COP-3530 Data Structures

Programming Assignment 4:
Heaps

Due Date: July 18 at 11:59 PM

This assignment has two parts: 1) implementing a binary heap 2) and comparing the efficiency of the implemented heap class with the standard java.util.PriorityQueue⟨⟩ in a given scenario when merging 25 sorted lists.

1 Constructing Binary Heap

In this section, you need to first implement a class MyBinaryHeap based on Chapter 8 of the textbook. Your class should be able to store a heap of elements with keys of type long integers. Also, the heap should be stored in the form of a linear array (either an array or an ArrayList) inside the class. Your class should have two key methods: insert and deleteMin. As mentioned during our Zoom class, insertion of a key to the heap can be done by simply adding an element to the end of array and then applying a percolate-up operation. Also, deleteMin requires you to:

- swap the first and last elements of the array;
- shorten the array by removing the last element;
- and finally apply a percolate-down operation.

2 Efficiency Analysis of the Implemented MyBinaryHeap

In this part, you need to write a program that compares the efficiency of MyBinaryHeap iplemented in part 1 with the standard *java.util.PriorityQueue*⟨⟩. To do so, you need to find and compare the running times of the following program for both data structures:

Test Scenario: Merging the 25 Rows of in.csv File into One Sorted List

Program below merges 25 sorted lists into one sorted list in an efficient way using heap data structure. The 25 sorted lists are given in a table with 25 rows given by "in.csv" file available on Canvas.

1. long startTime = System.nanoTime();
2. long totalTime = 0;
3. Create a new Scanner (new File("... *local address* ... \\"+"in.csv")) (in.csv is available on Canvas).
4. For $i = 1, 2, \ldots, 25$,
   (a) Create an array and call it a_i.
   (b) Read the i^{th} row and split it into long integers (use a scanner with comma delimiter). Store all the long integers into a_i.
5. startTime = System.nanoTime();
6. build a heap/priorityQueue of size 25 with the first elements of the 25 arrays.
7. For $i = 1$ to $250,000 - 25$
   (a) extract the element (x) stored in the heap/priorityQueue with minimum key value.
   (b) add x to an ArrayList which will eventually store the sorted list of all elements.
   (c) insert the next element of the array from which x is taken from to the heap/priorityQueue.
8. totalTime+= (System.nanoTime()-startTime);
9. System.out.println(n + ", " + totalTime);

3 Submissions

You need to submit a *.zip* file compressing the following folder:

- all the Java source file(s).
- A readme.txt file containing the program output and short paragraph comparing the efficiency of binary heap and priority Queue.