship between sibling nodes.

These properties guarantee that the smallest element in the heap will be at the root of the heap.

Although heaps are usually classified as a type of tree, they are commonly implemented using an array due to their completeness. In your implementation, you should **leave index 0 empty and begin your heap at index 1**. This will make the arithmetic to find parents and children simpler.

To find the children of index i, the left child (if one exists) is at index $2i$ and the right child (if one exists) is at $2i + 1$. Conversely, the parent of index i is index $i/2$ (where any decimals are truncated). Recall that java automatically truncates decimals when dividing two `ints`.

You should implement two constructors for this heap. One constructor initializes an empty heap with the capacity specified by a constant in `MinHeap.java`. The other constructor should implement the BuildHeap algorithm that was taught in lecture, which is an algorithm that creates a heap in $O(n)$ time. **Simply adding the elements one by one will not receive credit** since it would be $O(n \log(n))$ in the worst case; see the javadocs for this constructor for more specifications.

You may assume that your implementation does not need to handle duplicate elements. That is, the add method will never be passed duplicates and the remove method will never have to deal with the heap having duplicates. To be clear, your implementation would most likely work even if we were to test for duplicates; however, this will help remove ambiguity surrounding grading and testing your implementation.

Unlike your BST homework, you are not required to use recursion in this assignment. Use whatever you find most intuitive - recursion, iteration, or both. However, regardless of the technique you use, make sure to meet efficiency requirements as discussed in lecture.

Grading

Here is the grading breakdown for the assignment. There are various deductions not listed that are incurred when breaking the rules listed in this PDF and in other various circumstances.

Methods:	
constructor / buildHeap	20pts
add	20pts
remove	20pts
getMin	5pts
isEmpty	5pts
clear	5pts
Other:	
Checkstyle	10pts
Efficiency	15pts
Total:	100pts

Provided

The following file(s) have been provided to you. There are several, but we've noted the ones to edit.

1. `MinHeap.java`

   This is the class in which you will implement the `MinHeap`. Feel free to add private helper methods but **do not add any new public methods, inner/nested classes, instance variables, or static variables**.

2. `MinHeapStudentTest.java`

   This is the test class that contains a set of tests covering the basic operations on the `MinHeap` class. It is not intended to be exhaustive and does not guarantee any type of grade. **Write your own tests to ensure you cover all edge cases.**

Deliverables

You must submit **all** of the following file(s) to the course Gradescope. Make sure all file(s) listed below are in each submission, as only the last submission will be graded. Make sure the filename(s) matches the filename(s) below, and that *only* the following file(s) are present. If you resubmit, be sure only one copy of each file is present in the submission. If there are multiple files, do not zip up the files before submitting; submit them all as separate files.

Once submitted, double check that it has uploaded properly on Gradescope. To do this, download your uploaded file(s) to a new folder, copy over the support file(s), recompile, and run. It is your sole responsibility to re-test your submission and discover editing oddities, upload issues, etc.

1. `MinHeap.java`