BLG381E, Advanced Data Structures, Fall 2012
Project 2

Handed out: 31.10.2012
Due: 13.11.2012, until 5 PM

Problem: You need to implement the Heap Sort algorithm and priority queue operations in this project. You also need to compare running time and space requirements of Heap Sort with Insertion and Merge Sort.

Insertion Sort and Merge Sort algorithms are already provided in BLG381E_project2.cpp. You can directly use these sort functions. You need to add necessary template functions for Heap Sort **[40 pts.]** as well as following priority queue operations for heaps **[20 pts.]**:

- $INSERT(S, x)$: insert element x into set S
- $MAXIMUM(S)$: return element of S with the largest key
- $EXTRACT - MAX(S)$: remove and return element of S with the largest key
- $INCREASE - KEY(S, x, k)$: increase value of x's key to k, where k is at least as large as x's current key value

You also need to update main program accordingly to test your functions.

Detailed Instructions:

- All your code must be written in C++ using object oriented approach and able to compile and run on Linux using g++ compiler.
- Submissions will be done through the Ninova system. You must submit all your program and header files. You must also submit a softcopy report.
- In your report, you are also expected to analyze and compare the running times and space requirements of 3 sorting algorithms.
    - **a.** Give the asymptotic upper bound on the running time for of heap sort (which you can find in the lecture slides) and show that your implementation of the algorithm fit that value. **[10 pts.]**
    - **b.** Run each three search functions for 10 times for each different value of N as $\{10, 20, 50, 100, 1000, 5000, 10000\}$ and get the average time of execution for each value of N.
    - **c.** You can use the clock() function under ctime library for calculating time of execution for your search functions. Refer this link for more details. For precise calculations, you can keep time to sort the same unsorted array for k times and divide the calculated duration by k. (You can choose $k = 100$ or $k = 1000$, for example.)
    - **d.** After calculating execution times, you need to prepare three line plots in order to visualize the runtime complexity of the sorting algorithms for different values of N. **[20 pts.]**
    - **e.** Calculate how much temporary storage space these 3 algorithms would require to sort N elements. Suppose that you are developing a program for an embedded system where you have data storage limitations. Which sorting algorithm would you prefer? **[20 pts.]**