# EECS 233 Homework 7 (Minor typo fixed in 2(c) on October 28 at 10:30 AM.)

**General requirements:**

- Due at 11:00 PM on the posted due date.
- Include your name and network ID as a comment at the top of all of your programs.
- Create a typed document (.docx or .pdf) for the report with your name and network ID at the top.
- Upload your document and all .java files as a .zip file to Canvas. Do not use other formats such as .rar.
- All work should be your own, as explained in the Academic Integrity policy from the syllabus. File sharing is prohibited.

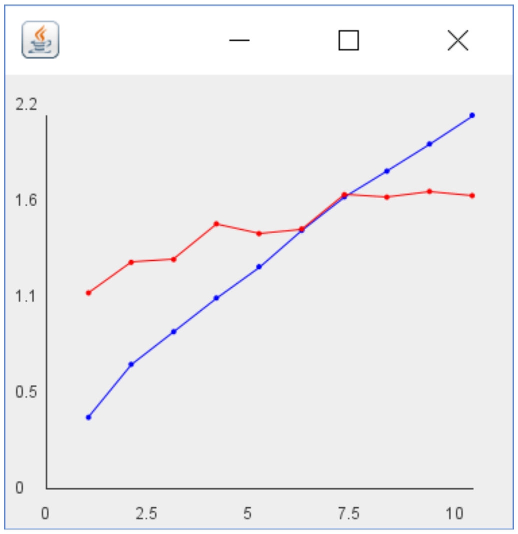
*WARNING: There are various implementations of heaps available on the Internet. However, students are required to use only the techniques that have been presented in class and in the following instructions. Additionally, students will be tested on their understanding of these particular techniques. Developing the programs on your own will maximize your understanding.*

1. In your report, draw the state of a binary heap after each of the following (in order a – n):
    a. Insert 1
    b. Insert 3
    c. Insert 2
    d. Insert 4
    e. Insert 6
    f. Insert 7
    g. Insert 8
    h. Insert 5
    i. Remove maximum
    j. Remove maximum
    k. Remove maximum
    l. Remove maximum
    m. Remove maximum
    n. Remove maximum

2. Revise Heap.java such that it implements a class for a priority heap with the following methods. Your solution is not required to work for duplicate values.
    a. Complete add() to add a value using the algorithm described in section 10.1 of the textbook and discussed in class. You are free to make trivial changes, such as with variable names, but do not deviate from the core algorithm.
    b. Complete removeMax() to remove the root value using the algorithm described in section 10.1 of the textbook and discussed in class. You are free to make trivial changes, such as with variable names, but do not deviate from the core algorithm.
    c. main() should create the tree produced in problem #1 above and print the final values using the print() method provided. You are free to change the output format for print() if you wish. Include the output in your report.

3. Perform the following experiment to compare the run time for IntArrayBag.java (from HW #3) to the run time for Heap.java. Using at least 5 evenly spaced N values, create an initial data structure with N-1 values. Note that you will not compute the time for these operations. Then compute the run time for a fixed number of the operations described below. Using any software you choose, create a single graph with the run times for each of the two data structures below (2 lines on one graph). Provide a legend. **Print the N values and run times, and include them in your report.** EXTRA CREDIT (10 points): Use the program Grapher2.java that is provided to automatically create the graphs (see HW #1 solutions for an example of using it).
    a. Using the IntArrayBag class, repeatedly add() and remove() the value N a fixed number of times. Note that we are adding N so that the "remove" operation will require O(N) time.
    b. Using the Heap class from problem #2, repeatedly add() the value N and removeMax() a fixed number of times. Note that we are adding N so that the "removeMax" operation will always remove the root.

    Below is an example of how your graph might look. As in HW #1, you can use indexes (1, 2, ...) for the x-axis values instead of actual N values to visually compare the algorithms, since the N values may be different.



blue = IntArrayBag, red = Heap
(y-axis in seconds)

*Tip (why N-1?):* The goal is to add/remove a maximum value to the data, so that's why you are instructed to first create structure with N-1 values.

*Tip (why is this different from HW #3?):* In HW #3, we computed the time for building the data structure. However, this wasn't a careful way to do it because the size of the data structure was changing with each N value. In this assignment, we are trying to be more careful by excluding the building time.

*Tip (good values for N and number of operations):* As in HW #1, it is easiest to compare the trends in the graph if the times for IntArrayBag and Heap are similar. You may have to play with different N values to find what works best. The "*fixed number of operations*" refers to how many times you perform the add/remove, which should be large enough to get some reasonable times. You may find that you get the best results by using small N values (less than 1,000) and a large number of operations (over $10^6$). However, other combinations may work too.

4. What is the expected Big-O time of the following operations? Explain your reasoning in 1 – 3 sentences.
   a. add(X) for the IntArrayBag class, where X is any value.
   b. remove(X) for the IntArrayBag class, where X is the last value in the array.
   c. The sum of add(X) and remove(X) for the IntArrayBag class, where X is unique.
   d. add(X) for the Heap class, where X is larger than all other values in the heap.
   e. removeMax() for the Heap class.
   f. The sum of add(X) and removeMax() for the Heap class, where X is larger than all other values in the heap.

5. How do the experimental results from problem #3 compare to the expected results from problem #4?

*Rubric:*

| **Item** | **Points** |
|---|---|
| 1. Each state (14 total, 2 points for each operation) | 28 |
| 2. General programming requirements | 4 |
| 2(a). Inserting value | 5 |
| 2(a). Moving value up | 5 |
| 2(b). Inserting value | 5 |
| 2(b). Moving value down | 5 |

| | |
|---|---|
| 2(c). Creating tree | 5 |
| 3(a). Building data structure | 4 |
| 3(a). Using add/remove and timing | 4 |
| 3(a). Reporting results | 4 |
| 3(b). Building data structure | 5 |
| 3(b). Using add/removeMax and timing | 4 |
| 3(b). Reporting results | 5 |
| 4. Six questions. 2 points each. | 12 |
| 5. Any reasonable answer will receive full credit. | 5 |
| *Total* | 100 |