CS3460: Data Structures Spring 2021
Assignment 07: Fun with Priority Queues
Due Date: Monday, April 12; 23:00 hrs
Total Points: 20

In this assignment, we will build a max-priority queue data structure implemented as a binary heap, and then use it to perform numerical computations. Specifically, we will compute an approximate solution to an area under a curve.

1. **Max-Heap (10 points):** A max-heap is an implementation of the max-priority queue data structure that supports two fundamental operations.
   (a) `insert(k)`: This inserts a new element `k` into the heap.
   (b) `remove_max()`: Returns and removes the max element in the heap. Returns `null` if the heap is empty.

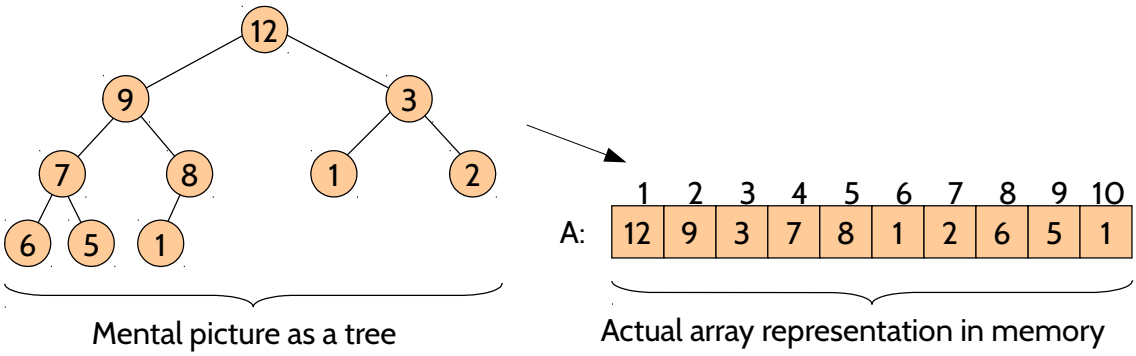

Figure 1: A mental picture of a max-heap of integers and its corresponding array representation.

The max-heap can be encoded as an array and visualized as a tree that is complete at every level except possibly the last (see figure above). The tree also satisfies the max-heap property – if element with key k is stored at the root, then both its left and right subtrees contain keys at most k. The elements of the heap are stored in an array one level at a time from left to right. This allows us to compute the parent, left and right child of every node by performing arithmetic on the indices. For example, if the array uses 1-based indexing, then the following arithmetic operations work.

(a) `parent(i) = i / 2`; computes the index of the parent of a heap element at position `i`.
(b) `left(i) = 2 * i`; computes the index of the left child of a heap element at position `i`.
(c) `right(i) = 2 * i + 1`; computes the index of the right child of a heap element at position `i`.

The heap operations `insert` and `remove_max` can be performed using other operations like `siftdown(i)` and `siftup(i)`, which fix violations of the heap property for an element at position `i` with any of its children and parent respectively, by repeatedly swapping with the smallest child (for `siftdown`) or parent (for `siftup`). To perform `insert`, we put the new elements at the next available position in the array and call `siftup`, and for `remove_max`, we swap the root with the last element and call `siftdown` at the root. All operations in a binary heap take only $O(\log n)$ time.

Getting Started: In this problem, we will be storing a set of intervals as a max-heap keyed primarily on their lengths. Each interval is represented using its starting and ending points on the number line. The file `Interval.java` is already implemented, and includes an argument constructor, several accessor methods, and a `compareTo` method that is used to compare two `Interval` objects. The comparison function compares the two `Interval` objects based on their lengths first, and if both lengths are equal, then compares their starting points. Note that when creating an interval with starting point a_i and ending point b_i, the class assumes that the constructor will always be called with $a_i \leq b_i$. Since these are stored as `double` variables, which could potentially have floating point round off errors, the comparison function checks if the two doubles being compared are close together up to a predetermined error. You are not required to modify this class.

The PriorityQueue Class: This class currently includes an array that encodes a max-heap, and keeps track of the size of the allocated buffer and the number of elements stored. It needs to expand the buffer if a new element is inserted into a full buffer. It also includes a `print` function that prints the contents of the array. Please fill the two main priority queue methods – `insert(k)`, that inserts a new interval `k` into the heap, and `remove_max`, that returns the largest (based on the comparison function) interval in the heap. You may include several private methods like `siftup`, `siftdown`, `parent`, etc. to help with your implementation. Please test all your methods thoroughly.

2. **Area Under a Curve (10 points):** In this problem we will use the max-heap data structure developed in the last problem to compute an area under a curve. Given some function $f(x)$, we can compute the area under $f(x)$ between two points $x = a$ and $x = b$ by integrating the function $\int_{x=a}^{b} f(x)\,\mathrm{d}x$. Since we are trying to compute the area of a continuous function using a computer with limited precision and floating-point round off errors, we will try to get an approximation to the area under a given curve. The study of finding approximate solutions to problems when correct or optimal solutions may not be possible due to limitations of a modern computer opens the fascinating field of numerical approximation algorithms. Such algorithms form the bulk of implementations of math libraries in most programming languages, including all trigonometric functions, and mathematical constants like π and e.

   To find the area under a curve $f(x)$ between $x = a$ and $x = b$, we first break the interval (a, b) into several smaller intervals $(a, x_1), (x_1, x_2), \ldots (x_{n-1}, x_n)$, where $x_i = a + i\,\mathrm{d}x$ for some arbitrarily small value of dx, and then add up areas of respective rectangles (see Figure 2 (i) below). The area of a specific rectangle that starts at point x_i can be computed as $(x_i + \mathrm{d}x - x_i)f(x_i + \mathrm{d}x) = \mathrm{d}x f(x_i + \mathrm{d}x)$, where $\mathrm{d}x$ is chosen to be arbitrarily small. However, the choice of $\mathrm{d}x$ impacts the area computed, as smaller its value, more approximate is the area (see Figure 2 (ii)).

   In this problem, we can use a priority queue to maintain a set of intervals within the given region (a, b). Initially, we only have the interval (a, b) with area $c = (b - a) * f(b)$. In

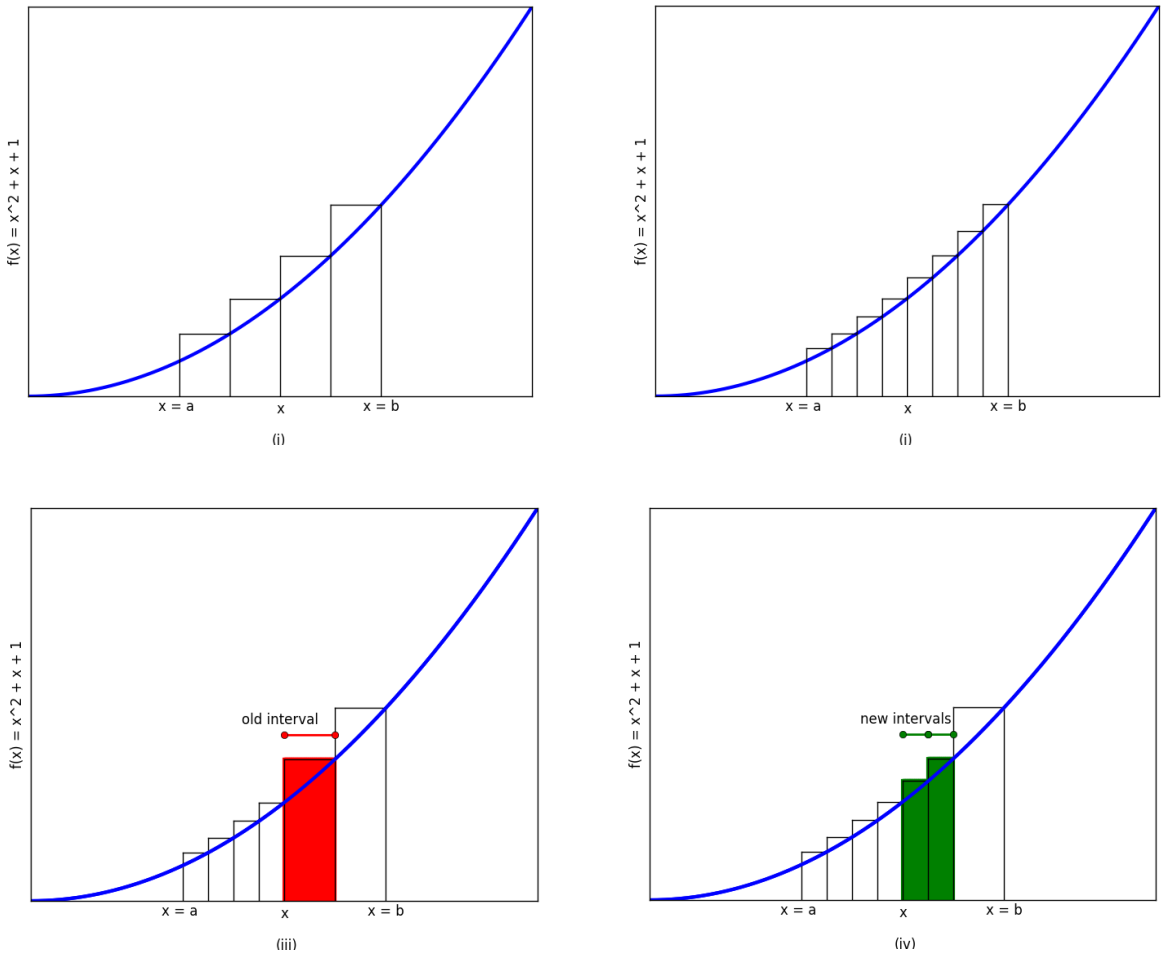

Figure 2: (i) Computing the area under the curve $f(x)$ between two points $x = a$ and $x = b$. (ii) Using smaller values for dx gives better approximations to the area. (iii) The largest interval is broken, and its corresponding area shown in red removed, and (iv) two new interval added, and their corresponding areas shown in greed are added.

every iteration of the algorithm, we can take the largest interval (m, n) and break it into two smaller half-sized intervals (m, p) and (p, n), where $p = \frac{m+n}{2}$. The new area can be computed by subtracting out the area of the rectangle formed from the old interval, and adding in the areas formed formed from the new intervals (see Figures 2 (iii) and (iv)). So the new area d is computed as

$$d = c - (n - m)f(n) + (p - m)f(p) + (n - p)f(n).$$

If the new area computed does not significantly change from the old area, then this gives a good stopping point for our algorithm. Since we are using doubles, we can check for this

condition using $|d - c| \leq e$, for some precision e. Otherwise, we insert our two new intervals into the priority queue, update the current area $c = d$, and continue with the next iteration of the algorithm. When the algorithm stops, c will hold the current area computed.

Implementation, Input and Output: Some startup code is already provided in `AreaUnderACurve.java`. Please complete the function `computeArea(a, b)` that returns the area under the curve `f(x)` between $x = a$ and $x = b$. The function f is also defined in the program. A sample input and output is shown below.

```
Sample Input and Output:

We have the function f(x) = x^2 + x + 1.
Please enter lower value a: 12
Please enter upper value b: 15

An approximation for the area under the curve f(x)
between a = 12.0 and b = 15.0 is 592.500961300582
```

Please thoroughly test your program for other values of a and b.

Submission: Please submit all the files as a single zip archive `h07.zip` through ASULearn. The zip archive should only contain the individual files of the assignment, and these files should not be inside folders. Moreover, please do not include other extraneous files. Only include all files that belong to your solution.

Input/ Output Instructions: For all programs, until and otherwise stated, we will be taking the input from standard input (`System.in`) and will be sending the output to standard output (`System.out`). Since we will be using an automatic grading system, please make sure that your output format exactly matches the description above. So make sure that there are no extraneous output in terms of spaces, newlines and other characters.

Notes on Coding: Please do not include user-defined packages in your code. Your code should run in the Unix/Linux machine using the commands `javac` and `java`.

Please note that you are **not allowed to use Java Collections** which contain pre-defined libraries for many of the data structures that we learn in class. The purpose of these assignments is to build these data structures from first principles. Programs that includes these objects will receive 0 points.