1 Assignment 2

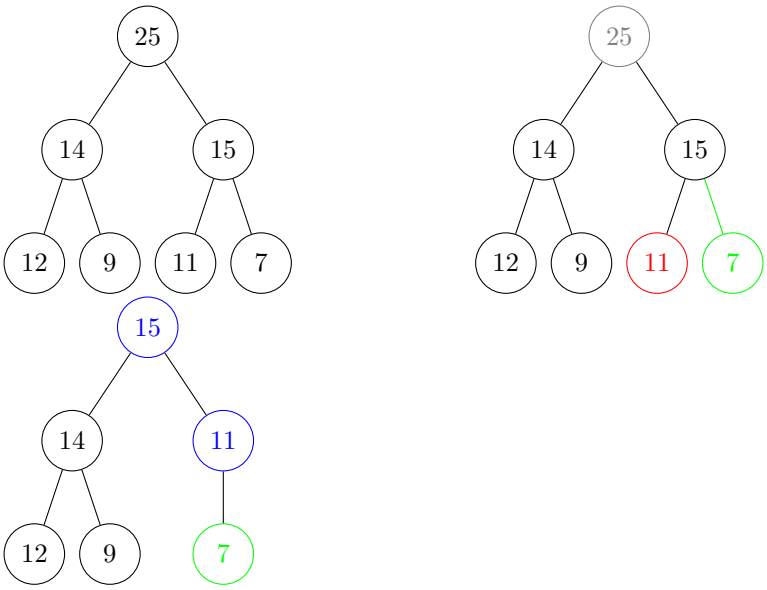
Submit source code and running instructions to EAS[1]. Do it in Java. Submit the textual/diagram components in a single word/pdf file, providing the assignment numbering to help your marker. Do not use java's search methods! Do not use Java's Collections or Subclasses, code your own.

Please put all files in the default pacakge. This isn't good design, but it *is* easier to mark.

This assignment should use Command-Line. Not Scanner, not anything else. Command-Line input. Learn what String[] args are for in the main method! Similar, but a bit smarter, to what you may have used in C++.

Posted: Wednesday, May 16^{th}
Due: Monday, May 28^{th}
Grade: 5%

A bottom-up heap sort does not take the next value, place it at the top of the heap and sift it down, it finds where that value goes and pushes all its new ancestors up until the next value to be chosen is at the top of the heap.



Consider the above heap. 25 will be placed in the sorted list, and 7 is the next value we will consider (green). A bottom up heap would know that 11(red) is the leaf of the "prime path"[2], so 7 is checked against it, and then each parent in turn until a bigger value is found or it makes it to the root (which would only happens in one special case). As it turns out, **hint:** it is not uncommon for there to be very few comparisons.

[1] https://fis.encs.concordia.ca/eas/

[2] An arbitrary name to identify the path from the root, where each node in the path is the bigger of the two children until a leaf is found.

1. Draw a class diagram for your HeapSort class. Provide a text description and very simple pseudocode to make it clear how your class works to sort values. You can omit discussion of the trace/timing.

2. HeapSort

   (a) Implement a bottom-up Heapsort algorithm that takes an array as input

      i. This file should be called HeapSort.java

      ii. This class should have two sort methods:
      public static void sort(int[] input)
      public static void sort(int[] input, traceStep)

   (b) Write classes that are able to generate test inputs of size 10, 100, 10000, 1000000 (or any size)

      i. One file should be RandomGen.java

      ii. One file should be FixedGen.java

      iii. RandomGen should generate random integers of a uniform distribution.

      iv. FixedGen should always generate a fixed descending input.

   (c) Make a driver that sorts values from your input

      i. This file should be called HSDriver.java

      ii. This file should output the run-time in either ns or μs

      iii. it should accept command-line as follows:
      java HSDriver <gen> <length> <trace-step> [<seed>]

         A. <gen> is either RandomGen or FixedGen

         B. <length> is the number of ints to be sorted in the input array

         C. after <trace-step> values have been put into the sorted list, output the full array. Ignore if -1, show heapified array if 0.

         D. <seed> is an optional argument (it might not be passed) that lets you repeat the random seed for RandomGen (but is ignored by FixedGen)

   (d) Record performance times of runs for each input size specified in 2b for the HeapSort implemented in 2a using RandomGen.

HeapSort Sort Example

```
%java HSDriver RandomGen 6 2
sorting: 99 37 17 5 12 33
trace at step 2: 33 17 12 5 37 99
result: 5 12 17 33 37 99
completed in 1291555ns
```

3. In clear, natural language, describe the performance differences between this sort and a top-down HeapSort. Try to correlate this with the underlying mechanism.

   (a) This textual response should be no more than 8 lines / 80 words.

4. In clear, natural language, describe your approach the bottom-up HeapSort. Indicate changes in asymptotic complexity for either time or space as needed. Indicate briefly how you implemented it.

   (a) This textual response should be no more than 10 lines / 100 words.