CS566 – Summer 2017
Programming Assignment 1

Assigned: 6/14
Due: 6/21

The goal of this assignment is to give students an opportunity to study/verify the running time analysis of BUILD-MAX-HEAP program.

As we studied in the class, a cursory, informal analysis gives us an upper bound of $O(n \lg n)$ but a detailed analysis gives us a tighter upper bound of $O(n)$.

Your task it to implement the BUILD-MAX-HEAP, run it with input of different sizes, measure the running times for different input sizes, and discuss the result. If the theoretical, detailed analysis holds, you should be able to observe that the running time grows linearly (or approximately linearly) as the input size increases.

For this assignment, you need to measure the running times of the program. But, you can't measure the actual execution time of the program accurately because there are other processes running on your computer when you run the algorithms. However, you can estimate the running times for the purpose of identifying the trend of running times as the input size increases. When you run your program, you may want to disable your internet connection and you may want to stop any user-initiated programs and any other unnecessary services. The way you measure the running time (which is an elapsed time) of your program is to get the *time* before starting BUILD-MAX-HEAP and get another *time* after BUILD-MAX-HEAP terminates and compute the difference between these two *time*s. A typical programing language has a built-in function with which you can get the time and you must use this built-in function.

You must implement the pseudocodes of BUILD-MAX-HEAP and MAX-HEPIFY given in the textbook, and then implement the following pseudocode that measures the running times:

```
BUILD-MAX-HEAP-ANALYSIS

    // You will be given 10 input data with different sizes.
    // The following is for one input size.
    // So, you must repeat this this pseudocode 10 times, one for each input size.

    read input data from input file and make an array A
    totalElapsedTime = 0
    for i = 1 to 10 {
        shuffle A // randomize the order of elements in A
        startTime = System.currentTimeMillis(); // this is Java code
                                                 // if you use other language, change this
        call BUILD-MAX-HEAP(A)
        endTime = System.currentTimeMillis();  // this is Java code
```

```
                                                    // if you use other language, change this
        elapsedTime = endTime – startTime //convert this to an appropriate time unit
                                                    // (e.g., seconds or milliseconds as needed)
        totalElapsedTime = totalElapsedTime + elapsedTime
    } // end for
    averageElapsedTime = totalElapsedTime / 10
    print n and averageElapsedTime // n is the number of elements in A
```

Note: When you implement BUILD-MAX-HEAP, you may want to include a code segment at the end of the program that prints the first 15 nodes in the heap in the order they are stored in the array A. You can use this output to verify that your program builds a heap correctly.

Input data

The following ten input files are posted on the class web page as a single zip file. Each file has randomly generated integers. The input size is included as a part of each file name.

random-10000.txt
random-20000.txt
random-30000.txt
random-40000.txt
random-50000.txt
random-60000.txt
random-70000.txt
random-80000.txt
random-90000.txt
random-100000.txt

Experiment and summary of result

Run your program 10 times, as described above, on each of the above ten input data, collect 10 average elapsed times, and summarize the result in a table and as a graph.

A table should look like the following:

Table. Running Times for Different Input Sizes

n	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
Running time (average elapsed time)										

Then, plot a line graph that shows the running time (on Y-axis) vs. input size (on X-axis).

<u>**Deliverable**</u>

1. Electronic submission:
   a. If you used java, archive all your source code files and class files into a single file (e.g., a zip file) and upload it to the class web page. Name the archive file as *LastaName_FirstName*_p1.EXT, where EXT is an appropriate extension (e.g., zip or rar).
   b. If you used C/C++/C#, you need to submit both the source code and the executable files. Name them as *LastaName_FirstName*_p1.EXT (here, EXT is an appropriate source code extension) and *LastaName_FirstName*_p1.exe, respectively. Again, archive them all into a single file (follow the naming convention of the archive file described above for java programs) and upload it to the class web page. I will test your program by running the executable file from the command line. So, before submission, you may want to run your executable file from the command line to see that it runs OK.
   d. If you used another programming language (e.g., perl or python), indicate it in your report and let me know which version of the language you used, and use the default extension for the source code file name.
2. Hardcopy documentation
   a. A cover page.
   b. Include the table and the graph.
   c. Discussion of your result.

Important:

1. In the hardcopy submission, explain in detail how to compile and run your program. Don't forget to indicate how input file names are recognized by your program (e.g., I must pass the input file name as an argument, or input file name is hardcoded, etc.)
2. Include your name at the beginning of each file (e.g., as a comment in a code).