UNIVERSITY OF MINNESOTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
4041: ALGORITHMS AND DATA STRUCTURES
FALL 2017

PROGRAMMING ASSIGNMENT 1 :
Assigned: 10/7/17 Due: 10/15/17 at 11:55pm (submit via moodle)

Make sure you apply your code to different inputs to test the results. Copying code from others or the internet constitutes cheating and the University policies will be followed. For each problem hand in these three things (after putting them in a single .zip file):

1. The code you create.
2. A "readme.txt" file explaining any how to run your code and any assumptions that you made. If you were not able to complete a problem, describe your approach here.
3. A "run.sh" file that will both compile (if necessary) and run your program on the input text file. (See: sample.zip for examples on how to use .sh files)

We will use files to both send input into your program and to read the output of your program. We should be able to run your program by saying "./run.sh input.txt" for some input text file. In each part, your code should create a text file named "output.txt" with your solution. When printing multiple things, separate them by spaces (except after the last thing). To get your run.sh to work, you might need to type: chmod 700 run.sh

Please ensure your code is easily readable and well structured. If the code is obfuscated, has poorly named variables/functions, not well documented, etc., then points will be taken off. You may choose to code in any of these languages: C/C++, Java, or Python. For each problem the point breakdown will be roughly: 65% correct output, 15% readme.txt and .sh file sufficient, and 20% coding style. Your code must work on a caselabs machine. We will test it on the machine: csel-kh1260-13.cselabs.umn.edu

Problem 1. (20 points)
Implement a priority queue. In particular, you are asked to simulate a waiting line that forms in front of a bank teller in a typical bank. Different customers have different priorities. A priority queue should support the following functions (you must build these functions):

- Insert(S,x) inserts the element x into the set S.
- Maximum(S) returns the element of S with the largest key.
- Extract_Max(S) removes and returns the element of S with the largest key.

Your program should be based on a heap structure. The input must accept the different customers and their priorities (Customer Name and Customer Priority are two attributes that you must use). As an output, the program should produce a list of the customers based on their priorities (decreasing order). You may assume priorities are integers.

This is a sample output.txt file (spaces separating customer names (no space after final name)):

```
Alan Paul
```

... for this input file (one customer and priority per line in this format):

```
Paul : 2
Alan : 5
```

You can break ties in any order if their priorities are the same.

Problem 2. (30 points)
Build a data compression system using the Huffman's algorithm. In other other words, you are given a sequence of alphabet characters and you are asked to construct the Huffman's code. Your "output.txt" file should have the bit representation for every character that appears in the file (and should not contain characters that are NOT present in the input file). All characters in the input file will be normal English lower-case letters.

Sample output.txt (any valid Huffman code in this format):

```
a:0
b:101
c:100
d:111
e:1101
f:1100
```

Sample input.txt (all characters on a single line with no spaces bewteen):

```
ffffeeeeeeeefccccccccccccbbbbbbbbbbbbbddddddddddddddddaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Problem 3. (20 points)
Use selection (i.e. average runtime of O(n)) to find the ith smallest number. The first number in the text file (on its own line) will be "i". The second line will the the array of number which you want to find the ith smallest. You may assume the array contains only integers.

Sample output.txt (single number only):

```
4
```

Sample input.txt (first line will be the integer "i", second line will be array with spaces between elements (no space after last)):

```
7
4 5 8 2 4 7 5 0 8 2 3 9 23 48 -12 49
```

Problem 4. (30 points)
Assume you have a list of names that you want to sort in alphabetical order. Use bucket sort to solve this problem (i.e. the program should run in average case O(n)). However, use ***bucket sort recursively*** to ensure that each of the final buckets that are sorted via insertion sort contain no more than 10 elements/names. The way you would do this is when sorting each bucket, if there are more than 10 elements/names in the bucket then you would create sub-buckets and run a mini-bucket sort on only the elements in this large bucket. You may assume there are no spaces in the names and only contain normal English alphabet characters. Also the first letter of names will be capitalized.

Sample output.txt (sorted name array separated by spaces (no space after final name)):

```
Candy Mahesh Rishi Svetovid
```

Sample input.txt (unsorted names separated by spaces (no space after final name)):

```
Svetovid Candy Rishi Mahesh
```