UNIVERSITY OF MINNESOTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
4041: ALGORITHMS AND DATA STRUCTURES
SPRING 2021

PROGRAMMING ASSIGNMENT 1 :
Assigned: 2/11/21 Due: 2/24/21 at 11:55pm (submit via gradescope)

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 submit these four things:

1. The code you create.
2. A "readme.txt" file explaining how your code works (though you should comment as well). If you were not able to complete a problem, describe your approach here.
3. A "run.sh" which runs the code (just put the commands to start your program in here).
4. A "make.sh" file that will both compile (if necessary) and anything else you need to do before running the code with "run.sh".

The "readme.txt" (2), "run.sh" (3) and "make.sh" (4) should all not be in sub-folders, but rather directly uploaded (you may have your "run.sh" and "make.sh" use sub-folders if you wish). You can either directly upload multiple files to gradescope or upload a zip with your files, but make sure you do not zip a folder (as then the "sh" files will be in a sub-folder).

You cannot use any advanced libraries (basic mathematical or input/output ones are acceptable), but rather need to code all parts from scratch. We will use files to both send input into your program and to read the output of your program. You can assume the the file "input.txt" in the same folder as the "run.sh" is where the input will be that you need to read. In each part, your code should create a text file named "output.txt" with your solution. Please make sure you exactly follow the formats shown for each problem.

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, Python or Rust. If there is some other language you wish to use, check with the professor (me: jparker@cs.umn.edu) and we will see if this is viable. For each problem the point breakdown will be roughly: 70% correct output, 10% documentation (including readme.txt and .sh file sufficient), and 20% coding style. The code will be tested on gradescope's machines, which are linux based (and very similar to the linux machines in Keller).

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 each of these three commands in the format shown below (and a "quit" command to indicate we are done, though you could also just stop at the end of the file). As an output, should show each "max" command on a

separate line, then the remaining heap (in reverse sorted order) when the program quits. You may assume priorities are integers.

This is a sample output.txt file (spaces separating customer names):

```
Alan
Sally
Sally Alan Paul
```

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

```
insert Paul 2
insert Alan 5
max
insert Sue 999
insert Sally 99
extract
max
quit
```

(Note: 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 between):

```
ffffeeeeeeeeefccccccccccccbbbbbbbbbbbbbddddddddddddddddddaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

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 only 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
```