CMPT225, Fall 2026
Homework Assignment 4
Due date: Friday, November 20, 2026, 23:59

You need to implement the following classes:

- *heap.MinMaxHeap*
- *graph.Graph*

Note that all files must be under the correct package (folder)**.**
You may add more classes to your solution if necessary.

Submit all your files in ***assignment4.zip*** to CourSys. Make sure your zip file can be unzipped using the command "*unzip assignment4.zip*" in CSIL. All your solutions in the zip file must be under the **src** folder, same as in the provided *cmpret225-hw4-fall26.zip*.
That is, your zip file needs to have the following folder structure:

- */src/*
- */src/heap/MinMaxHeap.java*
- */src/graph/Graph.java*

1. See the provided example for the expected format.
2. Use the provided *check_format.sh* to check that your zip has the expected folder structure.

Discussion with others: You may discuss the assignment with your classmates/tutors (or anyone else), but writing code must be entirely your own.

References: You may use textbooks, wiki, stack overflow, geeksforgeeks, etc. If you do, specify the references in comments. Posting questions online asking for solutions is prohibited. Using AI tools for solving the assignment is prohibited.

Readability: Your code should be readable using the standard Java conventions. Add comments wherever is necessary. If needed, write helper functions or add classes to improve readability.

Compilation: Your code MUST compile in CSIL using javac.
Make sure that your code compiles without warnings/errors.
Even if you can't solve a problem completely, make sure it compiles.

Do not add main() to your solutions. The main() method will be in the test files.
Examples of such tests are provided in *TestHeap.java* and *TestGraph.java.*
The assignment will be graded mostly **automatically**, with some exceptions.

Warnings: Warnings during compilation will reduce points.
More importantly, they indicate that something is probably wrong with the code.

Testing: Test your code. Examples of tests are included. Your code will be tested using the provided tests as well as additional tests. You should create more tests to check your solution.

Good luck!

Part 1 [50 points] - `MinMaxHeap <T extends Comparable<T>>`

Write a data structure MinMaxHeap that supports the following operations.

`public MinMaxHeap()`
A default constructor.

`public void add(T item)`
Adds item to the heap in O(log(n)) time.

`public T getMin()`
Returns the minimal item in the heap in O(1) time.

`public T getMax()`
Returns the maximal item in the heap in O(1) time.

`public T removeMin()`
Removes the minimal item from the heap in O(log(n) time and returns it.

`public T removeMax()`
Removes the maximal item from the heap in O(log(n)) time and returns it.

`public int size()`
Returns the number of elements in the heap.

- Since all operations run in at most O(log(n)) time, your solutions need to run on heaps of size up to 100,000 in under 1 second
- If you are using ArrayList, you may assume all operations run in O(1) time worst case (ignore the fact that sometimes the array needs to be resized)

Part 2 [50 points] - Graph

```
public Graph(int n)
```
Creates an empty graph on n nodes. The "names" of the vertices are 0,1,..,n-1

```
public void addEdge(int i, int j)
```
Adds the edge (i,j) to the graph

```
public void removeEdge(int i, int j)
```
Removes the edge (i,j) from the graph

```
public boolean areAdjacent(int i, int j)
```
Returns true if (i,j) is an edge in the graph, and returns false otherwise

```
public int degree(int i)
```
Returns the degree of i

```
public Iterator<Integer> neighboursIterator(int i)
```
Returns an iterator that outputs the neighbors of i in the increasing order.
Assumption: the graph is not modified during the use of the iterator

```
public int numberOfVertices(int i)
```
Returns the number of vertices in the graph

```
public int numberOfEdges(int i)
```
Returns the number of edges in the graph

```
public int distance(int i, int j)
```
Returns the distance between i and j in the graph
If j is not reachable from i, return -1

```
public static Graph generateRandomGraph(int n, double p)
```
Creates a random graph on n vertices such that each edge appears in the graph with probability p independently of all others.
You may use Math.random() to generate a random number between 0 and 1.